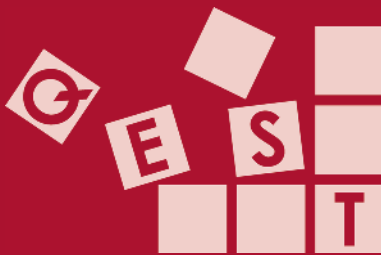


Annabelle McIver  
Andras Horvath (Eds.)

LNCS 11024

# Quantitative Evaluation of Systems

15th International Conference, QEST 2018  
Beijing, China, September 4–7, 2018  
Proceedings



Springer

*Commenced Publication in 1973*

Founding and Former Series Editors:

Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

## Editorial Board

David Hutchison

*Lancaster University, Lancaster, UK*

Takeo Kanade

*Carnegie Mellon University, Pittsburgh, PA, USA*

Josef Kittler

*University of Surrey, Guildford, UK*

Jon M. Kleinberg

*Cornell University, Ithaca, NY, USA*

Friedemann Mattern

*ETH Zurich, Zurich, Switzerland*

John C. Mitchell

*Stanford University, Stanford, CA, USA*

Moni Naor

*Weizmann Institute of Science, Rehovot, Israel*

C. Pandu Rangan

*Indian Institute of Technology Madras, Chennai, India*

Bernhard Steffen

*TU Dortmund University, Dortmund, Germany*

Demetri Terzopoulos

*University of California, Los Angeles, CA, USA*

Doug Tygar

*University of California, Berkeley, CA, USA*

Gerhard Weikum

*Max Planck Institute for Informatics, Saarbrücken, Germany*

More information about this series at <http://www.springer.com/series/7407>

Annabelle McIver · Andras Horvath (Eds.)


# Quantitative Evaluation of Systems

15th International Conference, QEST 2018  
Beijing, China, September 4–7, 2018  
Proceedings



*Editors*

Annabelle McIver  
Macquarie University  
Sydney, NSW  
Australia

Andras Horvath   
University of Turin  
Turin  
Italy

ISSN 0302-9743                      ISSN 1611-3349 (electronic)  
Lecture Notes in Computer Science  
ISBN 978-3-319-99153-5              ISBN 978-3-319-99154-2 (eBook)  
<https://doi.org/10.1007/978-3-319-99154-2>

Library of Congress Control Number: 2018951431

LNCS Sublibrary: SL1 – Theoretical Computer Science and General Issues

© Springer Nature Switzerland AG 2018

This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG  
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

This volume contains the papers presented at QEST 2018: 15th International Conference on Quantitative Evaluation of Systems held during September 4–7, 2018, in Beijing.

QEST is a leading forum on quantitative evaluation and verification of computer systems and networks, through stochastic models and measurements. This year's QEST was part of the CONFESTA event, which brought together the Conference on Concurrency Theory (CONCUR), the International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS), the Symposium on Dependable Software Engineering Theories, Tools and Applications (SETTA), as well as QEST. CONFESTA also included workshops and tutorials before and after these major conferences.

As one of the premier fora for research on quantitative system evaluation and verification of computer systems and networks, QEST covers topics including classic measures involving performance and reliability, as well as quantification of properties that are classically qualitative, such as safety, correctness, and security. QEST welcomes measurement-based studies as well as analytic studies, diversity in the model formalisms and methodologies employed, as well as development of new formalisms and methodologies. QEST also has a tradition in presenting case studies, highlighting the role of quantitative evaluation in the design of systems, where the notion of system is broad. Systems of interest include computer hardware and software architectures, communication systems, embedded systems, infrastructural systems, and biological systems. Moreover, tools for supporting the practical application of research results in all of the aforementioned areas are also of interest to QEST. In short, QEST aims to encourage all aspects of work centered around creating a sound methodological basis for assessing and designing systems using quantitative means.

This year's edition of QEST emphasized two frontier topics in research: quantitative information flow for security and industrial formal methods. Each topic was represented by two outstanding keynote speakers. Kostas Chatzikokolakis (École Polytechnique, France) gave a talk on "Quantifying Leakage and the Science of Quantitative Information Flow" and Mark Wallace (Monash University, Australia) spoke on "Constraints and the 4th Industrial Revolution." A special tutorial session providing an overview of machine learning was given by Mark Dras (Macquarie University, Australia). The program also included a joint keynote talk by Moshe Vardi (Rice University, USA).

The Program Committee (PC) consisted of 38 experts and we received a total of 51 submissions. Each submission was reviewed by several reviewers, either PC members or external reviewers. Based on the reviews and the PC discussion phase, 24 full papers were selected for the conference program.

Our thanks go to the QEST community for making this an interesting and lively event; in particular, we acknowledge the hard work of the PC members and the

additional reviewers for sharing their valued expertise with the rest of the community. The collection and selection of papers was organized through the EasyChair Conference System. We are also indebted to Alfred Hofmann and Anna Kramer for their help in the preparation of this LNCS volume, and we thank Springer for kindly sponsoring the prize for the best paper award.

Also, thanks to the local organization team, especially Lijun Zhang, for his dedication and excellent work. Finally, we would like to thank Jane Hillston, chair of the QEST Steering Committee, for her guidance throughout the past year, as well as the members of the QEST Steering Committee.

We hope that you find the conference proceedings rewarding and will consider submitting papers to QEST 2019.

July 2018

Annabelle McIver  
Andras Horvath

# Organization

## General Chair

Huimin Lin Chinese Academy of Sciences, China

## Program Co-chairs

Andras Horvath University of Turin, Italy  
Annabelle McIver Macquarie University, Australia

## Steering Committee

Alessandro Abate University of Oxford, UK  
Luca Bortolussi University of Trieste, Italy  
Javier Campos Universidad de Zaragoza, Spain  
Pedro D'Argenio Universidad Nacional de Cordoba, Argentina  
Josee Desharnais Université Laval, Canada  
Jane Hillston University of Edinburgh, UK  
Andras Horvath University of Turin, Italy  
Joost-Pieter Katoen RWTH Aachen University, Germany  
Andrea Marin University of Venice, Italy  
Gethin Norman University of Glasgow, UK  
Enrico Vicario University of Florence, Italy  
Katinka Wolter FU Berlin, Germany

## Program Committee

Alessandro Abate University of Oxford, UK  
Erika Abraham RWTH Aachen University, Germany  
Gul Agha University of Illinois, USA  
Nail Akar Bilkent University, Turkey  
Mario S. Alvim Federal University of Minas Gerais, Brazil  
Varsha Apte IIT Bombay, India  
Ezio Bartocci TU Wien, Austria  
Sergiy Bogomolov IST Austria, Austria  
Tomas Brazdil Masaryk University, Czech Republic  
Giuliano Casale Imperial College London, UK  
Aleksandar Chakarov Phase Change Software LLC, USA  
Yuxin Deng East China Normal University, China  
Yuan Feng University of Technology Sydney, Australia  
Andras Horvath University of Turin, Italy  
Kausthub Joshi AT&T Labs, USA

Benjamin Kaminski	RWTH Aachen University, Germany
William Knottenbelt	Imperial College London, UK
Jan Kretinsky	TU Munich, Germany
Fumio Machida	NEC Japan, Japan
Daniele Manini	University of Turin, Italy
Andrea Marin	University of Venice, Italy
Annabelle McIver	Macquarie University, Australia
Gethin Norman	University of Glasgow, UK
Pavithra Prabhakar	Kansas State University, USA
Tahiry Rabehaja	Macquarie University, Australia
Guido Sanguinetti	University of Edinburgh, UK
Marielle Stoelinga	University of Twente, The Netherlands
Miklos Telek	Technical University of Budapest, Hungary
P. S. Thiagarajan	Harvard University, USA
Benny Van Houdt	University of Antwerp, Belgium
Enrico Vicario	University of Florence, Italy
Carey Williamson	University of Calgary, Canada
Huaming Wu	Tianjin University, China
Jianwen Xiang	Wuhan University of Technology, China
Naijun Zhan	Chinese Academy of Science, China
Lijun Zhang	Chinese Academy of Science, China

# Contents

On the Additive Capacity Problem for Quantitative Information Flow . . . . .	1
<i>Konstantinos Chatzikokolakis</i>	
HyperPCTL: A Temporal Logic for Probabilistic Hyperproperties . . . . .	20
<i>Erika Ábrahám and Borzoo Bonakdarpour</i>	
How Fast Is MQTT? Statistical Model Checking and Testing of IoT Protocols . . . . .	36
<i>Bernhard K. Aichernig and Richard Schumi</i>	
Parameter-Independent Strategies for pMDPs via POMDPs . . . . .	53
<i>Sebastian Arming, Ezio Bartocci, Krishnendu Chatterjee, Joost-Pieter Katoen, and Ana Sokolova</i>	
On the Verification of Weighted Kripke Structures Under Uncertainty . . . . .	71
<i>Giovanni Bacci, Mikkel Hansen, and Kim Guldstrand Larsen</i>	
Hospital Inventory Management Through Markov Decision Processes @runtime . . . . .	87
<i>Marco Biagi, Laura Carnevali, Francesco Santoni, and Enrico Vicario</i>	
Guaranteed Error Bounds on Approximate Model Abstractions Through Reachability Analysis . . . . .	104
<i>Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin</i>	
Verifying Probabilistic Timed Automata Against Omega-Regular Dense-Time Properties . . . . .	122
<i>Hongfei Fu, Yi Li, and Jianlin Li</i>	
Incremental Verification of Parametric and Reconfigurable Markov Chains . . . . .	140
<i>Paul Gainer, Ernst Moritz Hahn, and Sven Schewe</i>	
Lumping the Approximate Master Equation for Multistate Processes on Complex Networks . . . . .	157
<i>Gerrit Großmann, Charalampos Kyriakopoulos, Luca Bortolussi, and Verena Wolf</i>	
Analytical Solution for Long Battery Lifetime Prediction in Nonadaptive Systems . . . . .	173
<i>Dmitry Ivanov, Kim G. Larsen, Sibylle Schupp, and Jiří Srba</i>	

Action and State Based Computation Tree Measurement Language and Algorithms . . . . .	190
<i>Yaping Jing and Andrew S. Miner</i>	
Model Checking for Safe Navigation Among Humans . . . . .	207
<i>Sebastian Junges, Nils Jansen, Joost-Pieter Katoen, Ufuk Topcu, Ruohan Zhang, and Mary Hayhoe</i>	
Automated Verification of Concurrent Stochastic Games . . . . .	223
<i>Marta Kwiatkowska, Gethin Norman, David Parker, and Gabriel Santos</i>	
Bounded Verification of Reachability of Probabilistic Hybrid Systems . . . . .	240
<i>Ratan Lal and Pavithra Prabhakar</i>	
Control and Optimization of the SRPT Service Policy by Frequency Scaling . . . . .	257
<i>Andrea Marin, Isi Mitrani, Maryam Elahi, and Carey Williamson</i>	
Biased Processor Sharing in Fork-Join Queues . . . . .	273
<i>Andrea Marin, Sabina Rossi, and Matteo Sottana</i>	
Probabilistic Model Checking for Continuous-Time Markov Chains via Sequential Bayesian Inference . . . . .	289
<i>Dimitrios Miliotis, Guido Sanguinetti, and David Schnoerr</i>	
LIFT: Learning Fault Trees from Observational Data . . . . .	306
<i>Meike Nauta, Doina Bucur, and Mariëlle Stoelinga</i>	
A Robust Genetic Algorithm for Learning Temporal Specifications from Data . . . . .	323
<i>Laura Nenzi, Simone Silveti, Ezio Bartocci, and Luca Bortolussi</i>	
A Hemimetric Extension of Simulation for Semi-Markov Decision Processes . . . . .	339
<i>Mathias Ruggaard Pedersen, Giorgio Bacci, Kim Guldstrand Larsen, and Radu Mardare</i>	
Policy Synthesis for Collective Dynamics . . . . .	356
<i>Paul Piho and Jane Hillston</i>	
Modeling Humans: A General Agent Model for the Evaluation of Security . . . . .	373
<i>Michael Rausch, Ahmed Fawaz, Ken Keefe, and William H. Sanders</i>	

Approximate Time Bounded Reachability for CTMCs and CTMDPs:  
 A Lyapunov Approach . . . . . 389  
*Mahmoud Salamati, Sadegh Soudjani, and Rupak Majumdar*


On Saturation Effects in Coupled Speed Scaling . . . . . 407  
*Maryam Elahi and Carey Williamson*

**Author Index** . . . . . 423





# On the Additive Capacity Problem for Quantitative Information Flow

Konstantinos Chatzikokolakis<sup>(✉)</sup> 

CNRS, Paris, France  
kostas@chatzi.org

**Abstract.** Preventing information leakage is a fundamental goal in achieving confidentiality. In many practical scenarios, however, eliminating such leaks is impossible. It becomes then desirable to *quantify* the severity of such leaks and establish bounds on the threat they impose. Aiming at developing measures that are *robust* wrt a variety of operational conditions, a theory of channel *capacity* for the *g*-leakage model was developed in [1], providing solutions for several scenarios in both the multiplicative and the additive setting.

This paper continues this line of work by providing substantial improvements over the results of [1] for *additive* leakage. The main idea of employing the Kantorovich distance remains, but it is now applied to *quasimetrics*, and in particular the novel “*convex-separation*” quasimetric. The benefits are threefold: first, it allows to maximize leakage over a larger class of gain functions, most notably including the one of Shannon. Second, a solution is obtained to the problem of maximizing leakage over both priors and gain functions, left open in [1]. Third, it allows to establish an additive variant of the “Miracle” theorem from [3].

**Keywords:** Quantitative information flow · Capacity  
Kantorovich distance

## 1 Introduction

Preventing sensitive information from being leaked is a fundamental goal of computer security. There are many situations, however, in which completely eliminating such leaks is impossible for a variety of reasons. Sometimes the leak is *intentional*: we *want* to extract knowledge from a statistical database; sometimes it is due to *side channels* that are hard or impossible to fully control; sometimes the leak is in exchange to a *service*, as in the case of Location-Based Services; sometimes it is in exchange for *efficiency*: i.e. using a weaker but more efficient anonymous communication system.

In these cases, it becomes crucial to *quantify* such leaks, measure how important the threat they pose is and decide whether they can be tolerated or not. This problem is studied in the area of *quantitative information flow*, in which much progress has been done in recent years, both from a foundational viewpoint [3, 6, 11, 12, 17, 21, 22, 24], but also in the development of counter-measures

and verification techniques [4, 5, 7, 8, 10, 16, 19, 23, 25, 27], and the analysis of real systems [14, 15, 18, 20].

*Robustness* is a fundamental theme in this area; we aim at developing measures and bounds that are robust wrt a variety of adversaries and operational scenarios. In the context of the successful  $g$ -leakage model, the operational scenario is captured by a gain function  $g$ , and the adversary’s knowledge by a prior  $\pi$ . Developing the theme of robustness in this model, [1] studied the theory of *channel capacity*, that is the problem of maximizing leakage over  $\pi$  for a fixed  $g$ , maximizing over  $g$  for a fixed  $\pi$ , or maximizing over both  $\pi$  and  $g$ . Comparing the system’s prior and posterior vulnerability can be done either *multiplicatively* or *additively*, leading to a total of six capacity scenarios.

In this paper we make substantial progress in two of the scenarios for additive leakage, namely in maximizing over  $g$  alone, or over both  $\pi, g$ . When maximizing over  $g$ , we quickly realize that if we allow vulnerability to take values in the whole  $\mathbb{R}_{\geq 0}$ , we can always *scale* it up, leading to unbounded capacity. In practice, however, it is common to measure vulnerability within a predefined range; for instance, vulnerabilities capturing the probability of some unfortunate event (e.g. Bayes vulnerability) take values in  $[0, 1]$ , while vulnerabilities measuring bits of information (e.g. Shannon vulnerability) take values in  $[0, \log_2 |\mathcal{X}|]$ . It is thus natural to restrict to a *class*  $\mathcal{G}$  of gain functions, in which the range of vulnerabilities is limited. In [1], this is achieved by the class  $\mathbb{G}^1\mathcal{X}$  of *1-spanning* gain functions, in which the gain of different secrets varies by at most 1.

Although  $\mathbb{G}^1\mathcal{X}$  provides a solution for capacity, this choice is not completely satisfactory from the point of view of robustness, since it excludes important vulnerability functions. Most notably, *Shannon vulnerability* (the complement of entropy) is not  $k$ -spanning for *any*  $k$ , hence the capacity bound for  $\mathbb{G}^1\mathcal{X}$  does not apply, and indeed the leakage in this case (known as *mutual information*) does *exceed* the bound. In this paper we take a more permissive approach, by imposing the 1-spanning condition not on  $g$  itself, but on the corresponding vulnerability function  $V_g$ , leading to the class  $\mathbb{G}^\dagger\mathcal{X}$ . Since *any* vulnerability is  $k$ -spanning for some  $k$ , this class does not a priori exclude any type of adversary, it only restricts the range of values.

Solving the capacity problem for  $\mathbb{G}^\dagger\mathcal{X}$  is however not straightforward. It turns out that the core technique from [1], namely the use of the *Kantorovich distance* on the *hyper-distribution* produced by the channel, can still be applied. However, substantial modifications are needed, involving the use of *quasimetrics*, and in particular the novel “*convex-separation*” quasimetric, replacing the total variation used in [1]. These improvements not only lead to a solution to the problem of maximizing leakage over  $g : \mathbb{G}^\dagger\mathcal{X}$ , but also lead to a solution for the third scenario of maximizing over both  $\pi, g$ , as well as to a variant of the “Miracle” theorem for the additive setting.

In detail, the paper makes the following contributions to the study of  $g$ -capacity:

- We present a general technique for computing additive capacity wrt a class of gain functions  $\mathcal{G}$ , using the Kantorovich distance over a properly constructed quasimetric.
- This technique is instantiated for  $\mathbb{G}^\downarrow \mathcal{X}$  using the total variation metric, recovering the results of [1] in a more structured way.
- The same technique is then instantiated for the larger class  $\mathbb{G}^\uparrow \mathcal{X}$ , using the novel “convex-separation” quasimetric for which an efficient solution is provided.
- The results for  $\mathbb{G}^\uparrow \mathcal{X}$  also provide an immediate solution to the scenario of maximizing over both  $\pi, g$ , which was left completely open in [1].
- Finally, the results for  $\mathbb{G}^\uparrow \mathcal{X}$  lead to an “Additive Miracle” theorem, similar in nature to the “Miracle” theorem of [3] for the multiplicative case.

## 2 Preliminaries

**Channels and Their Effect on the Adversary’s Knowledge.** A *channel*  $C$  is a simple probabilistic model describing the behavior of a system that takes input values from a finite set  $\mathcal{X}$  (the secrets) and produces outputs from a finite set  $\mathcal{Y}$  (the observations). Formally, it is a *stochastic*  $|\mathcal{X}| \times |\mathcal{Y}|$  *matrix*, meaning that elements are non-negative and rows sum to 1.  $C_{x,y}$  can be thought of as the conditional probability of producing  $y$  when the input is  $x$ .

We denote by  $\mathbb{D}\mathcal{A}$  the set of all discrete distributions on  $\mathcal{A}$ , and by  $[a]:\mathbb{D}\mathcal{A}$  the *point* distribution, assigning probability 1 to  $a: \mathcal{A}$ . Given  $C$  and a distribution  $\pi: \mathbb{D}\mathcal{X}$ , called the *prior*, we can create a *joint* distribution  $J: \mathbb{D}(\mathcal{X} \times \mathcal{Y})$  as  $J_{x,y} = \pi_x C_{x,y}$ . When  $J$  is understood, it is often written in the usual notation  $p(x, y)$ , in which case the conditional probabilities  $p(y|x) = p(x,y)/p(x)$  coincide with  $C_{x,y}$  (when  $p(x)$  is non-zero) and the  $x$ -marginals  $p(x) = \sum_y p(x, y)$  coincide with  $\pi_x$ .

The prior  $\pi$  can be thought of as the *initial knowledge* that the adversary has about the secret. When secrets are passwords, for instance, she might know that some are more likely to be chosen than others. Always assuming that  $C$  is known to the adversary, each output  $y$  provides evidence that allows her to update her knowledge, creating a *posterior* distribution  $\delta^y$ , defined as  $\delta_x^y = p(x,y)/p(y)$ . This, of course, can be done for each output; every  $y: \mathcal{Y}$  potentially provides information to the adversary leading to an updated probabilistic knowledge  $\delta^y$ . But not all outputs have the same status; each happens with a different marginal probability  $p(y) = \sum_x p(x, y)$ , denoted by  $a_y$ .

Hence, the *effect of a channel*  $C$  to the adversary’s prior knowledge  $\pi$ , is to produce a set of posteriors  $\delta^y$ , each with probability  $a_y$ . It is conceptually useful to view this outcome as a single *distribution on distributions*, called a *hyper*-distribution or just hyper. Such a hyper has type  $\mathbb{D}^2 \mathcal{X}$  and is denoted by  $[\pi \triangleright C] = \sum_y a_y [\delta^y]$ .<sup>1</sup> The  $a_y$ -component of  $[\pi \triangleright C]$  is called the *outer* distribution, expressing the probability of obtaining the posteriors  $\delta^y$ , called the *inner* distributions.

<sup>1</sup> The notation is due to the fact that distributions can be convexly combined ( $\mathbb{D}\mathcal{A}$  is a vector space).  $\sum_y a_y [\delta^y]$  is exactly the hyper assigning probability  $a_y$  to each  $\delta^y$ .

$$\begin{array}{c|cccc} \pi & & y_1 & y_2 & y_3 & y_4 \\ \hline 1/3 & x_1 & 1 & 0 & 0 & 0 \\ 1/3 & x_2 & 0 & 1/2 & 1/4 & 1/4 \\ 1/3 & x_3 & 1/2 & 1/3 & 1/6 & 0 \end{array} \longrightarrow \begin{array}{c|cccc} J & & y_1 & y_2 & y_3 & y_4 \\ \hline x_1 & 1/3 & 0 & 0 & 0 & 0 \\ x_2 & 0 & 1/6 & 1/12 & 1/12 & 0 \\ x_3 & 1/6 & 1/9 & 1/18 & 0 & 0 \end{array} \longrightarrow \begin{array}{c|cccc} [\pi \triangleright C] & & 1/2 & 5/12 & 1/12 \\ \hline x_1 & 2/3 & 0 & 0 & 0 \\ x_2 & 0 & 3/5 & 1 & 0 \\ x_3 & 1/3 & 2/5 & 0 & 0 \end{array}$$

**Fig. 1.** A prior  $\pi$  (type  $\mathbb{D}\mathcal{X}$ ), a channel  $C$  (rows have type  $\mathbb{D}\mathcal{Y}$ ), a joint  $J$  (type  $\mathbb{D}(\mathcal{X} \times \mathcal{Y})$ ), and a hyper  $[\pi \triangleright C]$  (type  $\mathbb{D}^2\mathcal{X}$ , each column has type  $\mathbb{D}\mathcal{X}$ ).

An example of all constructions is given in Fig. 1. From a channel  $C$  and the uniform prior  $\pi$  we can construct the joint  $J$  by multiplying (element-wise) each column of  $C$  by  $\pi$ .  $J$  is then a single distribution assigning probabilities to each pair  $(x, y)$ . To construct the hyper  $[\pi \triangleright C]$ , we normalize (i.e. divide by  $p(y)$ ) each column  $J_{-,y}$ , forming the posterior  $\delta^y$ . The marginals  $p(y)$  become the outer probabilities  $a_y$ , labeling the columns of  $[\pi \triangleright C]$ . Finally, note that  $[\pi \triangleright C]$  no longer records the original label  $y$  of each column. As a consequence, the columns  $y_2$  and  $y_3$ , both producing the *same posterior*  $\delta^{y_2} = \delta^{y_3} = (0, 3/5, 2/5)$ , are *merged* in  $[\pi \triangleright C]$ , which assigns to that posterior the combined probability  $p(y_2) + p(y_3) = 5/12$ . This phenomenon happens automatically by the construction of the hyper.

**Vulnerability and Leakage.** A fundamental notion in measuring the information leakage of a system is that of a *vulnerability function*  $V : \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ . The goal of  $V(\pi)$  is to measure how vulnerable a system is when the adversary has knowledge  $\pi$  about the secret. To create a suitable vulnerability function we need to consider the operational scenario at hand: we first determine what the adversary is trying to achieve, then take  $V(\pi)$  as a measure of how successful the adversary is in that goal. Clearly, no single function can capture all operational scenarios; as a consequence a variety of vulnerability functions has been proposed in the literature, each having a different operational interpretation.

For instance, *Bayes-vulnerability*  $V_B(\pi) := \max_x \pi_x$  measures the probability of success of an adversary who tries to guess the complete secret in one try; *Shannon-vulnerability*  $V_H(\pi) := \log_2 |\mathcal{X}| + \sum_x \pi_x \log_2 \pi_x$  (the complement of entropy) measures the expected number of Boolean questions needed to reveal the secret; and *Guessing-vulnerability*  $V_G(\pi) = |\mathcal{X}|^{1/2} - \sum_i i\pi_i$  (where the  $i$ -indexing of  $\mathcal{X}$  is in non-decreasing probability order) measures the expected numbers of tries to guess the secret correctly.

To study vulnerability in a unifying way, the  $g$ -leakage framework was introduced in [3], in which the operating scenario is parametrized by a (possibly infinite) set of *actions*  $\mathcal{W}$ , and a *gain function*  $g : \mathcal{W} \times \mathcal{X} \rightarrow \mathbb{R}$ . Intuitively,  $\mathcal{W}$  consists of actions that the adversary can perform to *exploit* his knowledge about the system. Then,  $g(w, x)$  models the adversary's reward when he performs the action  $w$  and the actual secret is  $x$ . In such an operational scenario, it is natural to define  $g$ -*vulnerability*  $V_g$  as the expected gain of a rational adversary who chooses the best available action:

$$V_g(\pi) := \sup_w \sum_x \pi_x g(w, x) .$$

The  $g$ -leakage framework is quite expressive, allowing to obtain a variety of vulnerability functions as special cases for suitable choices of  $g$ . For instance, by picking  $\mathcal{W} = \mathcal{X}$  and the *identity* gain function given by  $g_{\text{id}}(w, x) = 1$  iff  $w = x$  and 0 otherwise, we get  $V_{g_{\text{id}}} = V_B$ . Similarly, we can construct gain functions expressing Shannon (for which an infinite  $\mathcal{W}$  is needed) and Guessing vulnerabilities, as well as a variety of other operational scenarios. In fact, it can be shown that any continuous and convex vulnerability  $V : \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$  can be written as  $V_g$  for some  $g$  [2].

For expressiveness, it is crucial to allow  $g$  to potentially take negative values, and  $\mathcal{W}$  to be infinite. However, it is desirable that  $V_g$  *itself* be non-negative and finite-valued, since it is meant to express vulnerability. As a consequence we always restrict to the class of  $\mathbb{G}\mathcal{X}$  of gain functions, defined as those such that  $V_g : \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}_{\geq 0}$ . As already discussed in the introduction, it is often desirable to further restrict to subsets of  $\mathbb{G}\mathcal{X}$ .

Having established a way to measure vulnerability in the prior case, we move on to measuring how vulnerable our system is after observing the output of a channel  $C$ . Viewing the outcome of  $C$  on  $\pi$  as the hyper  $[\pi \triangleright C]$ , there is a natural answer: we can measure the vulnerability of each posterior (inner) distribution of  $[\pi \triangleright C]$ , then average by the outer probabilities, leading to the following definition of *posterior  $g$ -vulnerability*:

$$V_g[\pi \triangleright C] := \sum_y a_y V_g(\delta^y) \quad \text{where} \quad [\pi \triangleright C] = \sum_y a_y [\delta^y]$$

Finally, information *leakage* is measured by comparing the vulnerability in the prior and posterior case. Depending on how we compare the two vulnerabilities, this leads to the *additive* or *multiplicative* leakage, defined as:

$$\mathcal{L}_g^+(\pi, C) := V_g[\pi \triangleright C] - V_g(\pi), \quad \mathcal{L}_g^\times(\pi, C) := \frac{V_g[\pi \triangleright C]}{V_g(\pi)}.$$

**Additive  $g$ -Capacities.** A fundamental theme when measuring information leakage is *robustness*; we need bounds that are robust wrt a variety of different adversaries and operational scenarios. Following this theme, since the  $g$ -leakage of a channel  $C$  depends on both the prior  $\pi$  and the gain function  $g$ , it is natural to ask what is the *maximum leakage* of  $C$ , over a *class* of gain functions  $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$  and a class of priors  $\mathcal{D} \subseteq \mathbb{D}\mathcal{X}$ . This maximum leakage is known as the *capacity* of  $C$ .

**Definition 1.** *The additive  $(\mathcal{G}, \mathcal{D})$ -capacity of  $C$ , for  $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$ ,  $\mathcal{D} \subseteq \mathbb{D}\mathcal{X}$ , is*

$$\mathcal{ML}_{\mathcal{G}}^+(\mathcal{D}, C) := \sup_{g: \mathcal{G}, \pi: \mathcal{D}} \mathcal{L}_g^+(\pi, C).$$

For brevity, when maximizing *only* over  $\pi$  for a *fixed*  $g$ , we write  $\mathcal{ML}_g^+(\mathcal{D}, C)$  instead of  $\mathcal{ML}_{\{g\}}^+(\mathcal{D}, C)$ ; similarly, when  $\pi$  is fixed we write  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$ ; for specific classes, say  $\mathcal{G} = \mathbb{G}\mathcal{X}$ ,  $\mathcal{D} = \mathbb{D}\mathcal{X}$ , we write  $\mathcal{ML}_{\mathbb{G}}^+(\mathbb{D}, C)$  instead of  $\mathcal{ML}_{\mathbb{G}\mathcal{X}}^+(\mathbb{D}\mathcal{X}, C)$ . We can maximize over  $\pi$ , or  $g$ , or both, getting three scenarios

for additive capacity. The multiplicative capacity  $\mathcal{ML}_{\mathcal{G}}^{\times}(\mathcal{D}, C)$ , defined similarly, is outside the scope of this paper; the corresponding three scenarios are studied in [1].

For the first scenario,  $g$  is fixed and we maximize over the whole  $\mathbb{D}\mathcal{X}$ . For some gain functions an efficient solution exists; for instance, for  $g_H$  giving Shannon vulnerability,  $\mathcal{ML}_{g_H}^+(\mathbb{D}, C)$  is the Shannon capacity (maximum transmission rate)<sup>2</sup> which can be computed using the well-known Blahut-Arimoto algorithm [13]. However, for  $g_{\text{id}}$  (giving Bayes vulnerability), bounding  $\mathcal{ML}_{g_{\text{id}}}^+(\mathbb{D}, C)$  is known to be NP-complete [1], which of course leaves no hope for a general solution.

The second scenario (fixed  $\pi$ , maximize over  $g$ ) is the main focus of this paper and is studied in detail in Sect. 3. Our solution turns out to also provide an answer for the third scenario (maximize over both  $\pi, g$ ), discussed in Sect. 3.5.

### 3 Computing Additive Capacities

In this section we study the problem of computing the additive  $(\mathcal{G}, \pi)$ -capacity. We quickly realize, however, that the unrestricted maximization over the whole  $\mathbb{G}\mathcal{X}$  yields unbounded leakage. The problem is the *unbounded range* of  $V_g$ , and can be illustrated by “scaling”  $g$ . Define the scaling of  $g$  by  $k > 0$  as  $g_{\times k}(w, x) = k \cdot g(w, x)$ . It is easy to show [1] that this operation gives leakage  $\mathcal{L}_{g_{\times k}}^+(\pi, C) = k \cdot \mathcal{L}_g^+(\pi, C)$ , and since  $k$  can be arbitrary we get that  $\mathcal{ML}_{\mathbb{G}}^+(\pi, C) = +\infty$ .<sup>3</sup>

There are important classes of gain functions, however, which effectively *limit* the range of  $V_g$ , causing the additive leakage to remain bounded. Even when  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$  is finite, computing it efficiently is non-trivial. A solution can be obtained by exploiting the fact that  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$  is connected to the well-known *Kantorovich distance* between  $[\pi]$  and  $[\pi \triangleright C]$ .

This section proceeds as follows. In Sect. 3.1 we recall the Kantorovich distance and then use it in Sect. 3.2 to obtain a generic technique for computing  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$ , in time linear on the size of  $C$ , using properties of  $\mathcal{G}$ . We apply these bounds to obtain efficient solutions for the class  $\mathbb{G}^1\mathcal{X}$  of 1-spanning gain functions in Sect. 3.3, as well as the class  $\mathbb{G}^\dagger\mathcal{X}$  of gain functions giving 1-spanning vulnerability in Sect. 3.4. Finally, Sect. 3.5 discusses the scenario of maximizing over both  $\pi$  and  $g$ .

#### 3.1 The Kantorovich and Wasserstein Distances

We begin by recalling the Kantorovich distance between probability distributions. Given  $\alpha: \mathbb{D}\mathcal{A}$  and random variable  $F: \mathcal{A} \rightarrow \mathbb{R}$  we write  $\mathcal{E}_\alpha F$  for the expected value of  $F$  over  $\alpha$ , or  $\mathcal{E}_{x \sim \alpha} F(x)$  to make precise the variable we are averaging over. Observe that for a point distribution centered at  $a \in \mathcal{A}$  we have  $\mathcal{E}_{[a]} F = F(a)$ .

<sup>2</sup> Which is why we generally refer to the maximization of leakage as “capacity”.

<sup>3</sup> The same phenomenon happens for *multiplicative* leakage, this time demonstrated by shifting. To keep  $\mathcal{ML}_{\mathcal{G}}^{\times}(\pi, C)$  bounded we can restrict to the class  $\mathbb{G}^+\mathcal{X}$  of *non-negative* gain functions.

A function  $d : \mathcal{A}^2 \rightarrow \mathbb{R}$  is called a *quasimetric* iff it satisfies the triangle inequality and  $d(a, a') = 0 \wedge d(a', a) = 0$  iff  $a = a'$ . If  $d$  is also symmetric it is called a *metric*. The set of all quasimetrics on  $\mathcal{A}$  is denoted by  $\mathbb{M}\mathcal{A}$ . Although less frequently used than metrics, quasimetrics will play an important role in computing additive capacity in Sect. 3.4.

A natural quasimetric on  $\mathbb{R}$  is given by

$$d_{\mathbb{R}}^{\leq}(x, y) := \max\{y - x, 0\} .$$

Intuitively,  $d_{\mathbb{R}}^{\leq}(x, y)$  measures “how much smaller” than  $y$  is  $x$ ; 0 means that  $x$  is no smaller than  $y$ . This quasimetric can be extended to  $x, y \in \mathbb{R}^n$  as  $d_{\mathbb{R}^n}^{\leq}(x, y) = \sum_i d_{\mathbb{R}}^{\leq}(x_i, y_i)$ , giving an “asymmetric Manhattan” distance.

A function  $F : \mathcal{A} \rightarrow \mathbb{R}$  is called *d, d<sub>T</sub>-Lipschitz* iff

$$d_T(F(a), F(a')) \leq d(a, a') \quad \text{for all } a, a' \in \mathcal{A} . \quad (1)$$

The set of all such functions (also called *contractions*) is denoted by  $\mathbb{C}^{d, d_T}\mathcal{A}$ . For the source metric, a scaled distance  $k \cdot d$  (for some  $k \geq 0$ ) is often used. For the target metric  $d_T$  the Euclidean distance  $d_{\mathbb{R}}$  is commonly employed (in which case we might simply write *d-Lipschitz* for brevity). In this section, however, we consider functions that are *d, d<sub>ℝ</sub>^≤-Lipschitz*, which holds iff

$$F(a') - F(a) \leq d(a, a') \quad \text{for all } a, a' \in \mathcal{A} . \quad (2)$$

Note that the max from the definition of  $d_{\mathbb{R}}^{\leq}$  is not needed, since  $d(a, a')$  is non-negative.

The Kantorovich construction allows us to *lift* a metric  $d$  on  $\mathcal{A}$  to a metric on probability distributions over  $\mathcal{A}$ . The standard construction is done by maximizing  $|\mathcal{E}_{\alpha}F - \mathcal{E}_{\alpha'}F'|$  over all functions that are *d, d<sub>ℝ</sub>-Lipschitz*. Note that the Euclidean distance is implicitly used twice in this construction: first, in the Lipschitz condition and second, for comparing  $\mathcal{E}_{\alpha}F$  and  $\mathcal{E}_{\alpha'}F'$ . We can, however, define variants of the Kantorovich by using any other distance on  $\mathbb{R}$ . Our purpose is to work with quasimetrics, hence we employ  $d_{\mathbb{R}}^{\leq}$ , leading to the following definition.

**Definition 2.** *The Kantorovich quasimetric is the mapping  $\mathbb{K}^{\leq} : \mathbb{M}\mathcal{A} \rightarrow \text{MID}\mathcal{A}$ :*

$$\mathbb{K}^{\leq}(d)(\alpha, \alpha') := \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}}\mathcal{A}} d_{\mathbb{R}}^{\leq}(\mathcal{E}_{\alpha}F, \mathcal{E}_{\alpha'}F) = \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}}\mathcal{A}} \mathcal{E}_{\alpha'}F - \mathcal{E}_{\alpha}F .$$

Note that max was again dropped from  $d_{\mathbb{R}}^{\leq}$ , since the sup is anyway non-negative ( $\mathcal{E}_{\alpha'}F - \mathcal{E}_{\alpha}F = 0$  for  $F$  constant).

An important property of the Kantorovich distance is that it has a dual formulation as the Wasserstein (or “earth-moving”) metric, for which efficient algorithms exist. Earth moving measures measuring the cost of transforming one distribution into another, using the underlying distance  $d$  as the cost function. Given two distributions  $\alpha, \alpha' : \mathbb{D}\mathcal{A}$  (the “source” and “target”), an earth-moving *strategy* is a joint distribution  $S \in \mathbb{D}\mathcal{A}^2$  whose two marginals are  $\alpha$  and  $\alpha'$ .

We write  $\mathcal{S}_{\alpha, \alpha'}$  for the set of such strategies. The Wasserstein distance is then defined as the minimum transportation cost; similarly to Kantorovich, it provides a lifting from  $\mathbb{M}\mathcal{A}$  to  $\mathbb{MID}\mathcal{A}$ .

**Definition 3.** *The Wasserstein distance is the mapping  $\mathbb{W} : \mathbb{M}\mathcal{A} \rightarrow \mathbb{MID}\mathcal{A}$  given by:*

$$\mathbb{W}(d)(\alpha, \alpha') := \inf_{S: \mathcal{S}_{\alpha, \alpha'}} \mathcal{E}_S d .$$

The well-known Kantorovich-Rubinstein theorem [26] states that, if  $(d, \mathcal{A})$  is a *separable* metric space then  $\mathbb{K}(d) = \mathbb{W}(d)$ . For our purposes, we will use this result in the restricted case where one of the two distributions is a *point* distribution  $[a]$ . This restriction is useful for two reasons: first, it allows us to give a simplified proof, adjusted to our  $\mathbb{K}^<$  variant, drop the assumption of separability and allow  $d$  to be a quasimetric. Second, we show that  $\mathbb{W}(d)([a], \alpha)$  can be easily obtained as the expected (wrt  $\alpha$ ) distance between  $a$  and the elements in the support of  $\alpha$ .

We first fix some notation: given  $d \in \mathbb{M}\mathcal{A}$  and  $a \in \mathcal{A}$ , we denote by  $d_a : \mathcal{A} \rightarrow \mathbb{R}$  the function “currying”  $a$ , defined by  $d_a(x) = d(a, x)$ . Note that  $d_a$  is  $d, d_{\mathbb{R}}^<$ -Lipschitz since  $d_a(y) - d_a(x) = d(a, y) - d(a, x) \leq d(x, y)$  follows from the triangle inequality. We are now ready to state the result relating the two distances.

**Theorem 1.** *Let  $d \in \mathbb{M}\mathcal{A}$  be any quasimetric. For all  $[a], \alpha \in \mathbb{D}\mathcal{A}$  it holds that*

$$\mathbb{K}^<(d)([a], \alpha) = \mathbb{W}(d)([a], \alpha) = \mathcal{E}_\alpha d_a .$$

*Proof.* We start with the Wasserstein distance. The crucial observation is that for point  $[a]$ , there is informally a single source “pile of earth”: all the probability has to come from  $a$ . As a consequence,  $\mathcal{S}_{[a], \alpha}$  contains a unique strategy  $S_{x, y} = [a]_x \cdot \alpha_y$  with independent marginals  $[a]$  and  $\alpha$ . We can then calculate

$$\begin{aligned} & \mathbb{W}(d)([a], \alpha) \\ = & \inf_{S: \mathcal{S}_{[a], \alpha}} \mathcal{E}_S d && \text{“definition of } \mathbb{W} \text{”} \\ = & \mathcal{E}_{(x, y) \sim S} d(x, y) && \text{“take unique } S \text{ with independent marginals } [a], \alpha \text{”} \\ = & \mathcal{E}_{y \sim \alpha} \mathcal{E}_{x \in [a]} d(x, y) && \text{“independence of marginals”} \\ = & \mathcal{E}_{y \sim \alpha} d(a, y) && \text{“expectation over point distribution”} \\ = & \mathcal{E}_\alpha d_a \end{aligned}$$

For the Kantorovich distance, we bound it from above by  $\mathcal{E}_\alpha d_a$ , as follows:

$$\begin{aligned} & \mathbb{K}^<(d)([a], \alpha) \\ = & \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^<} \mathcal{A}} \mathcal{E}_\alpha F - \mathcal{E}_{[a]} F && \text{“definition of } \mathbb{K}^< \text{”} \\ = & \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^<} \mathcal{A}} \mathcal{E}_\alpha F - F(a) && \text{“expectation over point distribution”} \\ = & \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^<} \mathcal{A}} \mathcal{E}_{x \sim \alpha} (F(x) - F(a)) \\ \leq & \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^<} \mathcal{A}} \mathcal{E}_{x \sim \alpha} d(a, x) && \text{“(2), } F \text{ is } d, d_{\mathbb{R}}^<\text{-Lipschitz”} \\ = & \mathcal{E}_\alpha d_a \end{aligned}$$



Finally we bound  $\mathbb{K}^<(d)([a], \alpha)$  from below by  $\mathcal{E}_\alpha \bar{d}_a$ , as follows:

$$\begin{aligned}
& \mathbb{K}^<(d)([a], \alpha) \\
= & \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}} \mathcal{A}} \mathcal{E}_\alpha F - \mathcal{E}_{[a]} F && \text{“definition of } \mathbb{K}^< \text{”} \\
\geq & \mathcal{E}_\alpha \bar{d}_a - \mathcal{E}_{[a]} \bar{d}_a && \text{“} \bar{d}_a \in \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}} \mathcal{A} \text{”} \\
= & \mathcal{E}_\alpha \bar{d}_a && \text{“} \mathcal{E}_{[a]} \bar{d}_a = \bar{d}_a(a) = 0 \text{”} \quad \square
\end{aligned}$$

### 3.2 Computing Additive $(\mathcal{G}, \pi)$ -capacity

We now discuss a generic technique for computing the additive  $(\mathcal{G}, \pi)$ -capacity, for a given family of gain functions  $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$ , using the Kantorovich distance. Recall that  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$  (Definition 1) is defined as the maximum difference between the posterior and prior vulnerabilities  $V_g[\pi \triangleright C]$ ,  $V_g[\pi]$ . The latter are simply the expected value of  $V_g$  over the distributions  $[\pi]$ ,  $[\pi \triangleright C]$ , which are *hyper* distributions, having sample space  $\mathbb{D}\mathcal{X}$ .

We start by taking  $\mathcal{A} = \mathbb{D}\mathcal{X}$  as the underlying space of the Kantorovich construction. A quasimetric  $d \in \mathbb{M}\mathbb{D}\mathcal{X}$  measures the distance between two distributions on secrets. The key for bounding  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$  from above is to find such a quasimetric  $d$  wrt which  $V_g$  is Lipschitz for all  $g \in \mathcal{G}$ . Since the Kantorovich distance maximizes  $\mathcal{E}_\alpha F - \mathcal{E}_\alpha F$  over all Lipschitz functions  $F$ , it will provide an upper bound to the additive capacity.

Bounding the capacity from below is also possible if there exists some  $g \in \mathcal{G}$  such that  $d_\pi = V_g$ . This is due to the fact that the  $g$ -leakage for this  $g$  is exactly  $\mathcal{E}_{[\pi \triangleright C]} d_\pi$ .

In the following, given a class of gain function  $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$ , we denote by  $V_{\mathcal{G}} = \{V_g \mid g \in \mathcal{G}\}$  the set of  $g$ -vulnerabilities induced by  $\mathcal{G}$ . The bounding technique is formalized in the following result.

**Theorem 2.** *Let  $d \in \mathbb{M}\mathbb{D}\mathcal{X}$ , let  $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$  and fix a channel  $C$  and prior  $\pi$ . Then*

$$d_\pi \in V_{\mathcal{G}} \quad \text{implies} \quad \mathcal{ML}_{\mathcal{G}}^+(\pi, C) \geq k, \quad \text{and} \quad (3)$$

$$\mathbb{C}^{d, d_{\mathbb{R}}^{\leq}} \mathbb{D}\mathcal{X} \supseteq V_{\mathcal{G}} \quad \text{implies} \quad \mathcal{ML}_{\mathcal{G}}^+(\pi, C) \leq k, \quad (4)$$

where  $k = \mathbb{K}^<(d)([\pi], [\pi \triangleright C]) = \mathbb{W}(d)([\pi], [\pi \triangleright C]) = \mathcal{E}_{[\pi \triangleright C]} d_\pi$ .

*Proof.* The fact that  $\mathbb{K}^<(d)([\pi], [\pi \triangleright C]) = \mathbb{W}(d)([\pi], [\pi \triangleright C]) = \mathcal{E}_{[\pi \triangleright C]} d_\pi$  comes from Theorem 1, for  $\mathcal{A} = \mathbb{D}\mathcal{X}$ ,  $a = \pi$ ,  $\alpha = [\pi \triangleright C]$ . We start with (3): for  $V_g = d_\pi$  we have that  $V_g(\pi) = 0$  and  $V_g[\pi \triangleright C] = \mathcal{E}_{[\pi \triangleright C]} d_\pi$  hence  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C) \geq \mathcal{L}_g^+(\pi, C) = \mathcal{E}_{[\pi \triangleright C]} d_\pi$ . For (4) we have that

$$\begin{aligned}
& \mathcal{ML}_{\mathcal{G}}^+(\pi, C) \\
= & \sup_{V_g: V_{\mathcal{G}}} \mathcal{E}_{[\pi \triangleright C]} V_g - \mathcal{E}_{[\pi]} V_g && \text{“definition”} \\
\leq & \sup_{F: \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}} \mathbb{D}\mathcal{X}} \mathcal{E}_{[\pi \triangleright C]} F - \mathcal{E}_{[\pi]} F && \text{“sup over larger class”} \\
= & \mathbb{K}^<(d)([\pi], [\pi \triangleright C]) && \text{“definition”} \quad \square
\end{aligned}$$

So far we have considered an unknown quasimetric  $d$  on probability distributions, and identified in Theorem 2 two properties of  $d$  that provide bounds for additive leakage. It is not clear, however, whether such a quasimetric exists and what is its relationship with the class  $\mathcal{G}$ . We now show that the choice of  $d$  is in fact canonical for each class. More precisely, for any  $\mathcal{G}$  we can construct a quasimetric  $d_{\mathcal{G}}^{\leq}$  satisfying the second condition of Theorem 2. Furthermore, if a quasimetric  $d$  satisfying *both* conditions (for any  $\pi$ ) does exist, then it is *unique* and equal to  $d_{\mathcal{G}}^{\leq}$ .

**Theorem 3.** *Let  $\mathcal{G} \subseteq \mathbb{G}\mathcal{X}$  and define a quasimetric  $d_{\mathcal{G}}^{\leq}: \mathbb{M}\mathbb{D}\mathcal{X}$  as*

$$d_{\mathcal{G}}^{\leq}(\pi, \sigma) := \sup_{g \in \mathcal{G}} d_{\mathbb{R}}^{\leq}(V_g(\pi), V_g(\sigma)) .$$

*Then  $V_{\mathcal{G}} \subseteq \mathbb{C}^{d_{\mathcal{G}}^{\leq}, d_{\mathbb{R}}^{\leq}}\mathbb{D}\mathcal{X}$ . Moreover, if  $\{d_{\pi} \mid \pi: \mathbb{D}\mathcal{X}\} \subseteq V_{\mathcal{G}} \subseteq \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}}\mathbb{D}\mathcal{X}$  holds for some quasimetric  $d$ , then  $d = d_{\mathcal{G}}^{\leq}$ .*

*Proof.* Let  $g \in \mathcal{G}$ . We trivially have that

$$d_{\mathbb{R}}^{\leq}(V_g(\pi), V_g(\sigma)) \leq \sup_{g \in \mathcal{G}} d_{\mathbb{R}}^{\leq}(V_g(\pi), V_g(\sigma)) = d_{\mathcal{G}}^{\leq}(\pi, \sigma) ,$$

hence  $V_g$  is  $d_{\mathcal{G}}^{\leq}, d_{\mathbb{R}}^{\leq}$ -Lipschitz. Now let  $d: \mathbb{M}\mathbb{D}\mathcal{X}$  such that  $\{d_{\pi} \mid \pi: \mathbb{D}\mathcal{X}\} \subseteq V_{\mathcal{G}} \subseteq \mathbb{C}^{d, d_{\mathbb{R}}^{\leq}}\mathbb{D}\mathcal{X}$  and let  $\pi, \sigma: \mathbb{D}\mathcal{X}$ . From  $d_{\pi} \in V_{\mathcal{G}}$ , we get that

$$d(\pi, \sigma) = d_{\mathbb{R}}^{\leq}(d_{\pi}(\pi), d_{\pi}(\sigma)) \leq \sup_{g \in \mathcal{G}} d_{\mathbb{R}}^{\leq}(V_g(\pi), V_g(\sigma)) = d_{\mathcal{G}}^{\leq}(\pi, \sigma) .$$

Then, since  $V_g$  is  $d, d_{\mathbb{R}}^{\leq}$ -Lipschitz for all  $g \in \mathcal{G}$ , we get that

$$d_{\mathcal{G}}^{\leq}(\pi, \sigma) = \sup_{g \in \mathcal{G}} d_{\mathbb{R}}^{\leq}(V_g(\pi), V_g(\sigma)) \leq \sup_{g \in \mathcal{G}} d(\pi, \sigma) = d(\pi, \sigma) ,$$

hence  $d$  and  $d_{\mathcal{G}}^{\leq}$  coincide.  $\square$

Finally, an important corollary of this technique is that, assuming that  $d$  can be computed in time  $O(|\mathcal{X}|)$ ,  $\mathcal{ML}_{\mathcal{G}}^+(\pi, C)$  can be computed in time  $O(|\mathcal{X}||\mathcal{Y}|)$ . Indeed, calculating  $\mathcal{E}_{[\pi \triangleright C]} d_{\pi}$  involves computing the output and posterior distributions of  $[\pi \triangleright C]$ . The former can be computed in  $O(|\mathcal{X}||\mathcal{Y}|)$  time via the joint matrix  $J$ ; then for each posterior  $\delta^y$ , we need to construct  $\delta^y$  ( $O(|\mathcal{X}|)$ ) and compute  $d(\pi, \delta^y)$  ( $O(|\mathcal{X}|)$ ).

### 3.3 Additive Capacity for 1-spanning Gain Functions

We are now ready to provide a complete method for computing additive capacity for an important family of gain functions. The *span* of a function  $f: \mathcal{A} \rightarrow \mathbb{R}$  is defined as

$$\|f\| := \sup_{a, a' \in \mathcal{A}} |f(a) - f(a')| ,$$

while for gain functions (having two arguments) we define  $\|g\| := \sup_{w \in \mathcal{W}} \|g(w, \cdot)\|$ . Since scaling  $g$  causes unbounded leakage, a natural solution is to

limit the range of  $g$ . This can be done in an elegant way, without completely fixing  $g$ 's range, by requiring that  $\|g\| \leq 1$ , which brings us to the class  $\mathbb{G}^1\mathcal{X}$  of 1-spanning gain functions, the topic of study in this section. In the following we see that this restriction in fact limits the steepness of  $V_g$ .

Note that any result for  $\mathbb{G}^1\mathcal{X}$  can be straightforwardly extended to  $k$ -spanning gain functions, since  $\mathcal{L}_{g \times k}^+(\pi, C) = k \cdot \mathcal{L}_g^+(\pi, C)$  implies that the  $k$ -spanning additive capacity is simply  $k \cdot \mathcal{ML}_{\mathbb{G}^1}^+(\pi, C)$ . Note also that this class is quite large: any  $g$  with a finite number of actions has finite span. For an infinite number of actions, however, it is possible that  $\|g\| = +\infty$ , i.e. that  $g$  is not  $k$ -spanning for any  $k$ ; important functions such as Shannon vulnerability fall in this category. In Sect. 3.4 we enlarge our class of gain functions to include such cases.

**Total Variation, Steepness and  $g$ 's span.** To apply the bounding technique of Sect. 3.2 to  $\mathbb{G}^1\mathcal{X}$  we need a (quasi)metric  $d \in \mathbb{M}\mathbb{D}\mathcal{X}$  with respect to which  $V_g$  is Lipschitz when  $g$  is 1-spanning. Conveniently, this is the case of the well-known total variation distance:

$$\text{tv}(\pi, \pi') := \sup_{X \subseteq \mathcal{X}} |\pi(X) - \pi'(X)|.$$

For discrete distributions, expressed as vectors, the total variation is equal to  $d_{\mathbb{R}^n}^{\leq}(\pi, \pi')$ , which is in fact symmetric when restricted to probability distributions (because the elements sum up to 1), and equal to  $1/2$  the Manhattan distance  $\|\pi - \pi'\|_1$ . Total variation is a natural choice for  $\mathbb{D}\mathcal{X}$  when  $\mathcal{X}$  is an “unstructured” space with no underlying metric. Indeed, such spaces can be naturally equipped with the discrete metric  $\text{dm}: \mathbb{M}\mathcal{X}$ , defined as  $\text{dm}(x, x') = 0$  iff  $x = x'$  and 1 otherwise. It is well known that the Kantorovich lifting of this metric gives total variation, namely  $\text{tv} = \mathbb{K}(\text{dm})$ . Note, however, that the fact that  $\text{tv}$  is the result of Kantorovich is not important for our goals; our technique involves applying  $\mathbb{K}^{\leq}$  to  $\text{tv}$  itself, lifting it to hyper-distributions.

The Lipschitz property wrt  $\text{tv}$  and the standard Euclidean  $d_{\mathbb{R}}$  naturally expresses the steepness of  $V_g$ . If  $V_g$  is  $k \cdot \text{tv}$ -Lipschitz then the vulnerability can be modified by at most  $k \cdot \epsilon$  while changing the probability of any subset of secrets by  $\epsilon$ . The larger  $k$  is, the steeper  $V_g$  can be, i.e. the faster it is allowed to change. It turns out that this property is tightly connected to the span of  $g$ , as the following result from [1] states.

**Proposition 1.** *For all  $g: \mathbb{G}\mathcal{X}$  it holds that  $V_g$  is  $\|g\| \cdot \text{tv}$ -Lipschitz.*

As a consequence of the above result we get that  $\|V_g\| \leq \|g\|$ , since  $|V_g(\pi) - V_g(\pi')|$  can be no greater than  $\|g\| \cdot \text{tv}(\pi, \pi') \leq \|g\|$ .

**Putting the Pieces Together.** We can finally recover (in a more structured way) the result of [1] for computing the additive  $(\mathbb{G}^1, \pi)$ -capacity. Denote by  $\mathbf{1}_S(x)$  the indicator function, equal to 1 if  $x \in S$  and 0 otherwise.

**Theorem 4.** *Given a channel  $C$  and prior  $\pi$ , it holds that*

$$\mathcal{ML}_{\mathbb{G}^1}^+(\pi, C) = \mathcal{E}_{[\pi \ C]} \text{tv}_\pi .$$

*The capacity is realized by the gain function  $g: \mathbb{G}^1 \mathcal{X}$  defined by*

$$\mathcal{W} := 2^{\mathcal{X}}, \quad g(W, x) := \mathbf{1}_W(x) - \pi(W),$$

*for which it holds that  $V_g = \text{tv}_\pi$ .*<sup>4</sup>

*Proof.* The result comes from Theorem 2 for  $d = \text{tv}$  and  $\mathcal{G} = \mathbb{G}^1 \mathcal{X}$ ; we show here that the two conditions of this theorem hold.

For the upper bound we need to show that  $g: \mathbb{G}^1 \mathcal{X}$  implies that  $V_g$  is  $\text{tv}$ ,  $d_{\mathbb{R}}^{\leq}$ -Lipschitz. But from Proposition 1 we know that  $V_g$  is  $\text{tv}$ ,  $d_{\mathbb{R}}$ -Lipschitz, which is a *stronger* property since  $d_{\mathbb{R}}^{\leq}$  is no greater than  $d_{\mathbb{R}}$  for all reals.

For the lower bound, we need to show that the claimed gain function  $g$  is 1-spanning and  $V_g(\tau) = \text{tv}_\pi(\tau)$ . Note that  $g$  clearly depends on the fixed  $\pi$ . For the 1-spanning part we have that  $|g(W, x) - g(W, x')| = |\mathbf{1}_W(x) - \mathbf{1}_W(x')| \leq 1$ . Moreover, it holds that

$$\begin{aligned} & V_g(\tau) \\ = & \max_W \mathcal{E}_{x \sim \tau} g(w, x) && \text{“definition of } V_g \text{”} \\ = & \max_W (\mathcal{E}_{x \sim \tau} \mathbf{1}_W(x) - \pi(W)) && \text{“definition of } g \text{”} \\ = & \max_W \sum_{x \in W} (\tau_x - \pi_x) \\ = & \sum_x d_{\mathbb{R}}^{\leq}(\pi_x, \tau_x) && \text{“take } W = \{x \mid \tau_x \geq \pi_x\} \text{”} \\ = & \text{tv}(\pi, \tau). && \text{“tv}(\pi, \pi') = d_{\mathbb{R}^n}^{\leq}(\pi, \pi') \text{”} \quad \square \end{aligned}$$

In the above proof we showed that  $\text{tv}$  satisfies the two conditions of Theorem 2. From Theorem 3 we know that there is a unique quasimetric satisfying these conditions which can be constructed explicitly from the class  $\mathcal{G} = \mathbb{G}^1 \mathcal{X}$ , that is:  $\text{tv}(\pi, \pi') = d_{\mathbb{G}^1}^{\leq}(\pi, \sigma) = \sup_{g: \mathbb{G}^1 \mathcal{X}} V_g(\sigma) - V_g(\pi)$ . Note also that  $\text{tv}$  can be computed in  $|\mathcal{X}|$  time, hence, as discussed in Sect. 3.2,  $\mathcal{ML}_{\mathbb{G}^1}^+(\pi, C)$  can be computed in time  $O(|\mathcal{X}||\mathcal{Y}|)$ .

### 3.4 Additive Capacity for 1-spanning Vulnerability Functions

As discussed in the introduction it is often desirable to measure vulnerability within a predefined range, for instance  $[0, 1]$  or  $[0, \log_2 n]$  ( $n = |\mathcal{X}|$ ). A natural way to achieve this is to consider  $k$ -spanning gain functions, implicitly limiting the range of  $V_g$  to an interval of size at most  $k$ . This choice, however, excludes important vulnerabilities that cannot be expressed as  $V_g$  for any  $k$ -spanning  $g$ .

For instance, for the Shannon vulnerability function  $V_H$  (see Sect. 2) the additive capacity is equal to the well known Shannon mutual information. Although

<sup>4</sup> The choice  $\mathcal{W} = \mathcal{X} \rightarrow \{-1, 1\}$ ,  $g(w, x) = \frac{1}{2}(w(x) - \varepsilon_\pi w)$  is also capacity-realizing [1].

$V_H$  lies within  $[0, \log_2 n]$  and it can be expressed as  $V_{g_H}$  for a suitable  $g_H$ , this gain function is *not*  $\log_2 n$ -spanning, in fact  $\|g_H\| = +\infty$ . As a consequence, the additive capacity  $\mathcal{ML}_{\mathbb{G}^1}^+(\pi, C)$ , discussed in the previous section, does not provide a bound for  $g_H$ -leakage. Indeed, the mutual-information of the fully transparent identity channel  $C_{\text{id}}$  on a uniform prior is  $\mathcal{L}_{g_H}^+(\pi^u, C_{\text{id}}) = \log_2 n$ , which exceeds its additive capacity wrt  $\log_2 n$ -spanning gain functions, which is equal to  $\log_2 n \cdot \mathcal{ML}_{\mathbb{G}^1}^+(\pi^u, C_{\text{id}}) = \frac{n-1}{n} \log_2 n$ .

Aiming at robustness wrt a larger class of vulnerabilities, we can allow functions  $V_g$  that have a bounded range, even though  $g$  itself is unbounded. Similarly to  $\mathbb{G}^1 \mathcal{X}$ , we choose to *limit* the range of  $V_g$  without completely fixing it, by restricting to the class  $\mathbb{G}^\dagger \mathcal{X} = \{g: \mathbb{G} \mathcal{X} \mid \|V_g\| \leq 1\}$  of *1-spanning vulnerability functions*.

Since  $\|V_g\| \leq \|g\|$ , but not vice-versa (as  $V_H$  demonstrates), it holds that  $\mathbb{G}^\dagger \mathcal{X} \subset \mathbb{G}^1 \mathcal{X}$ . Since any convex (and continuous) function can be expressed as  $V_g$  for some properly constructed  $g$  [2],  $V_{\mathbb{G}^\dagger \mathcal{X}}$  is the class of all 1-spanning convex functions. Note also that, since any  $V_g, g: \mathbb{G} \mathcal{X}$  is bounded, it is  $k$ -spanning for some  $k$ .

To compute the additive capacity via the technique of Sect. 3.2, we need a quasimetric satisfying both conditions of Theorem 2. From Theorem 3 we know that such a quasimetric (if it exists) is unique and equal to the  $d_{\mathbb{G}}^{\zeta}$  construction. In the previous section, this turned out to be the well-known total variation distance. In this section, on the other hand, we start directly with  $d_{\mathbb{G}}^{\zeta}$  for our class  $\mathcal{G} = \mathbb{G}^\dagger \mathcal{X}$ . The resulting quasimetric  $d_{\mathbb{G}^\dagger}^{\zeta}$  is called the “convex-separation” quasimetric, and is given by

$$d_{\mathbb{G}^\dagger}^{\zeta}(\pi, \sigma) := \sup_{g \in \mathbb{G}^\dagger \mathcal{X}} d_{\mathbb{R}}^{\zeta}(V_g(\pi), V_g(\sigma)) = \sup_{g \in \mathbb{G}^\dagger \mathcal{X}} V_g(\sigma) - V_g(\pi).$$

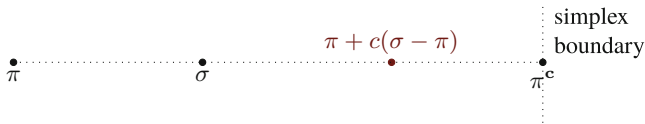
Note that, once again, we removed the max from the definition of  $d_{\mathbb{R}}^{\zeta}$  since the sup is anyway non-negative.

An important property of  $d_{\mathbb{G}^\dagger}^{\zeta}$  is that it admits a simple closed-form solution.

**Theorem 5.** *The convex-separation quasimetric  $d_{\mathbb{G}^\dagger}^{\zeta}$  is equal to*

$$d_{\mathbb{G}^\dagger}^{\zeta}(\pi, \sigma) = \max_{x: \lceil \pi \rceil} 1 - \frac{\sigma_x}{\pi_x}.$$

*Proof.* Let  $\pi, \sigma \in \mathbb{D} \mathcal{X}$ . Assume  $\pi \neq \sigma$  (the case  $\pi = \sigma$  is trivial) and consider the line in  $\mathbb{R}^n$  joining the two priors, as shown in Fig. 2. The points on that line,



**Fig. 2.** Line connecting  $\pi$  and  $\sigma$ , extended to the boundary of the simplex.

starting from  $\pi$  and moving towards  $\sigma$  can be written as  $\pi^c = \pi + c(\sigma - \pi)$  for  $c \geq 0$ . Continuing on that line, at some point we are going to hit the boundary of the probability simplex  $\mathbb{D}\mathcal{X}$ . Let  $\pi^c$  be the point on that boundary, i.e.

$$\mathbf{c} := \max\{c \mid \pi^c \in \mathbb{D}\mathcal{X}\} . \quad (5)$$

Note that  $\mathbf{c} \geq 1$  since  $\pi^1 = \sigma \in \mathbb{D}\mathcal{X}$ . Now let  $F: \mathbb{D}\mathcal{X} \rightarrow \mathbb{R}$  be convex and 1-spanning. Since  $\sigma$  lies in the line segment between  $\pi$  and  $\pi^c$ , we can write it as a convex combination

$$\sigma = \mathbf{c}^{-1}\pi^c + (1 - \mathbf{c}^{-1})\pi \quad (6)$$

with  $\mathbf{c}^{-1} \in (0, 1]$ . From convexity we get that

$$F(\sigma) \leq \mathbf{c}^{-1}F(\pi^c) + (1 - \mathbf{c}^{-1})F(\pi) ,$$

from which, together with  $F(\pi^c) - F(\pi) \leq 1$  ( $F$  is 1-spanning), we get

$$F(\sigma) - F(\pi) \leq \mathbf{c}^{-1}(F(\pi^c) - F(\pi)) \leq \mathbf{c}^{-1} . \quad (7)$$

We now compute  $\mathbf{c}$ , which is given by the maximization problem (5). The problem has a single variable  $c$  and the constraint  $\pi^c \in \mathbb{D}\mathcal{X}$  can be expressed by  $\sum_x \pi_x^c = 1$  and the inequalities  $\pi_x^c \geq 0$ . The first constraint  $\sum_x \pi_x^c = 1$  is always satisfied by construction of  $\pi^c$ . Hence we only need to ensure that  $\pi_x^c = \pi_x + c(\sigma_x - \pi_x) \geq 0$  for all  $x: \mathcal{X}$ . If  $\pi_x = \sigma_x$  this is always satisfied, and if  $\pi_x < \sigma_x$  then this imposes a *lower* bound on  $c$ . The only interesting case is when  $\pi_x > \sigma_x$  which gives us an upper bound  $c \leq \pi_x / (\pi_x - \sigma_x)$ . The max  $c$  satisfying all upper bounds is equal to the smallest of them:

$$\mathbf{c} = \min_{x: \pi_x > \sigma_x} \frac{\pi_x}{\pi_x - \sigma_x} .$$

Replacing  $\mathbf{c}$  in (7) we get

$$F(\sigma) - F(\pi) \leq \max_{x: \pi_x > \sigma_x} \frac{\pi_x - \sigma_x}{\pi_x} = \max_{x: \lceil \pi \rceil} 1 - \frac{\sigma_x}{\pi_x} .$$

We finally show that the above bound is attainable. Define

$$F_\pi(\tau) := \max_{x: \lceil \pi \rceil} 1 - \frac{\tau_x}{\pi_x} .$$

$F_\pi$  is convex as the max of convex (in fact linear) functions of  $\tau$ , so it can be expressed as  $V_g$  (see Theorem 6 for the exact  $g$ ). Moreover,  $F_\pi(\pi) = 0$ , hence  $F_\pi(\sigma) - F_\pi(\pi) = \max_{x: \lceil \pi \rceil} 1 - \frac{\sigma_x}{\pi_x}$ , which concludes the proof.  $\square$

We can now use  $d_{\mathbb{G}^\dagger}^<$  to compute the additive  $(\mathbb{G}^\dagger, \pi)$ -capacity.

**Theorem 6.** *Given a channel  $C$  and prior  $\pi$ , it holds that*

$$\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C) = \mathcal{E}_{[\pi \triangleright C]} d_{\mathbb{G}^\dagger}^< \pi = 1 - \sum_{y: \mathcal{Y}} \min_{x: [\pi]} C_{x,y}.$$

The capacity is realized by the complement of the “ $\pi$ -reciprocal” gain function

$$\mathcal{W} = [\pi], \quad g_{\pi^{-1}}^c = \begin{cases} 1 - \frac{1}{\pi_x}, & \text{if } w = x \\ 1, & \text{if } w \neq x \end{cases},$$

for which it holds that  $V_{g_{\pi^{-1}}^c} = d_{\mathbb{G}^\dagger}^< \pi$ , and as a consequence  $g_{\pi^{-1}}^c: \mathbb{G}^\dagger \mathcal{X}$ .

*Proof.* The result comes from Theorem 2 for  $d = d_{\mathbb{G}^\dagger}^<$  and  $\mathcal{G} = \mathbb{G}^\dagger \mathcal{X}$ ; we show here that its two conditions of the theorem hold. The second is satisfied automatically by the construction of  $d_{\mathbb{G}^\dagger}^<$  (Theorem 3). For the first condition, after simple calculations we find that the  $g_{\pi^{-1}}^c$ -vulnerability function is equal to  $V_{g_{\pi^{-1}}^c}(\sigma) = \max_{x: [\pi]} 1 - \frac{\sigma_x}{\pi_x}$ , hence from Theorem 5 we have that  $V_{g_{\pi^{-1}}^c} = d_{\mathbb{G}^\dagger}^< \pi$ . Finally, simple calculations show that  $V_{g_{\pi^{-1}}^c}[\pi \triangleright C] = 1 - \sum_y \min_{x \in [\pi]} C_{x,y}$ , which concludes the proof since  $V_{g_{\pi^{-1}}^c}(\pi) = d_{\mathbb{G}^\dagger}^< \pi(\pi) = 0$ .  $\square$

Note that the capacity-realizing gain function  $g_{\pi^{-1}}^c$  essentially “cancels out” the effect of the prior, making  $\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C)$  independent from  $\pi$ , and equal to 1 minus the sum of the column *minima* of  $C$  (ignoring the rows when  $\pi_x = 0$ ). Remarkably, the “ $\pi$ -reciprocal” gain function  $g_{\pi^{-1}}$  (the complement of  $g_{\pi^{-1}}^c$ ) produces the same “cancellation” effect for *multiplicative*  $(\mathbb{G}^+, \pi)$ -capacity, making it independent from  $\pi$ .

An observation that can be made from Theorem 6 is that the capacity realizing  $g$  is  $k$ -spanning for  $k = \max_{x: [\pi]} \frac{1}{\pi_x}$ . From this we can conclude that  $\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C) \leq k \cdot \mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C)$  for  $k = \max_{x: [\pi]} 1/\pi_x$ . In particular  $\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi^u, C) \leq |\mathcal{X}| \cdot \mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi^u, C)$  for the uniform  $\pi^u$ .

A final note about our use of quasimetrics. Although the total variation  $\text{tv}$ , used in Sect. 3.3 for  $\mathbb{G}^1 \mathcal{X}$ , is a proper metric,  $d_{\mathbb{G}^\dagger}^<$  used in this section for  $\mathbb{G}^\dagger \mathcal{X}$  is not, since it is not symmetric. This is why we had to work with quasimetrics; it is certainly possible to define a symmetric variant of  $d_{\mathbb{G}^\dagger}^<$  (eg. as  $\max\{d_{\mathbb{G}^\dagger}^<(\pi, \sigma), d_{\mathbb{G}^\dagger}^<(\sigma, \pi)\}$ ), however this metric would not satisfy both conditions of Theorem 2. Recall that for each class  $\mathcal{G}$  there can be at most one quasimetric satisfying both properties, and for  $\mathbb{G}^\dagger \mathcal{X}$  this is  $d_{\mathbb{G}^\dagger}^<$ .

### 3.5 Maximize over both $g$ and $\pi$

This scenario was left open in [1] (which uses the class  $\mathbb{G}^1 \mathcal{X}$ ), since  $\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C)$  depends on the prior, and maximizing it over  $\pi$  is challenging. Our results on the larger class  $\mathbb{G}^\dagger \mathcal{X}$ , however, lead to a complete solution since  $\mathcal{ML}_{\mathbb{G}^\dagger}^+(\pi, C)$  is independent from  $\pi$ . By Theorem 6, any full-support  $\pi$  with  $g_{\pi^{-1}}^c$  are capacity-realizing, giving

$$\mathcal{ML}_{\mathbb{G}^\dagger}^+(\mathbb{D}, C) = 1 - \sum_y \min_x C_{x,y}.$$

## 4 The Additive Miracle Theorem

The multiplicative Bayes-capacity  $\mathcal{ML}_{g_{\text{id}}}^{\times}(\mathbb{D}, C)$  is well known to be realized on a uniform prior, and is equal to the sum of the column maxima of  $C$  [9, 24]. A result from [3], which was surprising enough to be named “Miracle”, states that  $\mathcal{ML}_{g_{\text{id}}}^{\times}(\mathbb{D}, C)$  is in fact a *universal upper bound* for multiplicative leakage (wrt non-negative  $g$ ’s).

**Theorem 7 (“Miracle”, [3]).** *For any  $C$ ,  $\pi: \mathbb{D}\mathcal{X}$ , and non-negative  $g: \mathbb{G}^+\mathcal{X}$ , we have*

$$\mathcal{L}_g^{\times}(\pi, C) \leq \sum_y \max_x C_{x,y} = \mathcal{ML}_{g_{\text{id}}}^{\times}(\mathbb{D}, C).$$

In [1], this theorem was used to easily conclude that the capacity  $\mathcal{ML}_{\mathbb{G}^+}^{\times}(\pi, C)$  is equal to  $\mathcal{ML}_{g_{\text{id}}}^{\times}(\mathbb{D}, C)$  for any full-support  $\pi$ . In the additive case, having already a solution for  $\mathcal{ML}_{\mathbb{G}^{\downarrow}}^+(\pi, C)$ , we can go in the opposite direction and obtain an additive variant of the miracle theorem. Denote by  $g_{\text{id}}^c = 1 - g_{\text{id}}$  the complement of  $g_{\text{id}}$ .

**Theorem 8 (“Additive Miracle”).** *For any  $C$ ,  $\pi: \mathbb{D}\mathcal{X}$ , and  $g: \mathbb{G}^{\downarrow}\mathcal{X}$ , we have*

$$\mathcal{L}_g^+(\pi, C) \leq 1 - \sum_y \min_x C_{x,y} = |\mathcal{X}| \cdot \mathcal{ML}_{g_{\text{id}}^c}^+(\mathbb{D}, C).$$

*Proof.* The inequality is a direct consequence of Theorem 6; note that it holds for any prior since  $1 - \sum_y \min_x C_{x,y} \geq 1 - \sum_y \min_{x: \lceil \pi \rceil} C_{x,y}$ . Now let  $g^* = g_{\text{id}}^c \times |\mathcal{X}|$ , for which it holds that  $\mathcal{ML}_{g^*}^+(\mathbb{D}, C) = |\mathcal{X}| \cdot \mathcal{ML}_{g_{\text{id}}^c}^+(\mathbb{D}, C)$ . We have that  $V_{g^*} = |\mathcal{X}|(1 - \min_x \pi_x)$ . For uniform  $\pi^u$  we compute  $\mathcal{L}_{g^*}^+(\pi^u, C) = 1 - \sum_y \min_x C_{x,y}$ , and since  $g^*: \mathbb{G}^{\downarrow}\mathcal{X}$ , this is an upper bound for all  $\pi: \mathbb{D}\mathcal{X}$ , and hence equal to  $\mathcal{ML}_{g^*}^+(\mathbb{D}, C)$ .  $\square$

The multiplicative and additive miracle theorems are similar in nature, although they do have several differences. They both provide a universal bound for leakage, which holds for all priors and all gain functions within a certain class. In the multiplicative case, this is the class  $\mathbb{G}^+\mathcal{X}$  of non-negative gain functions, while in the additive case, the class  $\mathbb{G}^{\downarrow}\mathcal{X}$  of gain functions producing a 1-spanning  $V_g$ . In the multiplicative case the bound is given by the sum of column maxima of  $C$ , while in the additive case by 1 minus the sum of column minima. In the multiplicative case the bound coincides with the  $(g_{\text{id}}, \mathbb{D})$ -capacity for the identity gain function (i.e. the Bayes-capacity), which is realized on a uniform prior. In the additive case the bound is ( $|\mathcal{X}|$  times) the  $(g_{\text{id}}^c, \mathbb{D})$ -capacity for the “complement of identity” gain function, also realized on a uniform prior.

*Example.* Consider the case  $\mathcal{X} = \{x_1, x_2\}$  with a gain function penalizing wrong guesses, defined as  $\mathcal{W} = \mathcal{X}$ , and  $g(w, x) = 1$  iff  $w = x$  and  $-1$  otherwise. Note that  $V_g$  is always non-negative since the probability of a correct guess is at least 0.5 (for  $|\mathcal{X}| =$

2). For a uniform prior  $\pi^u$ , both guesses are equivalent, giving expected gain  $0.5 \cdot 1 + 0.5 \cdot -1 = 0$ , so  $V_g(\pi^u) = 0$ .

$C$	$y_1$	$y_2$
$x_1$	0.8	0.2
$x_2$	0.2	0.8



Now consider the illustrated channel  $C$  which gives rather good information about the secret. Computing the two posteriors we get  $\delta^{y_1} = (0.8, 0.2)$  and  $\delta^{y_2} = (0.2, 0.8)$ , which both give  $g$ -vulnerability  $V_g(\delta^{y_1}) = V_g(\delta^{y_2}) = 0.8 - 0.2 = 0.6$ . Hence  $V_g[\pi^u \triangleright C] = 0.6$  and as a consequence  $\mathcal{L}_g^\times(\pi^u, C) = +\infty$ , clearly larger than the multiplicative Bayes capacity  $\mathcal{ML}_{g_{\text{id}}}^\times(\mathbb{D}, C) = 0.8 + 0.8 = 1.6$ . The miracle theorem does not apply here since  $g$  takes negative values.

On the other hand,  $V_g$  is 1-spanning (although  $g$  itself is 2-spanning), since its value is at least 0 (for a uniform prior) and at most 1 (for a point prior). As a consequence the additive miracle theorem applies, guaranteeing that  $\mathcal{L}_g^+(\pi^u, C) \leq 1 - \sum_y \min_x C_{x,y} = 1 - 0.2 - 0.2 = 0.6$ . Indeed  $\mathcal{L}_g^+(\pi^u, C) = 0.6 - 0$ , exactly matching the bound.  $\square$

## 5 Conclusion and Future Work

We studied the problem of computing additive  $g$ -capacities. Extending the Kantorovich technique of [1] with quasimetrics, we provided a solution for the class  $\mathbb{G}^\uparrow \mathcal{X}$  of 1-spanning vulnerabilities, which, in contrast to  $\mathbb{G}^1 \mathcal{X}$ , can include any vulnerability function (by scaling). The results also provided a solution to the problem of maximizing leakage over both  $\pi$  and  $g$ , and lead to an additive variant of the miracle theorem of [3].

In future work we plan to study approximation algorithms for all scenarios, especially  $\mathcal{ML}_g^+(\mathbb{D}, C)$  which is NP-complete in general. Moreover, we aim at developing a theory that unifies the two main approaches to robustness, namely *capacity* and *refinement*.

**Acknowledgements.** All results were obtained in the process of preparing a manuscript on Quantitative Information Flow with my long-term collaborators M. Alvim, C. Morgan, A. McIver, C. Palamidessi and G. Smith, and were heavily influenced by their feedback.

## References

1. Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: Additive and multiplicative notions of leakage, and their capacities. In: Proceedings of CSF, pp. 308–322. IEEE (2014)
2. Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: Axioms for information leakage. In: Proceedings of CSF, pp. 77–92 (2016)
3. Alvim, M.S., Chatzikokolakis, K., Palamidessi, C., Smith, G.: Measuring information leakage using generalized gain functions. In: Proceedings of CSF, pp. 265–279 (2012)
4. Antonopoulos, T., Gazzillo, P., Hicks, M., Koskinen, E., Terauchi, T., Wei, S.: Decomposition instead of self-composition for proving the absence of timing channels. In: PLDI, pp. 362–375. ACM (2017)
5. Backes, M., Köpf, B., Rybalchenko, A.: Automatic discovery and quantification of information leaks. In: Proceedings of S&P, pp. 141–153 (2009)

6. Barthe, G., Köpf, B.: Information-theoretic bounds for differentially private mechanisms. In: Proceedings of CSF, pp. 191–204 (2011)
7. Biondi, F., Kawamoto, Y., Legay, A., Traonouez, L.-M.: HyLeak: hybrid analysis tool for information leakage. In: D’Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 156–163. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-68167-2\\_11](https://doi.org/10.1007/978-3-319-68167-2_11)
8. Biondi, F., Legay, A., Traonouez, L.-M., Wasowski, A.: QUAIL: a quantitative security analyzer for imperative code. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 702–707. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39799-8\\_49](https://doi.org/10.1007/978-3-642-39799-8_49)
9. Braun, C., Chatzikokolakis, K., Palamidessi, C.: Quantitative notions of leakage for one-try attacks. In: Proceedings of MFPS, ENTCS, vol. 249, pp. 75–91. Elsevier (2009)
10. Chatzikokolakis, K., Chothia, T., Guha, A.: Statistical measurement of information leakage. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 390–404. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-12002-2\\_33](https://doi.org/10.1007/978-3-642-12002-2_33)
11. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: On the Bayes risk in information-hiding protocols. *J. Comp. Secur.* **16**(5), 531–571 (2008)
12. Clarkson, M.R., Schneider, F.B.: Quantification of integrity. In: Proceedings of CSF, pp. 28–43 (2010)
13. Cover, T.M., Thomas, J.A.: *Elements of Information Theory*, 2nd edn. Wiley, Hoboken (2006)
14. Doychev, G., Köpf, B.: Rigorous analysis of software countermeasures against cache attacks. In: PLDI, pp. 406–421. ACM (2017)
15. Heusser, J., Malacaria, P.: Quantifying information leaks in software. In: Proceedings ACSAC 2010, pp. 261–269 (2010)
16. Khouzani, M.H.R., Malacaria, P.: Relative perfect secrecy: universally optimal strategies and channel design. In: Proceedings of CSF, pp. 61–76 (2016)
17. Köpf, B., Basin, D.: An information-theoretic model for adaptive side-channel attacks. In: Proceedings of CCS, pp. 286–296 (2007)
18. Köpf, B., Mauborgne, L., Ochoa, M.: Automatic quantification of cache side-channels. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 564–580. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31424-7\\_40](https://doi.org/10.1007/978-3-642-31424-7_40)
19. Köpf, B., Rybalchenko, A.: Approximation and randomization for quantitative information-flow analysis. In: Proceedings of CSF, pp. 3–14 (2010)
20. Köpf, B., Smith, G.: Vulnerability bounds and leakage resilience of blinded cryptography under timing attacks. In: Proceedings of CSF, pp. 44–56 (2010)
21. Malacaria, P.: Assessing security threats of looping constructs. In: Proceedings of POPL, pp. 225–235 (2007)
22. McIver, A., Meinicke, L., Morgan, C.: Compositional closure for bayes risk in probabilistic noninterference. In: Abramsky, S., Gavioille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010. LNCS, vol. 6199, pp. 223–235. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14162-1\\_19](https://doi.org/10.1007/978-3-642-14162-1_19)
23. Meng, Z., Smith, G.: Calculating bounds on information leakage using two-bit patterns. In: Proceedings of PLAS, pp. 1:1–1:12 (2011)
24. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FoSSaCS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-00596-1\\_21](https://doi.org/10.1007/978-3-642-00596-1_21)

25. Sweet, I., Trilla, J.M.C., Scherrer, C., Hicks, M., Magill, S.: What's the over/under? Probabilistic bounds on information leakage. In: Bauer, L., Küsters, R. (eds.) POST 2018. LNCS, vol. 10804, pp. 3–27. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-89722-6\\_1](https://doi.org/10.1007/978-3-319-89722-6_1)
26. Villani, C.: Topics in Optimal Transportation. No. 58, American Mathematical Society (2003)
27. Yasuoka, H., Terauchi, T.: Quantitative information flow – verification hardness and possibilities. In: Proceedings of CSF, pp. 15–27 (2010)



# HyperPCTL: A Temporal Logic for Probabilistic Hyperproperties

Erika Ábrahám<sup>1</sup> and Borzoo Bonakdarpour<sup>2</sup>(✉)

<sup>1</sup> RWTH Aachen University, Aachen, Germany

<sup>2</sup> Iowa State University, Ames, USA  
borzoo@iastate.edu

**Abstract.** In this paper, we propose a new temporal logic for expressing and reasoning about probabilistic hyperproperties. *Hyperproperties* characterize the relation between different independent executions of a system. *Probabilistic* hyperproperties express quantitative dependencies between such executions. The standard temporal logics for probabilistic systems, i.e., PCTL and PCTL\* can refer only to a single path at a time and, hence, cannot express many probabilistic hyperproperties of interest. The logic proposed in this paper, HyperPCTL, adds explicit and simultaneous quantification over multiple traces to PCTL. Such quantification allows expressing probabilistic hyperproperties. A model checking algorithm for the proposed logic is also introduced for discrete-time Markov chains.

## 1 Introduction

Four decades ago, Lamport [16] used the notion of *trace properties* as a means to specify the correctness of individual executions of concurrent programs. This notion was later formalized and classified by Alpern and Schneider [1] to *safety* and *liveness* properties. Temporal logics (e.g., LTL [17] and CTL [3]) were built based on these efforts to give formal syntax and semantics to requirements of trace properties. Subsequently, verification algorithms were developed to reason about individual traces of a system.

It turns out that many interesting requirements are not trace properties. For example, important information-flow security policies such as *noninterference*<sup>1</sup> [10] and *observational determinism*<sup>2</sup> [22] cannot be expressed as properties of individual execution traces of a system. Also, service level agreement requirements (e.g., mean response time and percentage uptime) that use statistics of a system across all executions of a system are not trace properties. Rather, they are properties of sets of execution traces, also known as *hyperproperties* [5]. Temporal logics HyperLTL and HyperCTL\* [4] have been proposed to provide a unifying

<sup>1</sup> Noninterference stipulates that input commands from high-privileged users have no effect on the system's behavior observed by low-privileged observers.

<sup>2</sup> Observational determinism requires that two executions that start at two low initial states appear deterministic to a low user.

framework to express and reason about hyperproperties. They allow explicit and simultaneous quantification over multiple paths to LTL and to CTL\*.

Hyperproperties can also be probabilistic. Such *probabilistic hyperproperties* generally express probabilistic relations between independent executions of a system. For example, in information-flow security, adding probabilities is motivated by establishing a connection between information theory and information flow across multiple traces. It is also motivated by using probabilistic schedulers, which opens up an opportunity for the attacker to set up a probabilistic *covert channel*, whereby information is obtained by statistical inferences drawn from the relative frequency of outcomes of a repeated computation. Policies that defend against such an attempt, known as *probabilistic noninterference*, stipulate that the probability of every low-observable trace be the same for every low-equivalent initial state. Such policies quantify on different execution traces and the probability of reaching certain states in the independent and simultaneous executions.

Consider the following classic example [21] comprising of two threads  $th$  and  $th'$ :

$$th : \text{ while } h > 0 \text{ do } \{h \leftarrow h - 1\}; l \leftarrow 2 \quad || \quad th' : l \leftarrow 1$$

where  $h$  is an input by a high-privileged user and  $l$  is an output observable by low-privileged users. Probabilistic noninterference would require that  $l$  obtains values of 1 and 2 with equal probabilities, regardless of the initial value of  $h$ . However, assuming that the scheduler chooses to execute atomic statements of the threads  $th$  and  $th'$  iteratively with uniform probability distribution, the likely outcome of the race between the two assignments  $l \leftarrow 1$  and  $l \leftarrow 2$  depends on the initial value of  $h$ : the larger the initial value of  $h$ , the greater the probability that the final value of  $l$  is 2. For example, if the initial value of  $h$  is 0 in one execution, then the final value of  $l$  is 1 with probability 1/4 and 2 with probability 3/4, but for the initial value  $h = 5$  in another independent execution we can observe the final value  $l = 1$  with probability 1/4096 and  $l = 2$  with probability 4095/4096. Thus, it holds that for two independent executions with initial  $h$  values 0 resp. 5 the larger  $h$  value leads to a lower probability for  $l = 1$  upon termination. I.e., this program does not satisfy probabilistic noninterference.

It is straightforward to observe that requirements such as probabilistic noninterference cannot be expressed in existing probabilistic temporal logics such as PCTL [12] and PCTL\*, as they cannot draw connection between the probability of reaching certain states in independent executions. Also, introducing probability operators to HyperLTL is not quite natural, as the semantics of HyperLTL is trace-based and probabilistic logics are branching-time in nature. Moreover, introducing probability operators to HyperCTL\* cannot be done trivially. With this motivation, in this paper, we propose the temporal logic HyperPCTL that lifts PCTL by allowing explicit quantification over initial states and, hence, multiple computation trees simultaneously, as well as probability of occurring propositions that stipulate relationships among those traces. For the above example, the following HyperPCTL formula expresses probabilistic noninterference, which obviously does not hold:

$$\forall \sigma. \forall \sigma'. \left( \text{init}_\sigma \wedge \text{init}_{\sigma'} \wedge h_\sigma \neq h_{\sigma'} \right) \Rightarrow \left( \left( \mathbb{P} \diamond (fn_\sigma \wedge (l=1)_\sigma) = \mathbb{P} \diamond (fn_{\sigma'} \wedge (l=1)_{\sigma'}) \right) \wedge \left( \mathbb{P} \diamond (fn_\sigma \wedge (l=2)_\sigma) = \mathbb{P} \diamond (fn_{\sigma'} \wedge (l=2)_{\sigma'}) \right) \right)$$

That is, for any two executions from initial states  $\sigma$  and  $\sigma'$  (i.e., initial values of  $h$ ), the probability distribution of terminating with value  $l = 1$  (or  $l = 2$ ) is uniform.

In addition to probabilistic noninterference, we show that **HyperPCTL** can express other important requirements and policies, some not related to information-flow security. First, we show that **HyperPCTL** subsumes probabilistic bisimulation. We also show that **HyperPCTL** can express requirements such as differential privacy, quantitative information flow, and probabilistic causation (a.k.a. causality). We also present a **HyperPCTL** model checking algorithm for discrete-time Markov chains (DTMCs). The complexity of the algorithm is polynomial-time in the size of the input DTMC and is **PSPACE-hard** in the size of the input **HyperPCTL** formula. We also discuss a wide range of open problem to be tackled by future research. We believe that this paper opens a new area in rigorous analysis of probabilistic systems.

*Organization.* The rest of the paper is organized as follows. Section 2 defines the syntax and semantics of **HyperPCTL**. Section 3 provides a diverse set of example requirements that **HyperPCTL** can express. We present our model checking algorithm in Sect. 4. Related work is discussed in Sect. 5. Finally, we make concluding remarks and discuss future work in Sect. 6.

## 2 HyperPCTL

We assume the systems to be described by **HyperPCTL** formulas to be modeled as discrete-time Markov chains.

**Definition 1.** A (discrete-time) Markov chain (DTMC)  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  is a tuple with the following components:

- $S$  is a finite nonempty set of states,
- $\mathbf{P} : S \times S \rightarrow [0, 1]$  is a transition probability function with  $\sum_{s' \in S} \mathbf{P}(s, s') = 1$  for all states  $s \in S$ ,
- $\text{AP}$  is a set of atomic propositions, and
- $L : S \rightarrow 2^{\text{AP}}$  is a labeling function. ■

A *path* of a Markov chain  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  is defined as an infinite sequence  $\pi = s_0 s_1 s_2 \dots \in S^\omega$  of states with  $\mathbf{P}(s_i, s_{i+1}) > 0$ , for all  $i \geq 0$ ; we write  $\pi[i]$  for  $s_i$ . Let  $\text{Paths}^s(\mathcal{M})$  denote the set of all (infinite) paths starting in  $s$  in  $\mathcal{M}$ , and  $\text{Paths}_{\text{fin}}^s(\mathcal{M})$  denote the set of all finite prefixes of paths from  $\text{Paths}^s(\mathcal{M})$ , which we sometimes call *finite paths*.

## 2.1 HyperPCTL Syntax

To be able to express probabilistic hyperproperties of DTMCs, the syntax of HyperPCTL differs from computation tree logic (CTL) in two different aspects. Firstly, CTL quantification over paths starting in a given state is replaced by a probability operator expressing the probability that a certain property holds on the paths starting in a given state; this extension is similar to probabilistic computation tree logic (PCTL), but whereas PCTL allows only the comparison of these probabilities to constant thresholds, we allow the arbitrary usage of such probabilities in arithmetic constraints. Secondly, we add quantification over states to express hyperproperties; note that whereas  $\text{HyperCTL}^*$  extends  $\text{CTL}^*$  by path quantification, in the probabilistic setting the argumentation moves from paths to the probabilities of paths, which are determined in the context of states (where the paths start).

HyperPCTL *state formulas* are inductively defined by the following grammar:

$$\begin{aligned} \psi ::= & \forall\sigma.\psi \mid \exists\sigma.\psi \mid \mathbf{true} \mid a_\sigma \mid \psi \wedge \psi \mid \neg\psi \mid p \sim p \\ p ::= & \mathbb{P}(\varphi) \mid c \mid p + p \mid p - p \mid p \cdot p \end{aligned}$$

where  $c \in \mathbb{Q}$ ,  $a \in \text{AP}$  is an atomic proposition,  $\sim \in \{<, \leq, =, \geq, >\}$ ,  $\sigma$  is a *state variable* from a countably infinite supply of variables  $\mathcal{V} = \{\sigma_1, \sigma_2, \dots\}$ ,  $p$  is a *probability expression*, and  $\varphi$  is a *path formula*. HyperPCTL path formulas are formed according to the following grammar:

$$\varphi ::= \bigcirc\psi \mid \psi \mathcal{U} \psi \mid \psi \mathcal{U}^{[k_1, k_2]} \psi$$

where  $\psi$  is a state formula and  $k_1, k_2 \in \mathbb{N}_{\geq 0}$  with  $k_1 \leq k_2$ .

As syntactic sugar, we introduce state formulas of the form  $p \in J$ , where  $J = [l, u] \subseteq [0, 1]$  is an interval with rational bounds, defined as  $l \leq p \wedge p \leq u$ . We also define the syntactic sugar  $\psi_1 \mathcal{U}^{\leq k} \psi_2$  for  $\psi \mathcal{U}^{[0, k]} \psi$ . As usual, we furthermore introduce  $\psi_1 \vee \psi_2 = \neg(\neg\psi_1 \wedge \neg\psi_2)$ ,  $\diamond\psi = \mathbf{true} \mathcal{U} \psi$ ,  $\diamond^{[k_1, k_2]} \psi = \mathbf{true} \mathcal{U}^{[k_1, k_2]} \psi$ ,  $\mathbb{P}(\square\psi) = 1 - \mathbb{P}(\diamond\neg\psi)$ , and  $\mathbb{P}(\square^{[k_1, k_2]} \psi) = 1 - \mathbb{P}(\diamond^{[k_1, k_2]} \neg\psi)$ . We denote by  $\mathcal{F}$  the set of all HyperPCTL state formulas.

An occurrence of an indexed atomic proposition  $a_\sigma$  in a HyperPCTL state formula  $\psi$  is *free* if it is not in the scope of a quantifier bounding  $\sigma$  and otherwise *bound*. HyperPCTL *sentences* are HyperPCTL state formulas in which all occurrences of all indexed atomic propositions are bound. HyperPCTL (*quantified*) *formulas* are HyperPCTL sentences.

*Example.* Consider the following formula:

$$\forall\sigma_1.\exists\sigma_2.\mathbb{P}(\diamond a_{\sigma_1}) = \mathbb{P}(\diamond b_{\sigma_2}).$$

This formula holds if for each instantiated state  $s_1$ , there exists another instantiated state  $s_2$ , such that the probability to finally reach a state labeled with  $a$  from  $s_1$  equals the probability of reaching  $b$  from  $s_2$ .

## 2.2 HyperPCTL Semantics

We present the semantics of HyperPCTL based on  $n$ -ary *self-composition* of a DTMC. We emphasize that it is possible to define the semantics in terms of the non-self-composed DTMC, but it will essentially result in a very similar setting, but more difficult to understand.

**Definition 2.** *The  $n$ -ary self-composition of a DTMC  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  is a DTMC  $\mathcal{M}^n = (S^n, \mathbf{P}^n, \text{AP}^n, L^n)$  with*

- $S^n = S \times \dots \times S$  is the  $n$ -ary Cartesian product of  $S$ ,
- $\mathbf{P}^n(s, s') = \mathbf{P}(s_1, s'_1) \cdot \dots \cdot \mathbf{P}(s_n, s'_n)$  for all  $s = (s_1, \dots, s_n) \in S^n$  and  $s' = (s'_1, \dots, s'_n) \in S^n$ ,
- $\text{AP}^n = \cup_{i=1}^n \text{AP}_i$ , where  $\text{AP}_i = \{a_i \mid a \in \text{AP}\}$  for  $i \in [1, n]$ , and
- $L^n(s) = \cup_{i=1}^n L_i(s_i)$  for all  $s = (s_1, \dots, s_n) \in S^n$  with  $L_i(s_i) = \{a_i \mid a \in L(s_i)\}$  for  $i \in [1, n]$ . ■

The satisfaction of a HyperPCTL quantified formula by a DTMC  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  is defined by:

$$\mathcal{M} \models \psi \quad \text{iff} \quad \mathcal{M}, () \models \psi$$

where  $()$  is the empty sequence of states. Thus, the satisfaction relation  $\models$  defines the values of HyperPCTL quantified, state, and path formulas in the context of a DTMC  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  and an  $n$ -tuple  $s = (s_1, \dots, s_n) \in S^n$  of states (which is  $()$  for  $n = 0$ ). Intuitively, the state sequence  $s$  stores instantiations for quantified state variables. Remember that HyperPCTL quantified formulas are sentences. The semantics evaluates HyperPCTL formulas by structural recursion. Quantifiers are instantiated and the instantiated values for state variables are stored in the state sequence  $s$ . To maintain the connection between a state in this sequence and the state variable which it instantiates, we introduce the auxiliary syntax  $a_i$  with  $a \in \text{AP}$  and  $i \in \mathbb{N}_{>0}$ , and if we instantiate  $\sigma$  in  $\exists\sigma.\psi$  or  $\forall\sigma.\psi$  by state  $s$ , then we append  $s$  at the end of the state sequence and replace all  $a_\sigma$  that is bound by the given quantifier by  $a_i$  with  $i$  being the index of  $s$  in the state sequence. We will express the meaning of path formulas based on the  $n$ -ary self-composition of  $\mathcal{M}$ ; the index  $i$  for the instantiation of  $\sigma$  also fixes the component index in which we keep track of the paths starting in  $\sigma$ . The semantics judgment



rules to evaluate formulas in the context of a DTMC  $\mathcal{M} = (S, \mathbf{P}, \mathbf{AP}, L)$  and an  $n$ -tuple  $s = (s_1, \dots, s_n) \in S^n$  of states are the following:

$$\begin{aligned}
\mathcal{M}, s &\models \forall \sigma. \psi && \text{iff } \forall s_{n+1} \in S. \mathcal{M}, (s_1, \dots, s_n, s_{n+1}) \models \psi[\mathbf{AP}_{n+1}/\mathbf{AP}_\sigma] \\
\mathcal{M}, s &\models \exists \sigma. \psi && \text{iff } \exists s_{n+1} \in S. \mathcal{M}, (s_1, \dots, s_n, s_{n+1}) \models \psi[\mathbf{AP}_{n+1}/\mathbf{AP}_\sigma] \\
\mathcal{M}, s &\models \mathbf{true} && \\
\mathcal{M}, s &\models a_i && \text{iff } a \in L(s_i) \\
\mathcal{M}, s &\models \psi_1 \wedge \psi_2 && \text{iff } \mathcal{M}, s \models \psi_1 \text{ and } \mathcal{M}, s \models \psi_2 \\
\mathcal{M}, s &\models \neg \psi && \text{iff } \mathcal{M}, s \not\models \psi \\
\mathcal{M}, s &\models p_1 \sim p_2 && \text{iff } \llbracket p_1 \rrbracket_{\mathcal{M}, s} \sim \llbracket p_2 \rrbracket_{\mathcal{M}, s} \\
\llbracket \mathbb{P}(\varphi) \rrbracket_{\mathcal{M}, s} &= && Pr\{\pi \in \text{Paths}^s(\mathcal{M}^n) \mid \mathcal{M}, \pi \models \varphi\} \\
\llbracket c \rrbracket_{\mathcal{M}, s} &= && c \\
\llbracket p_1 + p_2 \rrbracket_{\mathcal{M}, s} &= && \llbracket p_1 \rrbracket_{\mathcal{M}, s} + \llbracket p_2 \rrbracket_{\mathcal{M}, s} \\
\llbracket p_1 - p_2 \rrbracket_{\mathcal{M}, s} &= && \llbracket p_1 \rrbracket_{\mathcal{M}, s} - \llbracket p_2 \rrbracket_{\mathcal{M}, s} \\
\llbracket p_1 \cdot p_2 \rrbracket_{\mathcal{M}, s} &= && \llbracket p_1 \rrbracket_{\mathcal{M}, s} \cdot \llbracket p_2 \rrbracket_{\mathcal{M}, s}
\end{aligned}$$

where  $\psi$ ,  $\psi_1$ , and  $\psi_2$  are HyperPCTL state formulas; the substitution  $\psi[\mathbf{AP}_{n+1}/\mathbf{AP}_\sigma]$  replaces for each atomic proposition  $a \in \mathbf{AP}$  each free occurrence of  $a_\sigma$  in  $\psi$  by  $a_{n+1}$ ;  $a \in \mathbf{AP}$  is an atomic proposition and  $1 \leq i \leq n$ ;  $p_1$  and  $p_2$  are probability expressions and  $\sim \in \{<, \leq, =, \geq, >\}$ ;  $\varphi$  is a HyperPCTL path formula and  $c$  is a rational constant.

The satisfaction relation for HyperPCTL path formulas is defined as follows, where  $\pi$  is a path of  $\mathcal{M}^n$  for some  $n \in \mathbb{N}_{>0}$ ;  $\psi$ ,  $\psi_1$ , and  $\psi_2$  are HyperPCTL state formulas and  $k_1, k_2 \in \mathbb{N}_{\geq 0}$  with  $k_1 \leq k_2$ :

$$\begin{aligned}
\mathcal{M}, \pi &\models \bigcirc \psi && \text{iff } \mathcal{M}, \pi[1] \models \psi \\
\mathcal{M}, \pi &\models \psi_1 \mathcal{U} \psi_2 && \text{iff } \exists j \geq 0. (\mathcal{M}, \pi[j] \models \psi_2 \wedge \forall i \in [0, j). \mathcal{M}, \pi[i] \models \psi_1) \\
\mathcal{M}, \pi &\models \psi_1 \mathcal{U}^{[k_1, k_2]} \psi_2 && \text{iff } \exists j \in [k_1, k_2]. (\mathcal{M}, \pi[j] \models \psi_2 \wedge \\
&&& \forall i \in [0, j). \mathcal{M}, \pi[i] \models \psi_1)
\end{aligned}$$

Note that each HyperPCTL formula can be transformed into an equivalent formula in prenex normal form  $Q_1 \sigma_1 \dots Q_n \sigma_n. \psi$ , where each  $Q_i \in \{\forall, \exists\}$  is a quantifier,  $\sigma_i$  is a state variable, and  $\psi$  is a quantifier-free HyperPCTL formula. Note furthermore that the semantics assures that each path formula  $\varphi$  is evaluated in the context of a path of  $\mathcal{M}^n$  such that  $1 \leq i \leq n$  for each  $a_i$  in  $\varphi$ .

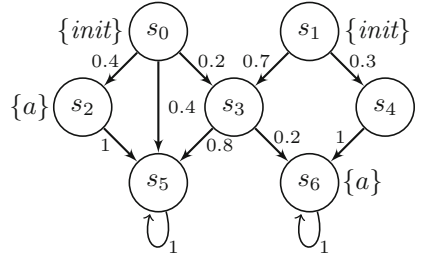


Fig. 1. Semantics example.

*Example.* Consider the DTMC  $\mathcal{M}$  in Fig. 1 and the following HyperPCTL formula:

$$\psi = \forall \sigma. \forall \sigma'. (init_\sigma \wedge init_{\sigma'}) \Rightarrow (\mathbb{P}(\diamond a_\sigma) = \mathbb{P}(\diamond a_{\sigma'}))$$

This formula is satisfied by  $\mathcal{M}$  if for all pairs of initial states (labeled by the atomic proposition *init*), the probability to satisfy  $a$  is the same, i.e., for each  $(s_i, s_j) \in S^2$  with  $init \in L(s_i)$  and  $init \in L(s_j)$  it holds that  $\mathcal{M}, (s_i, s_j) \models \mathbb{P}(\diamond a_1) = \mathbb{P}(\diamond a_2)$ . The probability of reaching  $a$  from  $s_0$  is  $0.4 + (0.2 \times 0.2) = 0.44$ . Moreover, the probability of reaching  $a$  from  $s_1$  is  $0.3 + (0.7 \times 0.2) = 0.44$ . Hence, we have  $\mathcal{M} \models \psi$ .

### 3 HyperPCTL in Action

We now put HyperPCTL into action by formulating probabilistic requirements from different areas, such as information-flow security, privacy, and causality analysis.

#### 3.1 Probabilistic Bisimulation

A *bisimulation* is an equivalence relation over a set of states of a system such that equivalent states cannot be distinguished by observing their behaviors. In the context of DTMC states and PCTL properties, a *probabilistic bisimulation* is an equivalence relation over the DTMC states such that any two equivalent states satisfy the same PCTL formulas. The latter property can be assured inductively by requiring that equivalent states have the same labels and the probability to move from them to any of the equivalence classes is the same.

Assume a partitioning  $S_1, \dots, S_k$  of  $S$  with  $\cup_{i=1}^k S_i = S$  and  $S_i \cap S_j = \emptyset$  for all  $1 \leq i < j \leq k$ . To express that the equivalence relation  $R = \cup_{i=1}^k S_i \times S_i$  is a probabilistic bisimulation, we define  $\mathcal{M}' = (S, \mathbf{P}, \text{AP}', L')$  with  $\text{AP}' = \text{AP} \cup \{a^1, \dots, a^k\}$ , where each  $a^i$ , for all  $i \in [1, k]$ , is a fresh atomic proposition not in  $\text{AP}$ , and for each  $s \in S_i$ , we set  $L'(s) = L(s) \cup \{a^i\}$ . The equivalence relation  $R$  is a bisimulation for  $\mathcal{M}$  if  $\mathcal{M}'$  satisfies the following HyperPCTL formula

$$\varphi_{\text{pb}} = \forall \sigma. \forall \sigma'. \bigwedge_{i=1}^k \left[ (a_{\sigma}^i \wedge a_{\sigma'}^i) \Rightarrow \left[ \psi^{\text{AP}} \wedge \bigwedge_{j=1}^k \mathbb{P}(\bigcirc a_{\sigma}^j) = \mathbb{P}(\bigcirc a_{\sigma'}^j) \right] \right]$$

where  $\psi^{\text{AP}} = \bigwedge_{a \in \text{AP}} (a_{\sigma} \Leftrightarrow a_{\sigma'})$ .

#### 3.2 Probabilistic Noninterference

*Noninterference* is an information-flow security policy that enforces that a low-privileged user (e.g., an attacker) should not be able to distinguish two computations from their publicly observable outputs if they only vary in their inputs by a high-privileged user (e.g., a secret). *Probabilistic noninterference* [14] establishes connection between information theory and information flow by employing probabilities to address covert channels. Intuitively, it requires that the probability of every low-observable trace pattern is the same for every low-equivalent initial state. Probabilistic noninterference can be expressed in HyperPCTL as follows:

$$\varphi_{\text{pni}} = \forall \sigma. \forall \sigma'. \left( l_{\sigma} \wedge l_{\sigma'} \right) \Rightarrow \left( \mathbb{P}(\bigcirc l_{\sigma}) = \mathbb{P}(\bigcirc l_{\sigma'}) \right)$$

where  $l$  denotes a low-observable atomic proposition. Observe that formula  $\varphi_{\text{pni}}$  is a simplification of formula  $\varphi_{\text{pb}}$  in Sect. 3.1, but a stronger form of the noninterference formula for the example in Sect. 1. In fact, most approaches to prove probabilistic noninterference is by showing probabilistic bisimulation with respect to low-observable propositions.

### 3.3 Differential Privacy

*Differential privacy* [6] is a commitment by a data holder to a data subject (normally an individual) that he/she will not be affected by allowing his/her data to be used in any study or analysis. Formally, let  $\epsilon$  be a positive real number and  $\mathcal{A}$  be a randomized algorithm that makes a query to an input database and produces an output. Algorithm  $\mathcal{A}$  is called  $\epsilon$ -*differentially private*, if for all databases  $D_1$  and  $D_2$  that differ on a single element, and all subsets  $S$  of possible outputs of  $\mathcal{A}$ , we have:

$$\Pr[\mathcal{A}(D_1) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{A}(D_2) \in S].$$

Differential privacy can be expressed in HyperPCTL by the following formula:

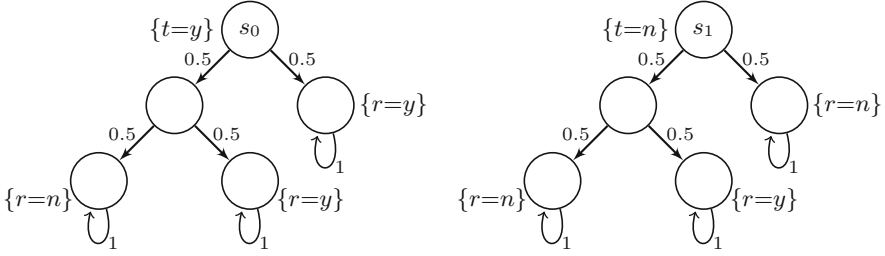
$$\psi_{\text{dp}} = \forall \sigma. \forall \sigma'. \left[ dbSim(\sigma, \sigma') \right] \Rightarrow \left[ \mathbb{P}(\diamond(qOut \in S)_\sigma) \leq e^\epsilon \cdot \mathbb{P}(\diamond(qOut \in S)_{\sigma'}) \right]$$

where  $dbSim(\sigma, \sigma')$  means that two different dataset inputs have all but one similarity and  $qOut$  is the result of the query. For example, one way to provide differential privacy is through *randomized response* in order to create noise and provide plausible deniability. Let  $A$  be an embarrassing or illegal activity. In a social study, each participant is faced with the query, “Have you engaged in activity  $A$  in the past week?” and is instructed to respond by the following protocol:

1. Flip a fair coin.
2. If tail, then answer truthfully.
3. If head, then flip the coin again and respond “Yes” if head and “No” if tail.

Thus, a “Yes” response may have been offered because the first and second coin flips were both heads. This implies that, there are no good or bad responses and an answer cannot be incriminating.

We now show that this social study is  $(\ln 3)$ -differentially private. For each participant in the study, Fig. 2 shows the Markov chain of the response protocol, where  $\{t = y\}$  (respectively,  $\{t = n\}$ ) denotes that the truth is that the participant did (respectively, did not) engage in activity  $A$ , and  $\{r = y\}$  (respectively,  $\{r = n\}$ ) means that the participant responds “Yes” (respectively, “No”).



**Fig. 2.** Markov chain of the randomized response protocol.

The HyperPCTL formula to express  $(\ln 3)$ -differentially privacy for this protocol is the following:

$$\forall \sigma. \forall \sigma'. \left[ \left( (t=n)_\sigma \wedge (t=y)_{\sigma'} \right) \Rightarrow \left( \mathbb{P}(\diamond(r=n)_\sigma) \leq e^{\ln 3} \cdot \mathbb{P}(\diamond(r=n)_{\sigma'}) \right) \right] \wedge \left[ \left( (t=y)_\sigma \wedge (t=n)_{\sigma'} \right) \Rightarrow \left( \mathbb{P}(\diamond(r=y)_\sigma) \leq e^{\ln 3} \cdot \mathbb{P}(\diamond(r=y)_{\sigma'}) \right) \right]$$

Observe that compared to formula  $\psi_{dp}$ , we have decomposed  $dbSim(\sigma, \sigma')$  to two cases of  $t = y$  and  $t = n$ . Thus, in the left conjunct, the set  $S$  represents the case where the response is “No” and in the right conjunct, the set  $S$  represents the case where the response is “Yes”. It is straightforward to see that the DTMC in Fig. 2 satisfies the formula, when for the left conjunct  $\sigma$  and  $\sigma'$  are instantiated by  $s_0$  and  $s_1$ , respectively, and for the right conjunct  $\sigma$  and  $\sigma'$  are instantiated by  $s_1$  and  $s_0$ , respectively.

### 3.4 Probabilistic Causation

*Probabilistic causation* [8] aims to characterize the relationship between *cause* and *effect* using the tools of probability theory. The reason for using probabilities is that most causes are not invariably followed by their effects. For example, smoking is a cause of lung cancer, even though some smokers do not develop lung cancer and some people who have lung cancer are not smokers. Thus, we need to somehow express that some causes are *more likely* to develop an effect. Specifically, the central idea in probabilistic causation is to assert that the probability of occurring effect  $e$  if cause  $c$  happens is higher than the probability of occurring  $e$  when  $c$  does not happen. We can express the most basic type of probabilistic causation in HyperPCTL as follows:

$$\psi_{pc1} = \forall \sigma. \forall \sigma'. c_\sigma \wedge \left( \mathbb{P}(\diamond e_\sigma) > \mathbb{P}(\neg c_{\sigma'} \mathcal{U} e_{\sigma'}) \right).$$

Observe that expressing causation in the standard PCTL by stripping the state quantifiers in formula  $\psi_{pc1}$  will damage the meaning of causation. The resulting PCTL formula captures the causation relation from each initial state in isolation

and it wrongly allows the probability of  $\diamond e$  from one initial state to be less than the probability of  $(\neg c \mathcal{U} e)$  from another initial state.

One problem with formula  $\psi_{\text{pc}_1}$  is spurious correlations. For example, if  $c$  is the drop in the level of mercury in a barometer, and  $e$  is the occurrence of a storm, then the above formula may hold in a system, though  $c$  is not really the cause of  $e$ . In fact, the real cause for both is the drop in atmospheric pressure. To address this problem, we add a constraint, where there should be no further event  $a$  that *screens off*  $e$  from  $c$  [18]:

$$\psi_{\text{pc}_2} = \forall \sigma. \forall \sigma'. \neg \exists \sigma''. c_\sigma \wedge \left( \mathbb{P}(\diamond e_\sigma) > \mathbb{P}(\neg c_{\sigma'} \mathcal{U} e_{\sigma'}) \right) \wedge \bigwedge_{a \in \text{AP} \setminus \{e, c\}} \left[ (a_{\sigma''} \wedge c_{\sigma''}) \wedge \left( \mathbb{P}(\diamond e_{\sigma''}) = \mathbb{P}(\diamond e_\sigma) \right) \right].$$

The negation behind the existential quantifier can be pushed inside to obtain a proper HyperPCTL formula. We note that for simplicity, in formula  $\psi_{\text{pc}_2}$ , propositions  $a$  and  $c$  occur in the same state in  $\sigma''$ . A more general way is to allow  $a$  happen before or simultaneously with  $c$ . Finally, we note that other concepts in probabilistic causation such as Reichenbach's Common Cause Principle and Fork Asymmetry [18] (which emulates the second law of thermodynamics), as well as Skyrms's Background Contexts [20] can be expressed in a similar fashion.

## 4 HyperPCTL Model Checking

In the following, we show that the HyperPCTL model checking problem is decidable by introducing a model checking algorithm. The space complexity of our algorithm is exponential in the number of quantifiers of the input formula, because for  $n$  state quantifiers, we build the  $n$ -ary self-composition of the input DTMC. We are uncertain whether there exists a PSPACE algorithm, but we show the PSPACE-hardness of the problem.

Let  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  be a DTMC and  $\psi$  be a HyperPCTL quantified formula. Let furthermore  $n$  be the number of state quantifiers in  $\psi$  if it has any and let  $n = 1$  otherwise. Informally, our model checking algorithm decides whether  $\mathcal{M} \models \psi$  as follows (detailed pseudo-code is formulated in the Algorithms 1–3):

1. Apply variable renaming such that the quantified state variables are named  $\sigma_1, \dots, \sigma_n$ .
2. Build the self-composition  $\mathcal{M}^n$ .
3. Compute a labeling  $\hat{L}^n(s)$  for all states  $s \in S^n$  of  $\mathcal{M}^n$  as follows. Initially  $\hat{L}^n(s) = \emptyset$  for all  $s \in S^n$  (Line 5 in Algorithm 1). For all sub-formulas  $\psi'$  of  $\psi$  inside-out do the following:
  - If the subformula  $\psi'$  has the form **true**, add **true** to the label sets  $\hat{L}^n(s)$  of all states  $s \in S^n$  (Line 3 in Algorithm 2).
  - If the subformula  $\psi'$  is an atomic proposition  $a_{\sigma_i}$ , add  $a_{\sigma_i}$  to the label set of each state  $s \in S^n$  with  $a_i \in L^n(s)$  (Line 5 in Algorithm 2).



**Algorithm 2.** HyperPCTL model checking algorithm II

---

**Input** : DTMC  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$ , HyperPCTL quantified formula  $\psi$ , non-negative integer  $n$ ,  $\hat{L}^n : S^n \rightarrow 2^{\mathcal{F}}$

**Output** : An extension of  $\hat{L}^n$  to label each state  $s \in S^n$  with sub-formulas of  $\psi$  that hold in  $s$

```

1 Function HyperPCTL( $\mathcal{M}, \psi, n, \hat{L}^n$ )
2   if  $\psi = \text{true}$  then
3      $\hat{L}^n := \hat{L}^n \cup \{\text{true}\}$ 
4   else if  $\psi = a_{\sigma_i}$  then
5      $\hat{L}^n := \hat{L}^n \cup \{a_{\sigma_i}\}$ 
6   else if  $\psi = \psi_1 \wedge \psi_2$  then
7      $\hat{L}^n := \text{HyperPCTL}(\mathcal{M}, \psi_1, n, \hat{L}^n)$ 
8      $\hat{L}^n := \text{HyperPCTL}(\mathcal{M}, \psi_2, n, \hat{L}^n)$ 
9     for all states  $s \in S^n$  with  $\{\psi_1, \psi_2\} \subseteq \hat{L}^n(s)$  set  $\hat{L}^n(s) := \hat{L}^n(s) \cup \{\psi\}$ 
10  else if  $\psi = \neg\psi_1$  then
11     $\hat{L}^n := \text{HyperPCTL}(\mathcal{M}, \psi_1, n, \hat{L}^n)$ 
12    for all states  $s \in S^n$  with  $\psi_1 \notin \hat{L}^n(s)$  set  $\hat{L}^n(s) := \hat{L}^n(s) \cup \{\psi\}$ 
13  else if  $\psi = p_1 \sim p_2$  then
14     $L_1^n := \text{ProbMC}(\mathcal{M}, p_1, n, \hat{L}^n)$            % see Algorithm 3
15     $L_2^n := \text{ProbMC}(\mathcal{M}, p_2, n, \hat{L}^n)$            % see Algorithm 3
16    for all states  $s \in S^n$  with  $L_1^n(s) \sim L_2^n(s)$  set  $\hat{L}^n(s) := \hat{L}^n(s) \cup \{\psi\}$ 
17  else if  $\psi = \exists\sigma_i.\psi_1$  then
18     $\hat{L}^n := \text{HyperPCTL}(\mathcal{M}, \psi_1, n, \hat{L}^n)$ 
19    for all states  $s = (s_1, \dots, s_n) \in S^n$  with  $\psi_1 \in \hat{L}^n(s')$  for some  $s'_i \in S$  and
20     $s' = (s_1, \dots, s_{i-1}, s'_i, s_{i+1}, \dots, s_n)$  set  $\hat{L}^n(s) := \hat{L}^n(s) \cup \{\psi\}$ 
21  else if  $\psi = \forall\sigma_i.\psi_1$  then
22     $\hat{L}^n := \text{HyperPCTL}(\mathcal{M}, \psi_1, n, \hat{L}^n)$ 
23    for all states  $s = (s_1, \dots, s_n) \in S^n$  with  $\psi_1 \in \hat{L}^n(s')$  for all  $s'_i \in S$  and
24     $s' = (s_1, \dots, s_{i-1}, s'_i, s_{i+1}, \dots, s_n)$  set  $\hat{L}^n(s) := \hat{L}^n(s) \cup \{\psi\}$ 
25  return  $\hat{L}^n$ 

```

---

*Proof.* We show that the HyperPCTL model checking problem is PSPACE-hard by reducing the following PSPACE-hard *quantified Boolean formula* (QBF) satisfiability problem [9] to it:

Given is a set  $\{x_1, x_2, \dots, x_n\}$  of Boolean variables and a quantified Boolean formula

$$y = \mathbb{Q}_1 x_1. \mathbb{Q}_2 x_2 \dots \mathbb{Q}_{n-1} x_{n-1}. \mathbb{Q}_n x_n. \psi$$

where  $\mathbb{Q}_i \in \{\forall, \exists\}$  for each  $i \in [1, n]$  and  $\psi$  is an arbitrary Boolean formula over variables  $\{x_1, \dots, x_n\}$ . Is  $y$  true?

**Algorithm 3.** HyperPCTL model checking algorithm III

---

**Input** : DTMC  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$ , HyperPCTL probability expression  $p$ , non-negative integer  $n$ ,  $\hat{L}^n : S^n \rightarrow 2^{\mathcal{F}}$

**Output** :  $L_p^n : S^n \rightarrow \mathbb{Q}$  specifying the values  $L_p^n(s)$  of  $p$  in all states  $s \in S^n$

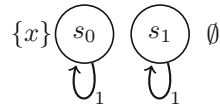
- 1 **Function** *ProbMC*( $\mathcal{M}, p, n, \hat{L}^n$ )
- 2   let  $L_p^n : S^n \rightarrow \mathbb{Q}$  with  $L_p^n(s) = 0$  for all  $s \in S^n$
- 3   **if**  $p = c$  **then**
- 4     for all  $s \in S^n$  set  $L_p^n(s) = c$
- 5   **else if**  $p = p_1 \text{ op } p_2$  with  $\text{op} \in \{+, -, \cdot\}$  **then**
- 6      $L_1^n := \text{probMC}(\mathcal{M}, p_1, n, \hat{L}^n)$
- 7      $L_2^n := \text{probMC}(\mathcal{M}, p_2, n, \hat{L}^n)$
- 8     for each  $s \in S^n$  set  $L_p^n(s) := L_1^n(s) \text{ op } L_2^n(s)$
- 9   **else if**  $p = \mathbb{P}(\varphi)$  **then**
- 10    **if**  $\varphi = \bigcirc\psi$  **then**
- 11     for all  $s \in S^n$  set  $L_p^n(s) = \sum_{s' \in S^n, \psi \in \hat{L}^n(s')} \mathbf{P}^n(s, s')$
- 12    **else if**  $\varphi = \psi_1 \mathcal{U} \psi_2$  **then**
- 13     compute the unique solution  $\nu$  for the following equation system:
- 14     (1)  $p_s = 0$  for all states  $s \in S^n$  with  $\psi_1 \notin \hat{L}^n(s)$  and  $\psi_2 \notin \hat{L}^n(s)$ , or if no state  $s'$  with  $\psi_2 \in \hat{L}^n(s')$  is reachable from  $s$
- 15     (2)  $p_s = 1$  for all states  $s \in S^n$  with  $\psi_2 \in \hat{L}^n(s)$
- 16     (3)  $p_s = \sum_{s' \in S^n} \mathbf{P}^n(s, s') \cdot p_{s'}$  for all other states
- 17     for all  $s \in S^n$  set  $L_p^n(s) = \nu(p_s)$
- 18    **else if**  $\varphi = \psi_1 \mathcal{U}^{[k_1, k_2]} \psi_2$  **then**
- 19     for each  $s \in S^n$  set  $P_0^n(s) = 1$  if  $\psi_2 \in \hat{L}^n(s)$  and  $P_0^n(s) = 0$  otherwise
- 20     **for**  $i = 1$  **to**  $k_2$  **do**
- 21       for each  $s \in S^n$  set  $P_i^n(s) = \sum_{s' \in S^n} \mathbf{P}^n(s, s') \cdot P_{i-1}^n(s')$  if
- 22         $\psi_1 \in \hat{L}^n(s)$  and  $P_i^n(s) = 0$  otherwise
- 23     for all  $s \in S^n$  set  $L_p^n(s) = \sum_{i=k_1}^{k_2} P_i^n(s)$
- 23    **return**  $L_p^n$

---

We reduce the satisfiability problem for a quantified Boolean formula to the model checking problem for a HyperPCTL formula with the same quantifier structure as follows. We define the simple DTMC  $\mathcal{M} = (S, \mathbf{P}, \text{AP}, L)$  shown in Fig. 3, which contains two states  $s_0$  and  $s_1$  and has two paths  $s_0^\omega$  and  $s_1^\omega$ . The HyperPCTL formula in our mapping is the following:

$$\mathbb{Q}_1 \sigma_1. \mathbb{Q}_1 \sigma_2 \dots \mathbb{Q}_{n-1} \sigma_{n-1}. \mathbb{Q}_n \sigma_n. \psi' \quad (1)$$

where  $\psi'$  is constructed from  $\psi$  by replacing every occurrence of a variable  $x_i$  in  $\psi$  by  $x_{\sigma_i}$ . The given quantified Boolean formula is **true** if and only if the DTMC obtained by our mapping satisfies HyperPCTL formula (1). We translate every assignment to the trace quantifiers to a corresponding assignment of the Boolean variables, and



**Fig. 3.** DTMC in the proof of Theorem 2.



vice versa, as follows: Assigning state  $s_0$  ( $s_1$ ) to  $\sigma_i$  means that  $x_i$  is set to **true** (**false**). ■

## 5 Related Work

*Probabilistic noninterference* [13,14] establishes connection between information theory and information flow by employing probabilities to address covert channels. Intuitively, it requires that the probability of every pattern of low-observable trace be the same for every low-equivalent initial state. Most efforts in reasoning about probabilistic noninterference is through probabilistic weak bisimulation (e.g. [21]). More recently, Sabelfeld and Sands [19] introduce a framework to ensure noninterference for multi-threaded programs, where a probabilistic scheduler non-deterministically manages the execution of threads. They introduce operational semantics for a simple imperative language with dynamic thread creation, and how compositionality is ensured.

*Epistemic logic* [7] is a subfield of modal logic that is concerned with reasoning about knowledge. The semantic model of the logic is a Kripke structure, where a set of agents are related with each other based on which states they consider possible. A probabilistic version of the logic [11] assigns a probability function to each agent at each state such that its domain is a non-empty subset of the set of possible states. Epistemic temporal logic has been used to express information-flow security policies (e.g., [2]). The relation between the expressive power of probabilistic epistemic logic and HyperPCTL remains an open question in this paper. Gray and Syverson [15] propose a modal logic for multi-level reasoning about security of probabilistic systems. The logic is axiomatic and is based on the Halpern and Tuttle [11] framework for reasoning about knowledge and probability. The logic is sound, but it may run into undecidability.

Clarkson and Schneider [5] introduce the notion of *hyperproperties*, a set-theoretic framework for expressing security policies. A hyperproperty is a set of sets of traces. In other words, a hyperproperty is a second-order property of properties. The expressive power of hyperproperties do not exceed the second-order logic, but it is currently unclear whether the full power of second-order logic is needed to express hyperproperties of interest. Clarkson and Schneider have shown two fundamental things: (1) a hyperproperty is an intersection of a safety and a liveness hyperproperty, and (2) hyperproperties can express many important requirements such as information-flow security policies (e.g., noninterference, observational determinism, etc.), service-level agreement, etc.

Second-order logic is not verifiable in general, as it cannot be effectively and completely axiomatized. Thus, temporal logics for subclasses of hyperproperties have emerged [4]. HyperLTL and HyperCTL\* allow explicit and simultaneous quantification over multiple paths to LTL and to CTL\*, respectively. As the names suggest, HyperLTL allow quantification of linear traces and HyperCTL\* permits quantification over multiple execution traces simultaneously while allowing branching-time paths for each trace. HyperLTL and HyperCTL\* are not equipped with probabilistic operators and cannot reason about probabilistic systems.

## 6 Conclusion and Future Work

In this paper, we proposed the temporal logic **HyperPCTL** to express and reason about probabilistic hyperproperties. **HyperPCTL** is a natural extension to **PCTL** by allowing explicit and simultaneous quantification over model states. We defined the syntax and semantics and presented a model checking algorithm for discrete-time Markov chains. The complexity of the algorithm is **PSPACE-hard** in the number of quantifiers in the input **HyperPCTL** formula. We presented multiple examples from different domains, where **HyperPCTL** can elegantly express the requirements.

We believe the results in this paper pave the path for new research directions. As for future work, an important unanswered question in this paper is to determine tighter lower and upper bounds for the complexity of **HyperPCTL** model checking in the size of the formula. We believe most of the literature and fundamental lines of research on **PCTL** verification should now be revisited in the context of **HyperPCTL**. Examples include **HyperPCTL** model checking for Markov decision processes (MDPs), Markov chains with costs, parameter synthesis and model repair for probabilistic hyperproperties, **HyperPCTL** conditional probabilities, developing abstraction/refinement, comparing expressive power to existing related logics such as probabilistic epistemic logic [11], etc. An orthogonal direction is deeper investigation of the examples presented in Sect. 3. Each of those areas (e.g., differential privacy and probabilistic causation) deserve more research to develop effective and efficient model checking techniques.

## References

1. Alpern, B., Schneider, F.B.: Defining liveness. *Inf. Process. Lett.* **21**, 181–185 (1985)
2. Balliu, M., Dam, M., Le Guernic, G.: Epistemic temporal logic for information flow security. In: *Proceedings of the 2011 Workshop on Programming Languages and Analysis for Security (PLAS)*, p. 6 (2011)
3. Clarke, E.M., Emerson, E.A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) *Logic of Programs 1981*. LNCS, vol. 131, pp. 52–71. Springer, Heidelberg (1982). <https://doi.org/10.1007/BFb0025774>
4. Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.: Temporal logics for hyperproperties. In: Abadi, M., Kremer, S. (eds.) *POST 2014*. LNCS, vol. 8414, pp. 265–284. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-54792-8\\_15](https://doi.org/10.1007/978-3-642-54792-8_15)
5. Clarkson, M.R., Schneider, F.B.: Hyperproperties. *J. Comput. Secur.* **18**(6), 1157–1210 (2010)
6. Dwork, C., Roth, A.: The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* **9**(3–4), 211–407 (2014)
7. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.: *Reasoning About Knowledge*. The MIT Press, Cambridge (1995)
8. Fetzer, J.H. (ed.): *Probability and Causality*. Synthesis Library. Springer, Dordrecht (1988). <https://doi.org/10.1007/978-94-009-3997-4>

9. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York (1979)
10. Goguen, J.A., Meseguer, J.: Security policies and security models. In: *IEEE Symposium on Security and Privacy*, pp. 11–20 (1982)
11. Halpern, J.Y., Tuttle, M.R.: Knowledge, probability, and adversaries. In: *Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing (PODC)*, pp. 103–118 (1989)
12. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects Comput.* **6**(5), 512–535 (1994)
13. Gray III, J.W.: Probabilistic interference. In: *Proceedings of the 1990 IEEE Symposium on Security and Privacy (S&P)*, pp. 170–179 (1990)
14. Gray III, J.W.: Toward a mathematical foundation for information flow security. *J. Comput. Secur.* **1**(3–4), 255–294 (1992)
15. Gray III, J.W., Syverson, P.F.: A logical approach to multilevel security of probabilistic systems. *Distrib. Comput.* **11**(2), 73–90 (1998)
16. Lamport, L.: Proving the correctness of multiprocess programs. *IEEE Trans. Softw. Eng.* **3**(2) (1977)
17. Pnueli, A.: The temporal logic of programs. In: *Symposium on Foundations of Computer Science (FOCS)*, pp. 46–57 (1977)
18. Reichenbach, H.: *The Direction of Time* (1956)
19. Sabelfeld, A., Sands, D.: Probabilistic noninterference for multi-threaded programs. In: *Proceedings of the 13th IEEE Computer Security Foundations Workshop (CSFW)*, pp. 200–214 (2000)
20. Skyrms, B.: *Causal Necessity*. Yale University Press, New Haven and London (1980)
21. Smith, G.: Probabilistic noninterference through weak probabilistic bisimulation. In: *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSF)*, pp. 3–13 (2003)
22. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: *Proceedings of the 16th IEEE Computer Security Foundations Workshop (CSFW)*, p. 29 (2003)



# How Fast Is MQTT?

## Statistical Model Checking and Testing of IoT Protocols

Bernhard K. Aichernig and Richard Schumi<sup>(✉)</sup>

Institute of Software Technology, Graz University of Technology, Graz, Austria  
{aichernig,rschumi}@ist.tugraz.at

**Abstract.** MQTT is one of the major messaging protocols in the Internet of things (IoT). In this work, we investigate the expected performance of MQTT implementations in various settings. We present a model-based performance testing approach that allows a fast simulation of specific usage scenarios in order to perform a quantitative analysis of the latency. Out of automatically generated log-data, we learn the distributions of latencies and apply statistical model checking to analyse the functional and timing behaviour. The result is a novel testing and verification technique for analysing the performance of IoT protocols. Two well-known open source MQTT implementations are evaluated and compared.

**Keywords:** Statistical model checking · Model-based testing  
Performance · Latency · Internet of things · MQTT · Mosquitto  
emqtt

## 1 Introduction

With the growing popularity of the Internet of Things (IoT), the quality of its underlying infrastructure moves into focus. In particular, the software needs special attention, since it is often exempted from any warranty. In this work, we are investigating implementations of the Message Queuing Telemetry Transport (MQTT) protocol, one of the major machine-to-machine messaging protocols of the IoT. MQTT follows a publish-subscribe pattern and allows clients, e.g., sensors in a smart home, to distribute messages via a central server, called the *broker* [7]. Recently, we have found 18 protocol violations in four open-source MQTT brokers [30]. In this work, we concentrate on performance.

In contrast to previous performance studies [13, 17, 20, 31], we present a statistical model checking (SMC) approach that is able to (1) predict the expected performance on a model, and (2) to verify the prediction on a real system. SMC [21] is a verification method that can answer both, quantitative and qualitative questions. The questions are expressed as properties of a stochastic model which are checked by analysing simulations of this model.

Our method is realised with a property-based testing (PBT) tool that performs the data generation for learning the distributions of broker latencies, the model simulation, and the verification of the system. PBT is a random testing

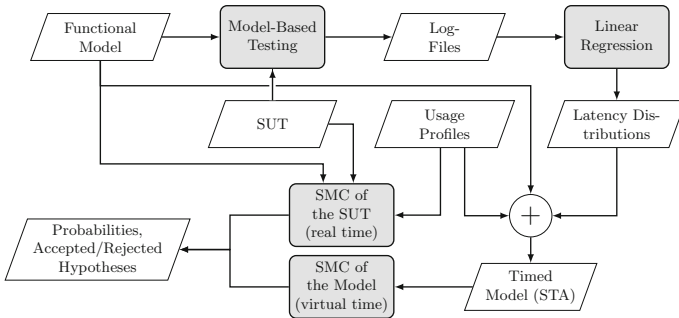
technique that tries to falsify a given property. Properties describe the expected behaviour of a system-under-test (SUT) and may be algebraic or model-based. A PBT tool generates inputs and checks if the property holds.

Previously, we had integrated SMC into a PBT tool [3] in order to check stochastic models as well as implementations. With this technique, we checked the expected response-time of an industrial web application [29]. Based on this previous work, we present a method that statistically verifies the latencies of MQTT brokers from a client’s perspective.

Figure 1 illustrates our method: (1) we automatically test a broker from multiple clients and record its latencies in log-files. For every client, we run a model-based testing process concurrently and generate test cases from a functional model. (2) We derive latency distributions via linear regression. Since the latency is influenced by the parallel activity on the server, the distributions are parametrised by the number of active clients. These latency distributions are added to the functional model resulting in a stochastic timed automata (STA) [6] model. (3) For simulating the behaviour of real MQTT clients, we add usage profiles, containing probabilities and waiting times related to messages. (4) We perform SMC on the resulting stochastic model in order to answer the question “What is the probability that the message latency is under a certain threshold?”. This process can be accelerated by using a virtual time scale, i.e., a fraction of real time. (5) We check if the predicted performance hypothesis holds on the SUT. This is achieved via a statistical hypothesis test (another SMC algorithm). For this final test, less samples than for forming the hypothesis are needed, and hence, this SMC step scales well, although carried out under real time with real delays.

**Related Work.** In contrast to our work, classical load testing methods analyse the performance directly on the SUT. Models are mostly used for test-case generation [5] and for modelling user populations [14, 28]. Others focus solely on simulation on the model-level [8, 9, 11, 23]. In our work, we exploit the models for both, testing a system as well as simulating the performance on the model.

The most related tool is UPPAAL SMC [10], because it supports SMC and test-case generation. However, our use of PBT facilitates the definition of



**Fig. 1.** Overview of the data flow of our method.

specialized generators for complex test-data, which is important for load testing. Furthermore, modelling in a programming language may be more acceptable to programmers and testers.

The performance of MQTT implementations has been tested in the past [13, 20, 31], but without constructing a performance model for simulating MQTT under different usage scenarios. The most similar work to ours [17] modelled MQTT with probabilistic timed automata and checked performance with SMC. However, they did not validate their model against real implementations, and hence, it did not include real timing behaviour.

To the best of our knowledge our work is novel: we are the first who apply SMC to the performance analysis of MQTT brokers with learned latency distributions, and who check the results from the model against real MQTT brokers by performing hypothesis testing.

**Contributions.** This research builds upon our previous work [29], where we introduced our method and applied it to an industrial web-service application. Here, we present the following novel contributions: (1) the evaluation of our method for another application domain, namely for protocol testing. This demonstrates the generality of our method. (2) We present a comparative evaluation of two MQTT implementations. This shows that our method is also able to compare the performance of different systems and helps to choose the right one, depending on a specific usage scenario. (3) This is the first SMC approach for MQTT that also supports a direct verification of the results by testing real MQTT brokers. (4) We release the source of our tool in order to make our method available to the public and to facilitate the reproduction of our results.<sup>1</sup>

**Structure.** First, Sect. 2 introduces the background of SMC and PBT based on our previous work [3]. Next, in Sect. 3 we give an example and demonstrate our method. Then, Sect. 4 presents an evaluation with two open-source MQTT implementations. Finally, we conclude in Sect. 5.

## 2 Background

### 2.1 Statistical Model Checking (SMC)

SMC is a verification method for checking qualitative and quantitative properties of a stochastic model. These properties are usually defined with (temporal) logics. In order to answer questions, like “What is the probability that the model satisfies a property?” or “Is the probability that the model satisfies a property above or below a certain threshold?”, a statistical model checker produces samples, i.e. random walks on the stochastic model and checks whether the property holds for these samples. Various SMC algorithms are applied in order to compute the total number of samples needed to find an answer for a specific question, or to compute a stopping criterion. This criterion determines when we can stop sampling, because we have found an answer with a required certainty. In this work, we focus on the following algorithms common in the SMC literature [21, 22].

<sup>1</sup> <https://github.com/schumi42/mqttCheck>

**Monte Carlo Simulation with Chernoff-Hoeffding Bound.** The algorithm computes the required number of simulations  $n$  in order to estimate the probability  $\gamma$  that a stochastic model satisfies a Boolean property. The procedure is based on the Chernoff-Hoeffding bound [16] that provides a lower limit for the probability that the estimation error is below a value  $\epsilon$ . Assuming a confidence  $1 - \delta$  the required number of simulations is  $n \geq 1/(2\epsilon^2) \ln(2/\delta)$ .

The  $n$  simulations represent Bernoulli random variables  $X_1, \dots, X_n$  with outcome  $x_i = 1$  if the property holds for the  $i$ -th simulation run and  $x_i = 0$  otherwise. Let the estimated probability be  $\tilde{\gamma}_n = (\sum_{i=1}^n x_i)/n$ , then the probability that the estimation error is below  $\epsilon$  is greater than our required confidence. Formally we have:  $Pr(|\tilde{\gamma}_n - \gamma| \leq \epsilon) \geq 1 - \delta$ . After the calculation of the number of required samples  $n$ , a standard Monte Carlo simulation is performed [22].

**Sequential Probability Ratio Test (SPRT).** This sequential method [32] is a form of hypothesis testing that can answer qualitative questions. Given a random variable  $X$  with a probability density function  $f(x, \theta)$ , we want to decide, whether a null hypothesis  $H_0 : \theta = \theta_0$  or an alternative hypothesis  $H_1 : \theta = \theta_1$  is true for desired type I and II errors  $(\alpha, \beta)$ . In order to make the decision, we start sampling and calculate the log-likelihood ratio after each observation of  $x_i$ :

$$\log \Lambda_m = \log \frac{p_1^m}{p_0^m} = \log \frac{\prod_{i=1}^m f(x_i, \theta_1)}{\prod_{i=1}^m f(x_i, \theta_0)} = \sum_{i=1}^m \log \frac{f(x_i, \theta_1)}{f(x_i, \theta_0)}$$

We continue sampling as long as the ratio is inside the indifference region  $\log \frac{\beta}{1-\alpha} < \log \Lambda_m < \log \frac{1-\beta}{\alpha}$ .  $H_1$  is accepted when  $\log \Lambda_m \geq \log \frac{1-\beta}{\alpha}$ , and  $H_0$  when  $\log \Lambda_m \leq \log \frac{\beta}{1-\alpha}$  [15].

In this work, we form a hypothesis about the expected latencies with the Monte Carlo method on the model. Then, we check with SPRT if this hypothesis holds on the SUT. This is faster than running Monte Carlo directly on the SUT.

## 2.2 Property-Based Testing (PBT)

PBT is a random-testing technique that aims to check the correctness of properties. A property is a high-level specification of the expected behaviour of a function- or system-under-test that should always hold. With PBT, inputs can be generated automatically by applying data generators, e.g., a random list generator. The inputs are fed to the function or system-under-test and the property is evaluated. If it holds, then this indicates that the function or system works as expected, otherwise a counterexample is produced.

One of the key features of PBT is its support for model-based testing. Models encoded as extended finite state machines (EFSMs) [19] can serve as source for state-machine properties. An EFSM is a 6-tuple  $(S, s_0, V, I, O, T)$ .  $S$  is a finite set of states,  $s_0 \in S$  is the initial state,  $V$  is a finite set of variables,  $I$  is a finite set of inputs,  $O$  is a finite set of outputs,  $T$  is a finite set of transitions.

A transition  $t \in T$  can be described as a 6-tuple  $(s_s, i, g, op, o, s_t)$ ,  $s_s$  is the source state,  $i$  is an input,  $g$  is a guard,  $op$  is a sequence of assignment operations,  $o$  is an output,  $s_t$  is the target state [19].

In order to derive a state-machine property from an EFSM, we have to write a specification comprising the initial state, commands and a generator for the next transition given the current state of the model. Commands encapsulate (1) preconditions that define the permitted transition sequences, (2) postconditions that specify the expected behaviour and (3) execution semantics of transitions for the model and the SUT. A state-machine property states that for all permitted transition sequences, the postcondition must hold after the execution of each command [18, 25]. Simplified, such properties can be defined as follows:

$$\begin{aligned} & cmd.runModel, cmd.runActual : S \times I \rightarrow S \times O \\ & cmd.pre : I \times S \rightarrow Boolean, cmd.post : (S \times O) \times (S \times O) \rightarrow Boolean \\ & \forall s \in S, i \in I, cmd \in Cmds : \\ & \quad cmd.pre(i, s) \implies cmd.post(cmd.runModel(i, s), cmd.runActual(i, s)) \end{aligned}$$

We have two functions to execute a command on the model and on the SUT:  $cmd.runModel$  and  $cmd.runActual$ . The precondition  $cmd.pre$  defines the valid inputs for a command. The postcondition  $cmd.post$  compares the outputs and states of the model and the SUT after the execution of a command.

PBT is a powerful testing technique that allows a flexible definition of generators and properties via inheritance or composition. The first implementation of PBT was QuickCheck for Haskell [12]. Numerous reimplementations followed for other programming languages. We use FsCheck<sup>2</sup> for C#.

## 2.3 Stochastic Timed Automata

Several probabilistic extensions of timed automata [4] have been proposed. Here, we follow the definition of stochastic timed automata (STA) by Ballarini et al. [6]: an STA is a tuple  $(L, l_0, A, C, I, E, F, W)$  comprising a classical timed automaton  $(L, l_0, A, C, I, E)$ , probability density functions (PDFs)  $F = (f_i)_{i \in L}$  for the sojourn time, and natural weights  $W = (w_e)_{e \in E}$  for the edges.  $L$  is a finite set of locations,  $l_0 \in L$  is the initial location,  $A$  is a finite set of actions,  $C$  is a finite set of clocks with valuations  $u(c) \in \mathbb{R}_{>0}$ ,  $I : L \mapsto \mathcal{B}(C)$  is a finite set of invariants for the locations and  $E \subseteq L \times A \times \mathcal{B}(C) \times 2^C \times L$  is a finite set of edges between locations, with an action, a guard and a set of clock resets.

The transition relation can be described as follows. For a state given by the pair  $(l, u)$ , where  $l$  is a location and  $u$  a clock valuation  $u \in C \rightarrow \mathbb{R}_{\geq 0}$ , the PDF  $f_i$  is used to choose the sojourn time  $d$ , which changes the state to  $(l, u + d)$ , where we lift the plus operator to the clock valuation as follows:  $u + d =_{def} \{c \mapsto u(c) + d \mid c \in C\}$ . After this change, an edge  $e$  is selected out of the set of enabled edges  $E(l, u + d)$  with the probability  $w_e / \sum_{h \in E(l, u + d)} w_h$ . Then, a transition to the target location  $l'$  of  $e$  and  $u' = u + d$  is performed.

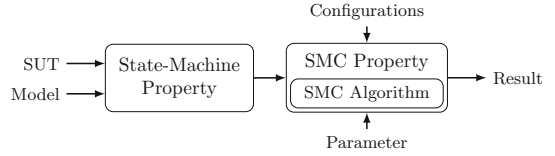
<sup>2</sup> <https://fscheck.github.io/FsCheck>



For our models the underlying stochastic process is a semi-Markov process, since the clocks are reset at every transition, but we do not assume exponential waiting times, and therefore, the process is not a standard continuous-time Markov chain.

## 2.4 Integration of SMC into PBT

Recently, we have demonstrated that SMC can be integrated into a PBT tool in order to perform SMC of PBT properties [3]. With this approach, we can verify stochastic models, like in classical SMC, as well as stochastic implemen-



**Fig. 2.** Data flow diagram of an SMC property.

tations. For the integration, we introduced our own new SMC properties that take a PBT property, configurations for the PBT execution, and parameters for the specific SMC algorithm as input. Then, our properties perform an SMC algorithm by utilizing the PBT tool as simulation environment and they return either a quantitative or qualitative result, depending on the algorithm. Figure 2 illustrates how we evaluate a PBT state-machine property within an SMC property.

Algorithm 1 shows pseudo code of an SMC property for the SPRT (see Sect. 2.1). The inputs of this algorithm are a PBT property, configurations for PBT, probabilities for  $H_0/H_1$  and the type I and type II error parameters  $\alpha, \beta$ . The algorithm produces samples (Line 3) and calculates the log likelihood ratio (Line 4 & 6) repeatedly, until we are outside the indifference region that is defined by  $\alpha$  and  $\beta$  (Line 7). Finally, when we are outside the indifference region, we return  $H_1$  as result, when the ratio is below the lower bound and  $H_0$  otherwise. Our integration method can, e.g., be applied for a statistical conformance analysis by comparing an ideal model to a stochastic faulty implementation or it can also simulate a stochastic model. In this work, we apply it for a performance analysis with the model and for the verification of real brokers.

---

### Algorithm 1. Pseudo code of a SPRT Property.

---

**Input:** *prop*: PBT property for producing a sample, *config*: configuration for checking the property with PBT,  $p_0, p_1$ : probabilities for  $H_0$  and  $H_1$   $\alpha, \beta$ : type I and type II error parameters

```

1: ratio  $\leftarrow$  0
2: do
3:   if prop.Check(config) then                                 $\triangleright$  produces sample and checks result of PBT property
4:     ratio  $\leftarrow$  ratio +  $\log(\frac{p_1}{p_0})$                                  $\triangleright$  calculate the log likelihood ratio
5:   else
6:     ratio  $\leftarrow$  ratio +  $\log(\frac{1-p_1}{1-p_0})$                              $\triangleright$  calculate the log-likelihood ratio
7:   while  $\log \frac{\beta}{1-\alpha} < \textit{ratio} \wedge \textit{ratio} < \log \frac{1-\beta}{\alpha}$                  $\triangleright$  stop when threshold was reached
8:   if ratio  $\geq \log \frac{1-\beta}{\alpha}$  then
9:     return  $H_1$                                                      $\triangleright H_1$  is accepted
10:  else
11:    return  $H_0$                                                      $\triangleright H_0$  is accepted

```

---

### 3 Method

In this section, we show how we derive timed models from logs and how we can apply these models to simulate stochastic usage profiles. The description follows the steps from the overview in Fig. 1.

**Model-Based Testing.** Our SUT is an MQTT broker that allows clients to connect/disconnect, subscribe/unsubscribe to topics and publish messages for such topics. Each of these actions can be performed with a corresponding control message, which is defined by the MQTT standard [7]. We treat the broker as a black box and test it from a client’s perspective.

The upper state machine in Fig. 3 represents the messages that we test. We run multiple of these state machines concurrently, in order to produce log-data that includes latencies for simultaneous messages of several clients. Each transition of the state machine is labelled with an input  $i$ , an optional guard  $g$  / assignment operations  $op$ , and an output  $o$ . Some transition inputs are parametrised with generated data, e.g., a topic for subscribe. We apply PBT generators in order to produce inputs and their required data. Previously, we have demonstrated the data generation for such functional models and also model-based testing [1, 2]. To keep it simple, we assume that a client can only subscribe to topics that it did not subscribe to before (the same for unsubscribe).

In order to manage the subscriptions, we have a global map  $Subs$  that stores the subscription numbers for each topic. This map is needed when publishing, because we want to check if the number of received messages corresponds to the number of subscribed clients. In order to perform this check, we have a second state machine (Fig. 3 bottom) that represents the message receivers. This machine stores the number of received messages in a map  $Received$  that takes the topic concatenated with the message ( $topic\&msg$ ) as key. The map is updated for each message receiver, and when all messages were delivered, then a  $PubFin$  output is produced. For simplicity, we omit some assignment operations, e.g., for a subscriptions set.

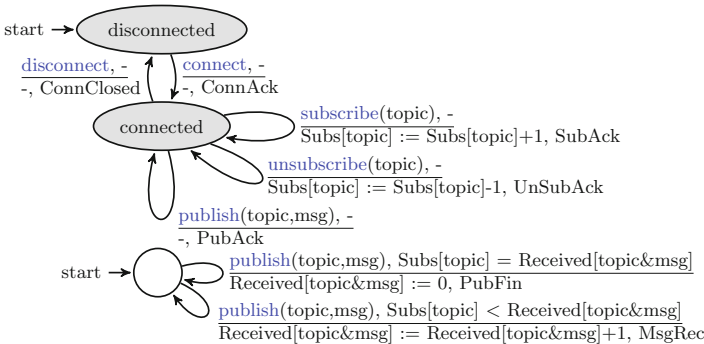


Fig. 3. Functional model for an MQTT client.

Based on this functional model, we perform model-based testing with a PBT tool, which generates random test cases that are executed on an MQTT broker. During this testing phase, we capture the latencies of messages in a log-file. Note that the latency is the duration that a client must wait until it receives a response to a sent message from the broker or until the message is delivered to all receivers in case of a publish.

**Table 1.** Example log-data of one client for Mosquitto.

Msg	#ActiveMsgs	#TotalSubs	TopicSize	MsgSize	#Subs	#Receivers	Latency [ms]
connect	47	266	0	-	-	-	110.82
subscribe	47	270	14	-	-	-	2.45
publish	47	270	14	52	7	7	32.72
unsubscribe	45	12	14	-	-	-	1.25
publish	46	272	14	74	1	1	2.13

A simplified log excerpt from the MQTT implementation Mosquitto is presented in Table 1. It shows that we record the message type ( $Msg$ ), the number of active clients resp. open message exchanges ( $\#ActiveMsgs$ ), the total number of subscriptions ( $\#TotalSubs$ ), the size of the topic ( $TopicSize$ ) and message string ( $MsgSize$ ), the number of subscribers for a topic when a *publish* occurs ( $\#Subs$ ), the number of receivers of a published message ( $\#Receivers$ ), and the latency. For this initial logging phase, the available transitions in the current state of the functional model (Fig. 3 top) are chosen with a uniform distribution. In the *disconnected* state, the only choice is a *connect* message and in the *connected* state all other messages are selected with equal frequency. We do not apply any sojourn times in this phase, since we want to capture the latencies for many concurrent messages.

**Linear Multiple Regression.** In previous work, we showed that linear regression can be applied for learning response-time distributions of a web application [29]. Now, we learn latency distributions with this method.

Linear regression produces a regression model that describes the relationships of the log-variables (or features) with the target variable and can be applied for the prediction of the target variable. The quality of the regression model can be measured with the coefficient of determination ( $R^2$ -score) [24], which defines how well a prediction model for regression fits given data.

First, we checked if we can find any bias in our logs, e.g., a bias might be caused by log-data generation that is not random enough. In this case, we could obtain an artificial correlation between features. Another problem might be that the log-data generation might be unintentionally set up in a way, where relevant scenarios for the prediction were not tested frequently enough. Both these issues can result in a regression model that has a good  $R^2$ -score, but it would not produce reliable predictions for our simulation with SMC. In order to reduce the risk of such biases, it is helpful to carefully analyse the data with visualisations,

like scatter plots, histograms or correlation matrices. For example, if a correlation matrix shows correlations that should not be there, then this might indicate a problem during the initial test-case generation.

In the next step, the *data cleaning*, these data visualisations also helped to find issues with the data. Here, log-entries with disproportionately long latencies, i.e. outliers, are removed. We consider the top 5% of the entries per message type as outlier. Moreover, we flag and remove entries where exceptions were raised, e.g., due to time-outs or connection failures, since they are rare and we are primarily interested in latencies of successful message exchanges.

Next, comes the *feature selection*, where we select variables that have a significant influence on the target variable. We can also apply the correlation matrices and look for features that are correlated with the target variable. The correlation can be measured with a correlation coefficient  $r$ , e.g., a common one was introduced by Pearson [26] and gives us a value  $r \in [-1, 1]$ , where 1 is a total positive correlation and  $-1$  a negative correlation. Features that have a medium or strong correlation  $r \geq 0.3$  are most important for the regression, but sometimes also features with a weak correlation  $0.1 < r \leq 0.3$  can help to improve the regression model. In addition, most regression tools show what features are relevant for the regression, which we will see later. Note, we should avoid features that have a high correlation among each other, since they might be redundant. For example, the number of subscribers to a topic of a published message is highly correlated with the number of message receivers. Hence, we only select one of these. Additionally, we checked if certain features only have an effect on specific message types, which can, e.g., be resolved by setting these features to zero for this message. The selected features can be seen in our regression formula:

$$Latency \sim Msg + \#ActiveMsgs + \#TotalSubs + \#Subs$$

We performed the regression in R with the standard *lm* function.<sup>3</sup> It was performed with log-data from Mosquitto, which contained 100 test cases with a random number of clients (3–100) and a length of 50 messages. This produced log-files with about 300,000 entries. The required number of test cases was determined by stepwise increasing the dataset and by executing the regression, until there was no more increase in the  $R^2$ -score.

Listing 1.1 shows the results of the linear multiple regression. We are mainly interested in the first three columns of this listing. The first col-

	Estimate	Std. Error	t value	Pr(> t )
1 (Intercept)	-8.009707	0.1106356	-72.397	< 2e-16 ***
2 Msgdisconnect	8.084679	0.1234019	65.515	< 2e-16 ***
3 Msgpublish	9.066681	0.1395017	64.993	< 2e-16 ***
4 Msgsubscribe	8.771242	0.1419899	61.774	< 2e-16 ***
5 Msgunsubscribe	9.294850	0.1294843	71.784	< 2e-16 ***
6 #ActiveMsgs	1.358794	0.0033433	406.417	< 2e-16 ***
7 #TotalSubs	0.002503	0.0002084	12.011	< 2e-16 ***
8 #Subs	0.294270	0.0307663	9.565	< 2e-16 ***

**Listing 1.1.** Excerpt of the linear regression output.

umn shows the intercept and the regression coefficients. The intercept is the mean of the latency, when all features are zero and the coefficients come from the features. For categorical variables, e.g., the message type *Msg*, we have multiple coefficients. In the second column, there is the estimate of the mean and

<sup>3</sup> <https://www.r-project.org/>

```

MinTimeBetwMsg: 0, MaxTimeBetwMsg: 500,
MsgWeights:{connect: 1, disconnect: 1, publish: 5, subscribe: 3, unsubscribe: 2}

```

**Listing 1.2.** Usage profile UP1 with time bounds and weights for messages.

the third column shows the standard error that gives the average variation of the estimate from the actual average value. Note that the \*\*\* at the end of each line, shows that the variables are all highly significant. For more details, see [27].

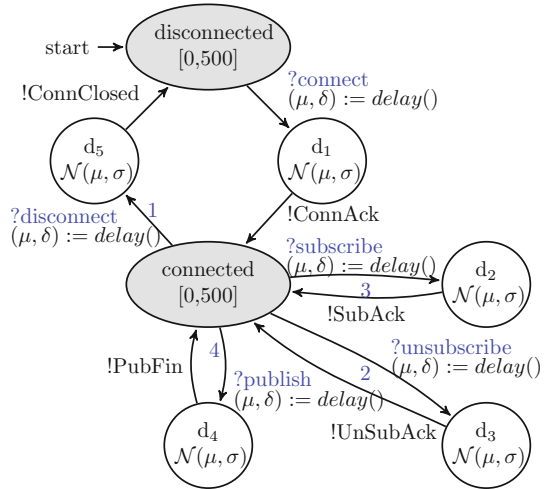
In order to apply this regression model in our method, we encode it in a *delay* function that takes the message type, the number of active messages, the total number of subscribers, and the number of subscribers for the currently published message as input and returns the parameters  $\mu$  and  $\sigma$  of the normal distribution as result:

$$delay : Msg \times \mathbb{N}_{>0} \times \mathbb{N}_{\geq 0} \times \mathbb{N}_{\geq 0} \rightarrow \mathbb{R} \times \mathbb{R}$$

In this function, we perform a linear combination of the distributions given by the estimates and standard errors of the associated regression coefficients for the inputs. This gives us a combined normal distribution that depends on the inputs of this function. For example, for a *subscribe* message that happens when 15 other messages are active and when there are zero subscribers, the linear combination works as follows. The associated regression coefficients of Listing 1.1 (Lines 2, 5 & 7) are combined in order to obtain parameters for a normal distribution  $\mu = -8.010 + 8.771 + 15 \times 1.359$  and  $\sigma = \sqrt{0.111^2 + 0.142^2 + (15 \times 0.003)^2}$ . In the next phase, we integrate the *delay* function that calculates these parameters into the functional model.

### Statistical Model Checking.

For the evaluation of the model, we introduce usage profiles that describe the behaviour of an MQTT client, i.e. how long it should wait between sending messages, and with what probabilities it should send certain messages. An example usage profile (UP1) is shown in Listing 1.2. The time between messages is selected uniformly inside the bounds [MinTimeBetwMsg, MaxTimeBetwMsg] and we have weights that define the message frequency. (In specific settings, it may make sense to create different usage profiles for certain types of components, e.g., a sensor might only publish messages.)



**Fig. 4.** Stochastic timed automaton for the timing behaviour of an MQTT client.

This usage profile is added to the functional model and also the learned latency distributions (expressed in the *delay* function) are integrated. This gives us a combined timed model in the form of a stochastic timed automaton, as explained in Sect. 2.3 and illustrated in Fig. 4. In this model, all locations have probability density functions  $f_l$  for the sojourn time. The *connected* and *disconnected* locations have a uniform distribution given by an upper and lower bound  $[a, b]$ . These bounds come from our usage profile. All other locations have a normal distribution for the sojourn time. The parameters for this distribution are computed by the delay function. In contrast to the functional model, these additional locations apply the message latencies. The locations have one incoming edge that represents sending a message and an outgoing edge for the response. Moreover, the weights  $w_e$  from our usage profile are added to the transitions for sending messages. Note that we have omitted the parameters of the delay function and also some assignments that are necessary for these parameters, in order to keep the figure more readable.

With this model, we can evaluate a usage profile by simulating the expected latencies. Moreover, we can simulate a complete MQTT setup by running multiple models concurrently. A run of the model can be defined as:  $(l_0, u_0) \xrightarrow{d_1, a_1} (l_1, u_1) \xrightarrow{d_2, a_2} \dots$  and it produces a timed trace in the form  $(d_1, a_1), (d_2, a_2), \dots$ , where  $d_i$  is a delay and  $a_i \in A$ . An example trace may look like this:

$(97, connect), (9, ConAck), (344, subscribe), (24, SubAck), (58, subscribe), (64, Suback)$

While we execute the model, we can check properties to answer questions, like “What is the probability that the latency of each interaction of a client within a given MQTT setup is under a certain threshold?”. In order to estimate the probability of such properties, we perform a Monte Carlo simulation with Chernoff-Hoeffding bound. This evaluation requires too many samples to be efficiently executed on the SUT, and hence, we only run it on the model. For example, checking the probability that the latency threshold of 50 ms is satisfied for each client of an MQTT setup with 130 clients with parameters  $\epsilon = 0.05$  and  $\delta = 0.01$ , requires 1060 samples and returns a probability of 0.84, when a test-case length of ten is considered.

Fortunately, the SPRT requires fewer samples, and is therefore, better suited for the evaluation of the SUT. The probability that was computed on the model serves as a hypothesis to be checked on the model, i.e. we check if the SUT is at least as good as predicted. We consider the predicted probability 0.84 as alternative hypothesis and select a probability of 0.74 as null hypothesis, which is 0.1 smaller, because we want to be able to reject the hypothesis that the SUT has a smaller probability. (We select a difference of 0.1, because for this difference our model prediction was close enough to the actual probability of the SUT in most cases, and a smaller difference would need too many samples for an efficient evaluation of the SUT.) By running the SPRT (with 0.01 as type I and II error parameters) for each client, we can check these hypotheses. The alternative hypothesis (probability 0.84) was accepted for all clients and on average 41.15 samples (test cases) were needed for the decision.

**Implementation.** Our method was implemented in a similar way, as described in our previous work [29], where we illustrated how timed models can be executed with PBT. Previously, we introduced custom generators for the simulation of response times, which work in a similar way for latencies. Moreover, we demonstrated the application of user profiles that work in the same way as our usage profiles, and we presented a test-case generation algorithm for PBT that can perform the initial model-based testing phase as well as the execution of our timed model. For brevity, we omit the details of the implementation and refer to our previously mentioned published source code.

## 4 Evaluation

We evaluated our method by applying it to two open-source MQTT implementations: Mosquitto 1.4.15 and emqtt 2.3.5, running with quality of service level one and with the default configurations. We analyse the needed number of samples and the run times. MQTT implementations typically have various settings, e.g., the length of the in-flight message queue or an option to group together TCP packets (Nagle’s algorithm). The influence of such settings might be a potential threat to the validity of our comparison. We worked with the default settings as this is commonly done and we also tried to adapt the mentioned settings to face this threat. A comparison of the regression models and response-time visualisations did not show a difference for the adapted settings. Note that Nagle’s algorithm has no effect, because it only groups messages if acknowledgements are pending. This situation does not occur, since our tests are synchronous, i.e. we always wait for an acknowledgement before sending a new message.

The evaluation was performed on a Windows server (version 2008 R2) with a 2.1 GHz Intel Xeon E5-2620 v4 CPU with 8 Cores and 32 GB RAM. This machine was running the clients and the broker in order to avoid an influence of the network. However, a possible influence of the client processes on the broker might cause a threat to validity of our evaluation. To face this issue, we measured the CPU load, to make sure that it is no bottle neck. During the evaluation, the CPU load was below 60% most of the time, and there were only some rare peaks, where the CPU was over 90%. We also tried to increase the priority of the broker process, but this showed no difference. The RAM usage of the brokers was insignificant since the total RAM of the servers was more than enough.

We applied Visual Studio 2012 with .NET framework 4.5, NUnit 2.64, and FsCheck 2.92 in order to run the tests and for SMC. The library M2Mqtt<sup>4</sup> served as a client interface to facilitate the interaction with the brokers.

We follow the method of Sect. 3, in order to answer the question “What is the probability that the message latency is under a certain threshold?”. Hence, we check the probability that all messages within a sequence of ten messages for all clients of a selected MQTT setup have a latency under this threshold. We perform the analysis as shown in Sect. 3, with the difference that we test Mosquitto and emqtt, and we check various thresholds and different numbers of

<sup>4</sup> <https://m2mqtt.wordpress.com>

```

MinTimeBetwMsg: 50, MaxTimeBetwMsg: 250,
MsgWeights:{connect: 1, disconnect: 1, publish: 7, subscribe: 1, unsubscribe: 1}

```

**Listing 1.3.** Usage profile UP2 with more frequent publish messages.

clients. We apply the same usage profile as before and the regression model for emqtt was similar to the one shown for Mosquitto in Listing 1.1. Additionally, we evaluate another usage profile (UP2), as shown in Listing 1.3 that has a higher weight for *publish* messages and different bounds for the time between messages.

As shown before, we apply a Monte Carlo simulation with Chernoff-Hoeffding bound with parameters  $\epsilon = 0.05$  and  $\delta = 0.01$ , which requires 1060 samples per data point, to evaluate the timed model. The results for Mosquitto and emqtt for both user profiles are shown in Figs. 5 and 6. Table 2 shows the average time needed for these evaluations.

As expected, a decrease in the probability can be observed, when the number of clients increases, and a higher threshold causes a higher probability. The advantage of applying SMC on a model is that it runs much faster than on the SUT. With a virtual time of 1/10 of the actual time, we can perform evaluations that would take hours on the SUT within minutes.

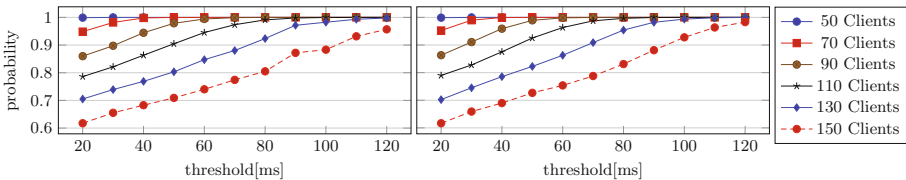
It is also important to check the probabilities that we received through SMC of the timed model, on the SUT. This was done as explained in Sect. 3 with the SPRT. Tables 3 and 4 show the results for both usage profiles and brokers.

We focused on some of the

**Table 2.** Average time [min:s] for the Monte Carlo simulation of the model.

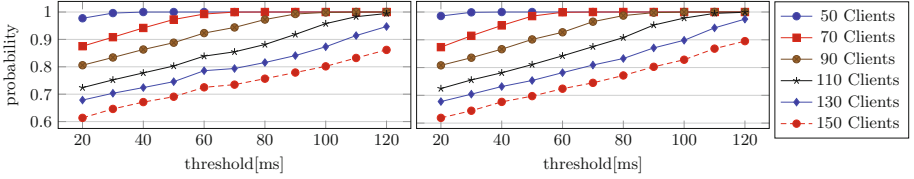
Number of Clients	50	70	90	110	130	150
UP1 Mosquitto	4:27	4:48	4:54	5:00	5:09	5:25
UP1 emqtt	4:28	4:49	4:57	5:05	5:15	5:23
UP2 Mosquitto	2:39	3:03	3:16	3:22	3:40	3:51
UP2 emqtt	2:39	3:02	3:18	3:27	3:41	3:55

more interesting data points for the evaluation. The tables show hypotheses, test results, the needed number of samples and execution times for different numbers of clients and thresholds. Note that in order to obtain an average number of needed samples, we run the SPRT concurrently for each client and calculate the average of these runs.

**Fig. 5.** UP1 Monte Carlo simulation results for Mosquitto (left) & emqtt (right).

In most cases, hypothesis  $H_1$  was accepted for almost all clients, which means that the probability of the SUT was at least as high, as the predicted one from the model. However, the prediction was not always accurate.  $H_0$  was also sometimes





**Fig. 6.** UP2 Monte Carlo simulation results for Mosquitto (left) & emqtt (right).

accepted and in some cases  $H_1$  was only accepted by a fraction of the clients that tested this hypothesis, e.g., for Mosquitto with threshold 30 ms and 90 clients, only 60% of the clients accepted  $H_1$  for UP1. The prediction was sometimes inaccurate for small latency thresholds. The reason might be that we mainly learned the latency distributions under conditions with high load, and hence, our model might not be completely accurate for small latencies.

Moreover, the prediction performed rather poorly for high numbers of clients ( $\geq 130$ ), especially for UP2. This might be caused by the fact that the initial testing phase for log-data had only a maximum of 100 clients and the higher number of clients might be too far away from this initial test phase. However,  $H_1$  was still accepted for most data points, which means that the model was good enough in these cases. Furthermore, it is apparent that the SPRT can be performed with fewer samples, i.e. we need mostly about 50 samples (except for some outliers), compared to the 1060 for the Monte Carlo simulation.

By comparing the results of Mosquitto and emqtt, it can be seen that predicted probabilities are too similar to make a clear distinction. However, the evaluation of the SUT with hypothesis testing was able to find some differences, i.e. in some cases emqtt showed a slightly better performance. For example, the second data row of Table 3 shows that Mosquitto was not able to accept  $H_1$ , where emqtt accepted it, although the same hypotheses were tested. This means that emqtt had a better performance in this case. For UP1, this was the case especially for small thresholds, for UP2 the performance was more similar for both implementations and there is also a case where Mosquitto showed better performance. (Row 13 of Table 4, shows that only 90% of the clients accepted  $H_1$  for emqtt, but all clients for Mosquitto.)

We analysed the execution times of the different phases of our method. The initial testing phase took about 5–8 min and the linear regression about 10–12 s. Note that these two phases have to be performed only once, and the resulting model can then be applied for various evaluations.

A Monte Carlo simulation of the model required about 3–5 min for 1060 samples as shown in Table 2. The evaluation of the SUT with hypothesis testing took most of the time 2–4 min, in some cases about 10 min and in only in one case 39 min. Hence, most of our predictions could be tested efficiently in about the same time that was needed to make the prediction with the timed model.

Running a Monte Carlo simulation with 1060 samples directly on the SUT would take approximately 2–3 h. Performing this simulation becomes quickly impractical when various data points should be analysed. Therefore, our model-based approach makes sense, because it can be executed faster.

**Table 3.** Results of the evaluation of the SUT with the SPRT for UP1.

Threshold	#Clients	Mosquitto					emqtt				
		$H_0$	$H_1$	Result	#Samples	Time [min:s]	$H_0$	$H_1$	Result	#Samples	Time [min:s]
30	50	0.9	1	$H_1$	44	2:31	0.9	1	$H_1$	44	2:28
30	70	0.88	0.98	$H_0$	22.47	5:43	0.88	0.98	$H_1$	44.14	2:51
30	90	0.79	0.89	60% $H_1$	276.31	39:12	0.8	0.9	$H_1$	41.02	2:56
30	110	0.74	0.84	$H_1$	73.26	7:22	0.72	0.82	$H_1$	42.55	3:40
30	130	0.68	0.78	$H_0$	46.68	11:33	0.64	0.74	$H_1$	77.92	9:21
50	50	0.9	1	$H_1$	44	2:10	0.9	1	$H_1$	44	2:06
50	70	0.9	1	73% $H_1$	43.53	10:01	0.9	1	$H_1$	44	2:09
50	90	0.88	0.98	$H_1$	50.47	4:18	0.88	0.98	$H_1$	43	2:30
50	110	0.8	0.9	$H_1$	41.35	3:19	0.84	0.94	$H_1$	41.25	2:50
50	130	0.74	0.84	$H_1$	41.15	3:12	0.75	0.85	$H_1$	38.41	2:37
70	50	0.9	1	$H_1$	44	2:04	0.9	1	$H_1$	44	2:33
70	70	0.9	1	$H_1$	44	2:10	0.9	1	$H_1$	44	2:08
70	90	0.9	1	$H_1$	44	2:37	0.9	1	$H_1$	44	2:29
70	110	0.88	0.98	$H_1$	43.16	2:57	0.89	0.99	$H_1$	44.38	3:14
70	130	0.78	0.88	$H_1$	39.32	3:00	0.83	0.93	$H_1$	41.21	2:37

**Table 4.** Results of the evaluation of the SUT with the SPRT for UP2.

Threshold	#Clients	Mosquitto					emqtt				
		$H_0$	$H_1$	Result	#Samples	Time [min:s]	$H_0$	$H_1$	Result	#Samples	Time [min:s]
30	50	0.9	1	96% $H_1$	42.88	1:18	0.9	1	96% $H_1$	43.42	1:16
30	70	0.88	0.98	$H_0$	17.6	1:15	0.88	0.98	$H_1$	46.4	2:07
30	90	0.8	0.9	$H_0$	17.18	3:55	0.8	0.9	$H_1$	44.98	1:42
30	110	0.72	0.82	$H_0$	13.43	1:39	0.72	0.82	$H_0$	14.61	1:14
30	130	0.65	0.75	$H_0$	14.55	0:56	0.7	0.8	$H_0$	12.68	0:24
50	50	0.9	1	$H_1$	44	1:17	0.9	1	$H_1$	44	1:19
50	70	0.9	1	67% $H_1$	37.94	3:48	0.9	1	$H_1$	44	1:44
50	90	0.88	0.98	$H_1$	51.36	3:01	0.89	0.99	$H_1$	46.2	1:54
50	110	0.79	0.89	$H_1$	41.42	1:46	0.81	0.91	87% $H_1$	152.34	8:34
50	130	0.72	0.82	$H_0$	16.46	0:58	0.76	0.86	$H_0$	9.51	0:23
70	50	0.9	1	$H_1$	44	2:07	0.9	1	$H_1$	44	1:16
70	70	0.9	1	$H_1$	44	2:11	0.9	1	$H_1$	44	1:18
70	90	0.9	1	$H_1$	46.04	2:41	0.9	1	90% $H_1$	52.81	2:47
70	110	0.87	0.97	$H_1$	65.55	4:39	0.88	0.98	$H_1$	43.82	1:58
70	130	0.79	0.89	$H_0$	48.18	5:28	0.81	0.91	$H_0$	9.58	0:34

## 5 Conclusion

We have shown, how to apply SMC in order to predict the performance of MQTT implementations under various usage scenarios. Moreover, we showed how such predictions can be verified by testing real implementations with the SPRT.

First, we collected log-data by running model-based testing with a functional model. Then, we applied linear regression to learn latency distributions that we integrated into our model. Additionally, we combined this model with usage profiles. The resulting model is a stochastic timed automaton that was simulated

to predict the expected latencies of different MQTT implementations. Finally, we verified our prediction with hypothesis testing of the implementations.

A big advantage of our method is that we can predict the performance for various usage scenarios with a fast model simulation and we can efficiently test the prediction on the SUT with the SPRT. The prediction can be accelerated by applying a virtual time that is a fraction of real time, and the test of the SUT is efficient, because it needs fewer samples. Another benefit is that we can do both, SMC and testing of models and SUTs, inside a PBT tool. This enables an easy verification of the model prediction inside the same tool and it facilitates the model and property definition in a high-level programming language.

We have evaluated our method by applying it to well-known open-source implementations of MQTT: Mosquitto and emqtt, and the results were promising. We analysed various numbers of clients and checked the probability that the latency is within certain thresholds. Moreover, we demonstrated that the predicted probability was accurate in most cases and we showed that emqtt has better performance in some cases.

In the future, we plan to evaluate different learning methods for latency distributions and we envisage to test various types of usage profiles.

**Acknowledgements.** This work was supported by the TU Graz LEAD project “Dependable Internet of Things in Adverse Environments”. We are grateful to our colleague Martin Tappler and to the anonymous reviewers for their excellent feedback that helped in improving the quality of the paper.

## References

1. Aichernig, B.K., Schumi, R.: Property-based testing with FsCheck by deriving properties from business rule models. In: ICSTW, pp. 219–228. IEEE (2016)
2. Aichernig, B.K., Schumi, R.: Property-based testing of web services by deriving properties from business-rule models. *Softw. Syst. Model.* (2017)
3. Aichernig, B.K., Schumi, R.: Statistical model checking meets property-based testing. In: ICST, pp. 390–400. IEEE (2017)
4. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**(2), 183–235 (1994)
5. Arts, T.: On shrinking randomly generated load tests. In: Erlang 2014, pp. 25–31. ACM (2014)
6. Ballarini, P., Bertrand, N., Horváth, A., Paolieri, M., Vicario, E.: Transient analysis of networks of stochastic timed automata using stochastic state classes. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 355–371. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40196-1\\_30](https://doi.org/10.1007/978-3-642-40196-1_30)
7. Banks, A., Gupta, R.: MQTT version 3.1.1. OASIS Standard, December 2014
8. Becker, S., Koziolok, H., Reussner, R.H.: The Palladio component model for model-driven performance prediction. *J. Syst. Softw.* **82**(1), 3–22 (2009)
9. Book, M., Gruhn, V., Hülder, M., Köhler, A., Kriegel, A.: Cost and response time simulation for web-based applications on mobile channels. In: QSIC, pp. 83–90. IEEE (2005)

10. Bulychev, P.E., et al.: UPPAAL-SMC: statistical model checking for priced timed automata. In: QAPL, EPTCS, vol. 85, pp. 1–16. Open Publishing Association (2012)
11. Chen, X., Mohapatra, P., Chen, H.: An admission control scheme for predictable server response time for web accesses. In: WWW, pp. 545–554. ACM (2001)
12. Claessen, K., Hughes, J.: QuickCheck: a lightweight tool for random testing of Haskell programs. In: ICFP, pp. 268–279. ACM (2000)
13. Collina, M., Corazza, G.E., Vanelli-Coralli, A.: Introducing the QEST broker: scaling the IoT by bridging MQTT and REST. In: PIMRC, pp. 36–41. IEEE (2012)
14. Draheim, D., Grundy, J.C., Hosking, J.G., Lutteroth, C., Weber, G.: Realistic load testing of web applications. In: CSMR, pp. 57–70. IEEE (2006)
15. Govindarajulu, Z.: Sequential Statistics. World Scientific (2004)
16. Hoeffding, W.: Probability inequalities for sums of bounded random variables. *J. Am. Stat. Assoc.* **58**(301), 13–30 (1963)
17. Houimli, M., Kahloul, L., Benaoun, S.: Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things. In: ICMIT, pp. 214–221. IEEE, December 2017
18. Hughes, J.: QuickCheck testing for fun and profit. In: Hanus, M. (ed.) PADL 2007. LNCS, vol. 4354, pp. 1–32. Springer, Heidelberg (2006). [https://doi.org/10.1007/978-3-540-69611-7\\_1](https://doi.org/10.1007/978-3-540-69611-7_1)
19. Kalaji, A.S., Hierons, R.M., Swift, S.: Generating feasible transition paths for testing from an extended finite state machine. In: ICST, pp. 230–239. IEEE (2009)
20. Lee, S., Kim, H., Hong, D., Ju, H.: Correlation analysis of MQTT loss and delay according to QoS level. In: ICOIN, pp. 714–717. IEEE (2013)
21. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: an overview. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-16612-9\\_11](https://doi.org/10.1007/978-3-642-16612-9_11)
22. Legay, A., Sedwards, S.: On statistical model checking with PLASMA. In: TASE, pp. 139–145. IEEE (2014)
23. Lu, Y., Nolte, T., Bate, I., Cucu-Grosjean, L.: A statistical response-time analysis of real-time embedded systems. In: RTSS, pp. 351–362. IEEE (2012)
24. Nagelkerke, N.J.: A note on a general definition of the coefficient of determination. *Biometrika* **78**(3), 691–692 (1991)
25. Papadakis, M., Sagonas, K.: A PropEr integration of types and function specifications with property-based testing. In: Erlang 2011, pp. 39–50. ACM (2011)
26. Pearson, K.: Note on regression and inheritance in the case of two parents. *Proc. Roy. Soc. Lond.* **58**, 240–242 (1895)
27. Rencher, A., Christensen, W.: *Methods of Multivariate Analysis*. Wiley (2012)
28. Tyagi, R.S.: A comparative study of performance testing tools. *Int. J. Adv. Res. Comput. Sci. Softw. Eng. IJARCSSE* **3**(5), 1300–1307 (2013)
29. Schumi, R., Lang, P., Aichernig, B.K., Krenn, W., Schlick, R.: Checking response-time properties of web-service applications under stochastic user profiles. In: Yevtushenko, N., Cavalli, A.R., Yenigün, H. (eds.) ICTSS 2017. LNCS, vol. 10533, pp. 293–310. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-67549-7\\_18](https://doi.org/10.1007/978-3-319-67549-7_18)
30. Tappier, M., Aichernig, B.K., Bloem, R.: Model-based testing IoT communication via active automata learning. In: ICST, pp. 276–287. IEEE (2017)
31. Thangavel, D., Ma, X., Valera, A.C., Tan, H., Tan, C.K.: Performance evaluation of MQTT and CoAP via a common middleware. In: ISSNIP, pp. 1–6. IEEE (2014)
32. Wald, A.: *Sequential analysis*. Courier Corporation (1973)



# Parameter-Independent Strategies for pMDPs via POMDPs

Sebastian Arming<sup>1</sup>(✉), Ezio Bartocci<sup>2</sup>, Krishnendu Chatterjee<sup>3</sup>,  
Joost-Pieter Katoen<sup>4</sup>, and Ana Sokolova<sup>1</sup>

<sup>1</sup> University of Salzburg, Salzburg, Austria  
sarming@cs.uni-salzburg.at

<sup>2</sup> TU Wien, Vienna, Austria

<sup>3</sup> IST Austria, Klosterneuburg, Austria

<sup>4</sup> RWTH Aachen University, Aachen, Germany

**Abstract.** Markov Decision Processes (MDPs) are a popular class of models suitable for solving control decision problems in probabilistic reactive systems. We consider parametric MDPs (pMDPs) that include parameters in some of the transition probabilities to account for stochastic uncertainties of the environment such as noise or input disturbances.

We study pMDPs with reachability objectives where the parameter values are unknown and impossible to measure directly during execution, but there is a probability distribution known over the parameter values. We study for the first time computing parameter-independent strategies that are expectation optimal, i.e., optimize the expected reachability probability under the probability distribution over the parameters. We present an encoding of our problem to partially observable MDPs (POMDPs), i.e., a reduction of our problem to computing optimal strategies in POMDPs.

We evaluate our method experimentally on several benchmarks: a motivating (repeated) *learner model*; a series of benchmarks of varying configurations of a robot moving on a grid; and a consensus protocol.

## 1 Introduction

Markov decision processes (MDPs) [5] are a popular class of models suitable for solving decision making and dependability problems in a randomized environment. An MDP is a state-based model representing a probabilistic process that satisfies the Markov property (*memorylessness*), where for each state it is possible to choose nondeterministically some action-labeled transitions governing the probability distribution to end up in the next state. MDPs are employed in

---

This work was supported by the Austrian FWF (National Research Network RiSE/SHiNE S11405-N23, S11407-N23 and S11411-N23) and by the RTG 2236 UnRAVeL funded by the German Research Foundation.

several applications including the analysis of queueing systems [38], bird flocking [30], confidentiality [6] and robotics [32].

One of the main problems of interest for MDPs is the synthesis of an optimal policy (scheduler, strategy) choosing the sequence of actions that maximizes/minimizes the probability or the expected accumulated reward/cost to reach a target state. Model-checking tools such as PRISM [28] or Storm [20] provide a push-button technology to automate such analyses and to derive simple deterministic and memoryless schedulers.

We study here *parametric* Markov decision processes (pMDPs) [16, 22], in which (some of) the transition probabilities depend on a set of parameters. This class allows to include unknown quantities in the model such as the fault rate or the input disturbances that are responsible for stochastic uncertainty. These quantities are often unavailable at the design time or impossible to measure directly at runtime. Intuitively, a pMDP represents a family of MDPs—one for each possible valuation of the parameters.

In the past years, there has been a great effort to solve reachability analysis in pMDPs using symbolic approaches [2, 16, 17, 22, 35, 36]. These methods generally partition the parameter space in regions, associating each region to the optimal memoryless scheduler that maximizes/minimizes the probability to reach the target state. The common assumption of all these approaches is the possibility to observe the unknown quantities at some point and then to choose accordingly the best scheduler. However, this is not always feasible.

*Our Contribution.* We analyze pMDPs with reachability objectives<sup>1</sup> without assuming that parameter values are accessible directly during execution. Specifically, *we find parameter-independent strategies that are expectation  $\varepsilon$ -optimal [2], for  $\varepsilon \geq 0$ , i.e., optimize the expected reachability probability given a probability distribution over the parameters.* To achieve this goal, we consider partially observable Markov decision processes (POMDPs) where an agent cannot directly observe the environment’s state and must take decisions according to its belief on the current state; the belief can be updated by interacting with the environment.

We provide an encoding of a pMDP as a POMDP where the states consist of pairs of the original pMDP states and parameter values and transitions can occur only between states with the same parameter values. We prove that solving the induced POMDP corresponds to finding parameter-independent expectation  $\varepsilon$ -optimal policies for the pMDP. Note that here memoryless policies are not sufficient for optimality, see the discussion on the motivating learner model in Sect. 5. We leverage algorithms such as *point-based value iteration* (PBVI) [34] and *Incremental Pruning* (IP) [10] to find the solution. We have implemented our approach using Storm [20] and AI-Toolbox [7].

Finally, we evaluate our approach experimentally on several benchmarks: a motivating (repeated) *learner model*; a series of benchmarks of varying configurations of a robot moving on a grid; and a consensus protocol model.

---

<sup>1</sup> We can deal with objectives beyond reachability as long as they are induced by a reward structure (for applicability of the available tools), see Sects. 3 and 4.

*Paper Organization.* Sect. 2 discusses related work. In Sect. 3 we introduce MDPs, pMDPs, and POMDPs. Section 4 presents the encoding of a pMDP in POMDP and the reduction result. In Sect. 5 we illustrate our approach on several case studies and report on experimental results, while in Sect. 6 we wrap up with conclusions and discussion of future work.

## 2 Related Work

Parametric probabilistic models [18, 29] are a special class of Markov models where some of the transition probabilities (or rates) depend on one or more parameters that are not known a-priori. These models are particularly useful to study systems characterized by stochastic uncertainty due to the impossibility to access certain quantities (e.g., fault rates, packet loss ratios, etc.).

In the last decade, there was a great effort to study the problem of symbolic model checking of parametric probabilistic Markov chains [8, 18, 19, 21, 25, 33, 36]. In [18], Daws introduced a method to express the probability to reach the target state as a multivariate rational function with the domain in the parameter space. This approach was then efficiently implemented in the PARAM1 and PARAM2 tools [21] and included later on in the PRISM model checker [28].

The *parameter synthesis* problem consists of (exploiting the generated multivariate rational function and) finding the parameter values that would maximize or minimize the probability to reach the target state.

*Repairing* a probabilistic model [8] consists instead of solving a constrained nonlinear optimization problem where the objective function represents the minimal change in the transition probabilities such that the probability to reach the target state is constrained to a given bound.

The price to pay for these techniques is the increasing complexity of the multivariate rational functions in the presence of large models [27, 29], causing the parameter synthesis to be also very computationally expensive. However, the introduction of new efficient heuristics [19, 25, 33, 36] has helped to alleviate this problem by supporting the parameter synthesis for quite large models.

This symbolic approach to parameter synthesis has been recently extended to handle also the nondeterministic choice in parametric Markov decision processes (pMDPs) [2, 16, 17, 22, 35, 36], where each different sequence of inputs can induce a distinct Markov chain, resulting potentially in several multivariate rational functions.

The parameter synthesis problem for pMDP consists in solving a nonlinear program (NLP) with multiple objectives. Recently, the authors in [17] have shown that many NLPs related to pMDP belong to a certain class of nonconvex optimization problems called signomial programs (SGPs). In the same paper, they have also introduced an approach to relax nonconvex constraints in SGPs generating geometric programs, a particular class of convex programs that can be solved in a number of steps that is polynomial in the number of variables.

Another approach proposed in [16] is based on sampling techniques (i.e., Metropolis-Hastings algorithm, particle swarm optimization, and cross-entropy method) that are used to search the parameter space. These heuristics usually do not guarantee that global optimal parameters will be found. Furthermore, when the regions of the parameters satisfying a requirement are very small, a large number of simulations is required.

All the proposed methods provide a map that relates the regions of the parameter space to the optimal memoryless scheduler that maximizes/minimizes the probability to reach the target state. The underlying assumption for these approaches is the possibility to measure the parameters during system execution. Once the values of the parameters have been measured, one can use this map to choose the best policy. In this paper, we consider a different assumption with respect to the state of the art: *We want to synthesize an expectation  $\varepsilon$ -optimal scheduler for pMDP that is independent from the possibility to measure the parameters.* To achieve this goal we show how to recast the problem into finding an  $\varepsilon$ -optimal policy for a partially observable Markov decision process (POMDP) after providing a suitable encoding. While we are not aware of any other work that establishes such a correspondence, it is worth mentioning that instead parameter synthesis for parametric Markov chains has been recently employed to find permissive finite-state controllers for POMDPs in [26].

The qualitative analysis of POMDPs has been widely studied: complexity results have been established [4, 11] and symbolic algorithms [13]. However, for the general quantitative problem and its approximation the computational questions are undecidable [12, 31]. Despite the undecidability there are several practical approaches such as point-based methods [39].

### 3 The Models

In this section we introduce the models of importance for this paper: MDPs, parametric MDPs, and partially observable MDPs. The models can be arbitrarily large, we do not impose restrictions on the size for the theoretical part of the paper.

#### 3.1 Markov Decision Processes – MDPs

A (discrete) probability distribution on a set  $S$  is a function  $\mu: S \rightarrow [0, 1]$  with the property  $\sum_{s \in S} \mu(s) = 1$ . By  $\mathcal{D}S$  we denote the set of all (discrete) probability distributions on  $S$ . For  $s \in S$ , we write  $\delta_s$  for the Dirac distribution that assigns 1 to  $s$ .

**Definition 3.1 (Markov Decision Process – MDP).** *A Markov Decision Process (MDP) is a tuple  $M = (S, A, T, i)$  where:*

- $S$  is a set of states,
- $A$  is a set of actions,
- $T: S \times A \rightarrow \mathcal{D}S$  is the transition function, and
- $i \in \mathcal{D}S$  is the initial state distribution.

◇



From a given state with a given label, an MDP makes a step to a probability distribution over states that describes the probability of reaching a next state. As usual, we write  $s \xrightarrow{a} \mu$  for  $\mu = T(s, a)$ . We will write  $s \xrightarrow{a,p} t$  for  $s \xrightarrow{a} \mu$  and  $p = \mu(t)$ , as well as  $s \xrightarrow{a} t$  for  $s \xrightarrow{a,p} t$  with  $p > 0$ . It is also common to write  $T(t|s, a)$  for  $T(s, a)(t)$ .

*Remark 3.1.* In our definition no action is disabled in any state. This is somewhat unusual for MDPs, but very common for partially observable MDPs which we are interested in.

A *run* (also called path or play) of an MDP is an infinite sequence  $s_0, a_0, s_1, a_1, \dots$  in  $(S \times A)^\omega$  of states and actions such that  $s_i \xrightarrow{a_i} s_{i+1}$  for all  $i \geq 0$ . When convenient, we will also write  $s_0 \cdot a_0 \cdot s_1 \cdot a_1 \dots$  for the run  $s_0, a_0, s_1, a_1, \dots$ . A *history*  $h$  is a finite prefix of a run in  $(S \times A)^* \times S$ . We write  $\text{first}(h)$  and  $\text{last}(h)$  for the first and last state in a history  $h$ , respectively. The *cone*  $\text{Cone}(h)$  of a history  $h$  is the set of all runs with prefix  $h$ . We write  $\text{Runs}$  for the set of all runs,  $\text{Hist}$  for the sets of all histories, and  $\text{Cones}$  for the smallest  $\sigma$ -algebra containing the cones of all histories.

**Definition 3.2 (MDP policy).** *A policy (strategy, scheduler)  $\pi$  for an MDP  $M$  is a map*

$$\pi: \text{Hist} \rightarrow \mathcal{D}A$$

*from histories to probability distributions over the actions. It is a deterministic policy if the image of  $\pi$  consists only of Dirac distributions. It is a memoryless (or Markov) policy if  $\pi(w \cdot s) = \pi(s)$  for  $w \in (S \times A)^*$  and  $s \in S$ .  $\diamond$*

A policy  $\pi$  together with the initial state distribution  $i \in \mathcal{D}S$  induces a probability space  $(\text{Runs}, \text{Cones}, \mathbb{P}_{\pi, i})$ , i.e., a probability measure  $\mathbb{P}_{\pi, i}$  on the measurable space  $(\text{Runs}, \text{Cones})$ . This construction is done in several steps: First, for a given state  $s$ , we consider a function  $\mathbb{P}_{\pi, s}$  assigning a number in  $[0, 1]$  to cones of histories in  $\text{Hist}$ . It is defined inductively as follows. We set  $\mathbb{P}_{\pi, s}(\text{Cone}(h)) = 1$  if  $h = s$  and  $\mathbb{P}_{\pi, s}(\text{Cone}(h)) = 0$  if  $h = t \neq s$ . For  $h = w \cdot a \cdot t$  we set  $\mathbb{P}_{\pi, s}(\text{Cone}(h)) = \mathbb{P}_{\pi, s}(\text{Cone}(w)) \cdot \pi(w)(a) \cdot T(\text{last}(w), a)(t)$ . By Carathéodory's extension theorem, the function  $\mathbb{P}_{\pi, s}$  extends to a unique measure on the measurable space  $(\text{Runs}, \text{Cones})$  which we denote by  $\mathbb{P}_{\pi, s}$ . Finally, given the initial state distribution  $i \in \mathcal{D}S$ ,  $\mathbb{P}_{\pi, i}$  is the measure on  $(\text{Runs}, \text{Cones})$  defined as  $\mathbb{P}_{\pi, i} = \sum_{s \in S} i(s) \cdot \mathbb{P}_{\pi, s}$ .

We write  $\mathbb{E}_{\pi, i}$  for the expectation operator of  $\mathbb{P}_{\pi, i}$ . Recall that the expectation operator of a measure  $\mu$  on a measurable space  $(X, \Sigma)$  is defined as  $E_\mu(f) = \int f d\mu$  for a measurable function  $f: X \rightarrow \mathbb{R}$  where we consider the Borel  $\sigma$ -algebra on the reals. Hence  $\mathbb{E}_{\pi, i}(f) = \int f d\mathbb{P}_{\pi, i}$  for a measurable function  $f: \text{Runs} \rightarrow \mathbb{R}$ . We may sometimes decorate the notation of the measures and the expectation operators by superscript  $M$ , to emphasize the involved model.

We can now specify what it means to solve an MDP.

**Definition 3.3 (Objective, value, solution of MDP).** *Given an MDP  $M = (S, A, T, i)$ , a Borel objective, also called return, is a measurable function  $r: \text{Runs} \rightarrow \mathbb{R}$ . The value of the MDP  $M$  for the objective  $r$  is defined as  $\text{Val}(r) = \sup_{\pi} \mathbb{E}_{\pi, i}(r)$ . A solution to an MDP  $M$  regarding the objective  $r$  is a policy  $\pi$  with  $\mathbb{E}_{\pi, i}(r) = \text{Val}(r)$ . A policy  $\pi$  is an  $\varepsilon$ -solution, for  $\varepsilon > 0$ , of  $M$  with respect to  $r$  if  $\mathbb{E}_{\pi, i}(r)$  is  $\varepsilon$ -close to  $\text{Val}(r)$ .  $\diamond$*

Note that a solution to an MDP need not exist. We will say objective for a Borel objective. Some objectives arise via the *payoff* or *accumulated reward* of runs. For MDPs with such reward-based objectives, a solution always exists. Solving (partially observable) MDPs often refers to solving reward-based objectives.

**Definition 3.4 (MDP with rewards).** *An MDP with rewards is a tuple  $M_R = (M, R)$  where  $M = (S, A, T, i)$  is an MDP and  $R: S \times A \times S \rightarrow \mathbb{R}$  is the reward function.  $\diamond$*

Upon performing a transition, an MDP with rewards collects reward as described by the reward function. We will sometimes write  $s \xrightarrow{a, p, r} t$  for  $s \xrightarrow{a, p} t$  and  $R(s, a, t) = r$ , and  $s \xrightarrow{a, r} t$  for  $s \xrightarrow{a} t$  and  $R(s, a, t) = r$ . If clear from the context, we will drop the subscript  $R$  in an MDP  $M_R$  with rewards.

Reward structures may induce objectives as follows: The (undiscounted) accumulated reward of a run in an MDP with rewards is

$$r_R(s_0, a_0, s_1, a_1 \dots) = \sum_{i \geq 0} R(s_i, a_i, s_{i+1}). \quad (1)$$

The accumulated reward induces the *reward objective*  $r_R$  if the above assignment defines a measurable function  $r_R: \text{Runs} \rightarrow \mathbb{R}$ .<sup>2</sup>

### Reachability Objectives

Of special interest to us is optimizing *reachability*, that is optimizing the probability to reach a target state (or a set of target states). Computing extremal (i.e., maximal/minimal) reachability probabilities is at the heart of MDP model checking: PCTL and LTL model checking boil down to computing reachability probabilities. The same holds for omega-regular properties: determining the maximal probability of any omega-regular property  $\varphi$  in an MDP  $M$  amounts to determining the maximal probability to reach an accepting end component in the product of  $M$  with a deterministic omega-automaton for  $\varphi$ . Verifying PCTL properties under fair policies, i.e., policies that can almost surely reach a state satisfying some fairness constraint, can also be reduced to computing reachability probabilities.

<sup>2</sup> The discounted accumulated reward objective is defined in a similar way, by adding a factor  $\gamma^i$  to the  $i$ -th summand in (1) with  $\gamma \in [0, 1)$  being the discount factor. For solving reachability objectives, undiscounted rewards are sufficient.

**Definition 3.5 (Reachability objective).** Let  $M = (S, A, T, i)$  be an MDP and  $t \in T$  a state. The reachability objective  $r_t$  of reaching the state  $t$  is given by the indicator function of the set  $\text{Runs}_t$  of runs that reach  $t$ , i.e.,

$$\text{Runs}_t = \{s_0, a_0, s_1, a_1, \dots \in \text{Runs} \mid \exists i \geq 0. s_i = t\}$$

and  $r_t(\rho) = 1$  if  $\rho \in \text{Runs}_t$  and  $r_t(\rho) = 0$  otherwise.  $\diamond$

Note that  $\text{Runs}_t$  is a measurable set, i.e.,  $\text{Runs}_t \in \text{Cones}$  and hence  $r_t$  is a measurable function. Moreover, a solution to the reachability objective  $r_t$  is a policy  $\pi$  that maximizes the reachability probability, as  $\mathbb{E}_{\pi, i}(r_t) = \mathbb{P}_{\pi, i}(\text{Runs}_t)$ .

Reachability objectives are induced by reward structures as follows. Given an MDP  $M = (S, A, T, i)$  and a target state  $t \in S$  we construct the MDP with rewards  $M_t = (S, A, T_t, i, R_t)$  where

$$T_t(s, a) = \begin{cases} \delta_t & \text{if } s = t \\ T(s, a) & \text{otherwise} \end{cases} \quad R_t(s, a, s') = \begin{cases} 1 & \text{if } s \neq t \text{ and } s' = t \\ 0 & \text{otherwise} \end{cases}.$$

The following standard property relates the accumulated reward of  $M_t$  to solving the reachability objective and is not difficult to show. (The condition  $i(t) = 0$  is technical: in  $M_t$  the history  $t$  does not accumulate any reward.)

**Proposition 3.1.** For any policy  $\pi$  for an MDP  $M = (S, A, T, i)$  with  $t \in S$  and  $i(t) = 0$ , the probability to reach  $t$  in  $M$  under  $\pi$  is the (undiscounted) accumulated reward of  $\pi$  in  $M_t$ , i.e.,  $\mathbb{P}_{\pi, i}^M(\text{Runs}_t) = \mathbb{E}_{\pi, i}^{M_t}(r_{R_t})$ .  $\square$

The finite-horizon reachability objective is reachability within  $k$  steps for  $k \in \mathbb{N}$  being called *horizon*. This objective is induced by the rewards via

$$r_{R_t}^k(s_0, a_0, s_1, a_1 \dots) = \sum_{0 \leq i \leq k-1} R_t(s_i, a_i, s_{i+1}) \quad (2)$$

with  $R_t$  being the reachability reward structure in  $M_t$ .

### 3.2 Parametric MDPs – pMDPs

**Definition 3.6 (Parametric MDP – pMDP<sup>3</sup>).** A parametric Markov Decision Process (*pMDP*) is a tuple  $M = (S, A, X, T, i)$  where  $S$ ,  $A$ , and  $i$  are as in the definition of an MDP and

- $X$  is the parameter space,
- $T: S \times A \rightarrow (\mathcal{D}S)^X$  is the transition function.  $\diamond$

<sup>3</sup> We use the abbreviation pMDP rather than PMDP as it is common in the recent literature, see (e.g. [17, 35]) and as it reminds of the parameter  $p$ ).

For a pMDP  $M$  as above and a parameter point  $x \in X$  we write  $M(x)$  for the *evaluation of  $M$  at  $x$* , that is the MDP  $M(x) = (S, A, T_x, i)$  with  $T_x(s, a) = T(s, a)(x)$ .

We are interested in finding  $\varepsilon$ -optimal policies independent of the parameter, i.e., policies that are somehow  $\varepsilon$ -optimal over the whole parameter space  $X$ . Since for pMDPs the expected return is a function in the parameter, it is not a priori clear what optimality criterion to choose — see [2] for several alternatives. Here we consider the case where a parameter distribution  $p \in \mathcal{D}X$  is given and we optimize the expected reward given  $p$  (this setting is called *expectation optimal* in [2])—as formalized below.

Runs, histories, and policies are defined similarly as for MDPs including in addition the parameter value, and the probability measure now also depends on  $p$ . For a parameter space  $X$ , the sample space is  $\text{Runs}_X = \{(x, \rho) \mid x \in X, \rho \in \text{Runs}(M(x))\}$ ; the set of histories is  $\text{Hist}_X = \{(x, h) \mid x \in X, h \in \text{Hist}(M(x))\}$ ; and the  $\sigma$ -algebra is the smallest  $\sigma$ -algebra  $\text{Cones}_X$  that contains the sets

$$\{\{x\} \times \text{Cone}(h) \mid x \in X, h \in \text{Hist}(M(x))\}.$$

**Definition 3.7 (pMDP policy).** A policy  $\pi_X$  of a pMDP  $M$  is a map  $\pi_X: \text{Hist}_X \rightarrow \mathcal{D}A$  that is independent from the parameters, i.e., that satisfies the property

$$\pi_X(x, h) = \pi_X(y, h) \quad \text{for all } x, y \in X, h \in \text{Hist}(M(x)) \cap \text{Hist}(M(y)). \quad (3)$$

Note that in general  $\text{Hist}(M(x))$  may differ from  $\text{Hist}(M(y))$  for  $x \neq y$ , as different parameter values may make the transition probabilities of certain transitions equal to zero. Furthermore, note that a pMDP policy, due to the requirement (3) can equivalently be defined as a map  $\pi_X: \bigcup_{x \in X} \text{Hist}(M(x)) \rightarrow \mathcal{D}A$ .

A pMDP policy  $\pi_X$ , induces a family of MDP policies  $(\pi_x \mid x \in X)$  with  $\pi_x$  a policy for  $M(x)$  by projection, i.e.,  $\pi_x(h) = \pi_X(x, h)$  for all  $h \in \text{Hist}(M(x))$ .

The measurable space  $(\text{Runs}_X, \text{Cones}_X)$  is (isomorphic to) the disjoint union (coproduct) measurable space

$$\text{Runs}_X = \coprod_{x \in X} \text{Runs}(M(x)), \quad \text{Cones}_X = \left\{ \coprod_{x \in X} A_x \mid A_x \in \text{Cones}(M(x)) \right\}$$

and by  $\mathbb{P}_{\pi_X, i, p}$  we denote the measure that is the  $p$ -convex combination of the measures  $\mathbb{P}_{\pi_x, i}^{M(x)}$ , i.e.,

$$\mathbb{P}_{\pi_X, i, p} \left( \coprod_{x \in X} A_x \right) = \sum_{x \in X} p(x) \cdot \mathbb{P}_{\pi_x, i}^{M(x)}(A_x). \quad (4)$$

*Remark 3.2.* Note that  $\mathbb{P}_{\pi_X, i, p}$  is the unique extension of the assignment

$$\mathbb{P}_{\pi_X, i, p}(\{x\} \times \text{Cone}(h)) = p(x) \cdot \mathbb{P}_{\pi_x, i}^{M(x)}(\text{Cone}(h))$$

to the measurable space  $(\text{Runs}_X, \text{Cones}_X)$ .

We write  $\mathbb{E}_{\pi_X, i, p}$  for the expectation operator of  $\mathbb{P}_{\pi_X, i, p}$ . The *value* of a pMDP  $M$  given parameter distribution  $p$  and objective  $r$  is  $\text{Val}(p, r) = \sup_{\pi_X} \mathbb{E}_{\pi_X, i, p}(r)$  where the supremum is taken over all pMDP policies  $\pi_X$ .

**Definition 3.8 (Expectation  $\varepsilon$ -optimal policy).** *A policy  $\pi_X$  is expectation  $\varepsilon$ -optimal for a pMDP  $M$  iff  $\mathbb{E}_{\pi_X, i, p}(r)$  is  $\varepsilon$ -close to  $\text{Val}(p, r)$ .*  $\diamond$

### 3.3 Partially Observable MDPs – POMDPs

**Definition 3.9 (Partially observable MDP – POMDP).** *A partially observable MDP (POMDP) is a tuple  $M = (S, A, T, i, \Omega, O)$  where:*

- $(S, A, T, i)$  is the underlying MDP,
  - $\Omega$  is the set of observations,
  - $O: S \rightarrow \Omega$  is the observation function.
- $\diamond$

Note that our observation function is deterministic and only state-dependent, which is not a restriction [14]. Runs and histories of a POMDP are the runs and the histories of its underlying MDP. The reward structure for a POMDP is a reward structure of its underlying MDP. The observation function  $O$  extends naturally to runs and histories as follows, by slight abuse of the notation we denote all these functions by  $O$ . We have  $O: \text{Runs} \rightarrow (\Omega \times A)^\omega$  given by

$$O(s_0, a_0, s_1, a_1, \dots) = O(s_0), a_0, O(s_1), a_1, \dots$$

and similarly we define  $O: \text{Hist} \rightarrow (\Omega \times A)^* \times \Omega$ .

The crucial difference to MDPs is that the policy of a POMDP can only observe the observations but not the states directly:

**Definition 3.10 (POMDP policy).** *A policy  $\pi$  for a POMDP  $M$  is a policy for the underlying MDP of  $M$  with the additional requirement that  $\pi(h) = \pi(h')$  whenever  $O(h) = O(h')$ , for all  $h, h' \in \text{Hist}$ .*  $\diamond$

POMDP policies, together with the initial state distribution, also induce a probability measure over runs. The measurable space is again  $(\text{Runs}, \text{Cones})$ . The measure  $\mathbb{P}_{\pi, i}$  is defined in exactly the same way as for the underlying MDP, and  $\mathbb{E}_{\pi, i}$  again denotes the expectation operator. The value of a POMDP  $M$  on an objective  $r$ , also denoted by  $\text{Val}(r)$ , is defined as  $\text{Val}(r) = \sup_{\pi} \mathbb{E}_{\pi, i}(r)$  where the supremum is taken over all POMDP policies  $\pi$ . A POMDP policy  $\pi$  is a *solution* of the POMDP  $M$  for an objective  $r$ , iff  $\mathbb{E}_{\pi, i}(r) = \text{Val}(r)$ ; it is an  $\varepsilon$ -*solution* for  $\varepsilon > 0$  iff  $\mathbb{E}_{\pi, i}(r)$  is  $\varepsilon$ -close to  $\text{Val}(r)$ .

Finite-horizon accumulated reward objectives as defined in (2) are by far the most studied class of POMDP objectives; one might even say that solving such objectives is *the* POMDP problem [10, 34].

*Remark 3.3.* We note two important facts for solutions of POMDPs:

- (1) For POMDPs, deterministic policies are not a restriction (they are as powerful as randomized policies, but can require more memory) for any Borel objective, see [15, Lemma 1, Theorem 7].

- (2) For POMDPs with reachability objectives, for  $\varepsilon$ -approximation with  $\varepsilon > 0$ , finite-memory policies are sufficient for optimality. This is because given any  $\varepsilon > 0$ , there exists a finite horizon  $N_\varepsilon$ , such that reachability within  $N_\varepsilon$  steps  $\varepsilon$ -approximates the optimal reachability probability, and for finite-horizon reachability optimal finite-memory policies are sufficient.

## 4 The Encoding

In this section we reduce the problem of finding an expectation-optimal policy for a pMDP to the problem of solving a POMDP, by presenting an encoding of pMDPs to POMDPs that will relate the policies in the desired way.

The main technical observation of our paper, the observation that enables the method of finding parameter-independent optimal policies for pMDPs via solving the induced POMDP, is the encoding and the correspondence result, Theorem 4.2 below. We start with presenting the encoding.

**Definition 4.1 (Induced POMDP).** *Given a pMDP  $M = (S, A, X, T, i)$ , its induced POMDP is  $M' = (S \times X, A, T', S, O)$  with:*

$$T'((s, x), a)(s', x') = T(s, a)(x)(s') \cdot \delta_x(x')$$

and  $O((s, x)) = s$ . ◇

Hence, the encoding, i.e., the induced POMDP of a pMDP  $M$  is a POMDP with much larger state space: new states are pairs of states of  $M$  (“old” states) and parameter values in  $X$ . Transitions are only possible among new states with the same parameter value, i.e., transitions can not change the parameter values, and the transitions are inherited from the pMDP. Observations are the old states, i.e., in a new state  $(s, x)$  we can observe the old state  $s$  but not the parameter value  $x$ .

Our correspondence result is a consequence of the classical “change of variable” result of measure theory, which we recall next.

**Theorem 4.1 ([24, Theorem VIII.C]).** *Let  $(X, \Sigma)$  and  $(X', \Sigma')$  be measurable spaces,  $f: (X, \Sigma) \rightarrow (X', \Sigma')$  a measurable function,  $\mu: \Sigma \rightarrow \mathbb{R}^+$  a measure, and  $\varphi': X' \rightarrow \mathbb{R}^+ \cup \{\infty\}$  a measurable function.*

*Let  $\mu': X' \rightarrow \mathbb{R}^+$  be the push-forward measure of  $\mu$  along  $f$ , i.e.  $\mu' = \mu \circ f^{-1}$  and let  $\varphi = \varphi' \circ f: X \rightarrow \mathbb{R}^+ \cup \{\infty\}$ .*

*Then*

$$\int_X \varphi d\mu = \int_{X'} \varphi' d\mu',$$

*that is, if one of the integrals exists, the other does too and they are equal. □*

At this point, let us denote by  $\text{Runs}'$ ,  $\text{Hist}'$  and  $\text{Cones}'$  the runs, histories, and the  $\sigma$ -algebra generated by the cones of histories of the encoding  $M'$ . Note that a policy of  $M'$  is a map  $\pi': \text{Hist}' \rightarrow \mathcal{D}A$ . Moreover, note that

$$\text{Runs}' = \{(s_0, x), a_0, (s_1, x), a_1, \dots \mid x \in X \text{ and } s_0, a_0, s_1, a_0, \dots \in \text{Runs}(M(x))\}$$

and analogously for histories. This is a consequence of the construction of the encoding, i.e., of the fact that every run of  $M'$  involves a single parameter point as  $T'((s, x), a)(s', x') = 0$  whenever  $x \neq x'$ .

We now state several direct consequences of the definitions needed for the correspondence result. The proofs of all results are in the extended version [1].

**Lemma 4.1.** *The function  $f : \text{Runs}_X \rightarrow \text{Runs}'$  defined by*

$$f(x, s_0 \cdot a_0 \cdot s_1 \cdots) = (s_0, x) \cdot a_0 \cdot (s_1, x) \cdots$$

*is an isomorphism between the measurable spaces  $(\text{Runs}_X, \text{Cones}_X)$  of the pMDP  $M$  and  $(\text{Runs}', \text{Cones}')$  of its induced POMDP  $M'$ .  $\square$*

This means that  $f$  is a bijection and both  $f$  and  $f^{-1}$  are measurable functions. In the sequel, we also write  $f$  for the bijection from  $\text{Hist}_X$  to  $\text{Hist}'$  defined in the same way. Note that  $f$  maps generators of  $\text{Cones}_X$  to generators of  $\text{Cones}'$  as

$$f(\{x\} \times \text{Cone}(h_x)) = \text{Cone}(f(x, h_x))$$

for any  $x \in X$  and  $h_x \in \text{Hist}(M(x))$ . Moreover, to state the obvious,  $f$  is related to the observation map  $O$  as follows:  $O(f(x, h)) = h$ .

**Lemma 4.2.** *There is a bijective correspondence  $\Phi$  between the policies of the pMDP  $M$  and the policies of its induced POMDP  $M'$  given by  $\Phi(\pi_X) = \pi_X \circ f^{-1}$ . Its inverse acts as  $\Phi^{-1}(\pi') = \pi' \circ f$ .  $\square$*

We are now able to relate the induced measures in a pMDP and its encoding.

**Lemma 4.3.** *Given a pMDP  $M = (S, A, X, T, i)$ , parameter distribution  $p$ , and policy  $\pi_X$ , we have*

$$\mathbb{P}_{\pi_X, i, p}^M = \mathbb{P}_{\pi', i'}^{M'} \circ f$$

*where  $M'$  is the induced POMDP of  $M$ ,  $\pi' = \Phi(\pi_X)$ , and  $i'(s, x) = i(s) \cdot p(x)$ .  $\square$*

Now the correctness of the encoding follows easily from the next result.

**Theorem 4.2.** *Given a pMDP  $M = (S, A, X, T, i)$ , parameter distribution  $p \in \mathcal{D}X$ , policy  $\pi_X$ , and an objective function  $r$ , we have*

$$\mathbb{E}_{\pi_X, i, p}^M(r) = \mathbb{E}_{\pi', i'}^{M'}(r') \tag{5}$$

*where  $M'$  is the induced POMDP of  $M$ ,  $\pi' = \Phi(\pi_X)$ ,  $r' = r \circ f^{-1}$ , and  $i'(s, x) = i(s) \cdot p(x)$ . The opposite also holds: Given a policy  $\pi'$  of the induced POMDP  $M'$  of  $M$ , Eq. (5) holds for  $\pi_X = \Phi^{-1}(\pi')$ .  $\square$*

As a consequence,  $\text{Val}^M(p, r) = \text{Val}^{M'}(r')$  and the policy  $\pi_X$  is expectation  $\varepsilon$ -optimal for  $M$  if and only if  $\pi' = \Phi(\pi_X)$  is an  $\varepsilon$ -solution of  $M'$ , for  $\varepsilon \geq 0$ .

## 5 Experiments<sup>4</sup>

In this section, we present several case studies to illustrate our approach of finding an expectation  $\varepsilon$ -optimal policy of a pMDP using an existing POMDP solver. The solver we use is AI-Toolbox [7], a well-known suite of algorithms for Markov models, which includes several algorithms for POMDPs with finite-horizon accumulated-reward objectives. For our case studies we evaluate two algorithms: *point-based value iteration* (PBVI) [34] and *Incremental Pruning* (IP) [10], for reasons explained in the extended version [1].

The PBVI implementation in AI-Toolbox does not generate beliefs on the fly (as in the anytime algorithm described in [34]), but generates a fixed number of beliefs upfront. First the simplex corners (i.e. the Diracs) and the midpoint are generated, then more are sampled (the point-based algorithms differ mainly in how the beliefs are sampled). Without at least the simplex corners and midpoint, the results are quite off - we chose to always pick at least all of those.

*Experimental Setup.* All the models that we analyze are described in PRISM file format. We use Storm [20] to parse the files and build the parametric model, that we then translate and pass to AI-Toolbox. Both Storm and AI-Toolbox have Python bindings allowing us to perform our encoding in Python. The experiments reported here ran on a NUMA machine with four 16-core 2.3 GHz AMD Opteron 6376 processors, 504 GB of main memory, and Linux kernel version 4.13.0. We used BenchExec [9] to run experiment series.

*Selected Case Studies.* We start by discussing a motivating *learner model* example that shows all important aspects. Then we present some results on a model of a robot moving on a grid and a consensus protocol model.

It is important to note here that many of the pMDP models studied (e.g., in the PARAM benchmarks [23]) exhibit only a weak form of nondeterminism, where the optimal policy does not depend on the parameters, i.e., the optimal policy is the same for any values of the parameters. Examples of such are, e.g., the Bounded Retransmission Protocol (BRP) and Zeroconf. Consensus is the exception, and we have solved some instances of Consensus as reported below. In the examples used in [35], showing how to obtain policies that optimize learning the parameter values, the optimal policy is again independent of the parameters.

**Motivating Example – The (Repeated) Learner Model.** Figure 1a shows the *learner model*, mentioned in [2], with initial state  $s$  and target state  $t$ , ignoring for the moment the grey (loop) transition labelled with action  $c$ .

After two steps we end up in state  $c$ , having visited state  $a$  with probability  $p$  and state  $b$  with probability  $1 - p$ . Here is the only choice in the model: between the actions  $a$  and  $b$ . Even though we assume  $p$  to be inaccessible, we can do better than flipping a coin: If we are given a prior belief over the parameter (an initial parameter distribution) we can use Bayesian inference to update this belief

<sup>4</sup> All of our code and models, as well as detailed results of the experiments can be found at <http://github.com/sarming/pMDP-Toolbox>.



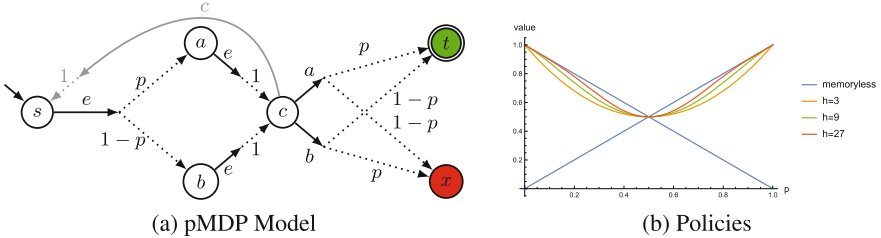


Fig. 1. (Repeated) learner

with the information that either state  $a$  or  $b$  was visited. As a concrete example, assume we start with the uniform distribution over the parameter values as prior and state  $a$  was visited. Doing the calculation gives a posterior distribution with higher probability that the parameter value is close to 1 than to 0, suggesting that  $a$  is the better action to choose.

Let  $\pi$  be the policy that chooses action  $a$  in state  $c$  when state  $a$  was visited and action  $b$  when state  $b$  was visited. Figure 1b shows  $\pi$  (labelled  $h = 3$ ) and the two memoryless policies (always  $a$  and always  $b$ ). The policy  $\pi$  is clearly better in expectation than the two memoryless ones.

Actually,  $\pi$  is the optimal policy (among the 4 possible deterministic policies) when assuming a uniform parameter distribution. For a concrete parameter value  $x \in [0, 1]$ , the probability to reach  $t$  under  $\pi$  in the evaluated MDP  $M(x)$  is  $x^2 + (1 - x)^2$ . Putting things together, using Eq. (4), we get that for a uniform parameter distribution the expected return is (a discrete approximation of)  $\int_0^1 x^2 + (1 - x)^2 dx = 2/3$ .

We consider the uniform distributions over 2, 3, 5, 10, 20, 50, 100, 200, 500, and 1000 evenly spaced points between 0 and 1. The expected return depends on the distribution: for just 2 points (the distribution assigning  $1/2$  to 0 and  $1/2$  to 1) the optimal policy provides probability 1 for reaching  $t$ ; while for 1000 points the expected return is 0.6670. IP can solve all mentioned instances and generates  $\pi$  verifying that it is optimal.

The learner is inherently a finite-horizon model, as nothing happens after three steps. When we add the grey transition, we obtain the *repeated learner model* in which we can repeat the “experiment” getting closer and closer to the “sea surface” [2] given by the memoryless policies. The larger the horizon, the more experiments the  $\varepsilon$ -optimal policy runs. Only an odd number of experiments gives an actual improvement because we need a majority: having two experiments is as good as just having one. Therefore the value increases at  $h = 3(2n + 1)$ , see Fig. 1b and Table 1b.

Figure 2 shows the results of the experimental evaluation of the repeated learner with increasing horizon and up to 100 parameter points. IP provides better policies than PBVI with respect to the same number of points in the parameter space. However, it does not scale as well as PBVI and we had to drop it for more complex models.

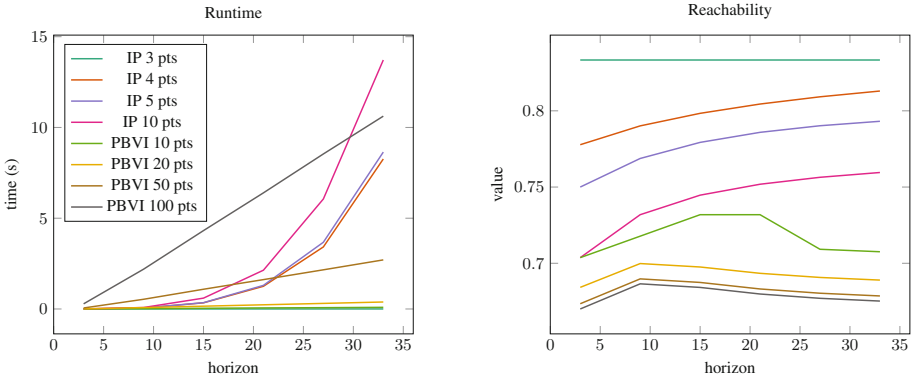
**Table 1.** IP results for (repeated) learner

points	states	time (s)	value	nodes
2	14	0.001	1	3
5	35	0.003	0.75	7
10	70	0.005	0.703	7
20	140	0.008	0.684	7
50	350	0.025	0.673	7
100	700	0.098	0.670	7
200	1400	0.423	0.668	7
500	3500	2.496	0.667	7
1000	7000	9.725	0.667	7

(a)  $h = 3$  (i.e. without repeating)

$h$	time (s)	value	nodes
3	0.003	0.704	7
9	0.081	0.732	23
15	0.598	0.745	47
21	2.137	0.752	79
27	6.059	0.756	119
33	13.715	0.760	167
39	30.324	0.762	223

(b) 10 points



**Fig. 2.** IP and PBVI results for repeated learner

**Robot on a Grid.** Our next case study is a variant of the ever popular Grid-world [37]. The instance that we report on is a  $3 \times 3$  grid with initial state at position (1, 1), sink at position (2, 2), and target at position (3, 3). We want to maximize the probability to reach the target from the initial position.

The robot has (up to) 4 actions available: up, down, left, right. The actions are probabilistic in the sense that with some probability, instead of going forward the robot may end up in the cell to the left or to the right. We compare two variants: In both variants there is a parameter  $p$  that describes the total error probability. In the 1-parameter variant, the probability to err left and right is equal to  $p/2$ ; In the 2-parameter variant we also include a left-right bias  $b$ , resulting in probability  $p * b$  to err left and  $p * (1 - b)$  to err right. If it is not possible to go left or right, then the other option gets probability  $p$ . For example, the action up in cell (1, 1) leads correctly to cell (2, 1) with probability  $1 - p$  and to cell (1, 2) on the right with probability  $p$  (as no cell is on the left), in both models. The action up in cell (1, 2) has the possibility to err left and right, hence

shows the difference between the two models. See [2] for a detailed description of robot-on-a-grid models. In that terminology, we use the “fixed failure” variant.

**Consensus.** The consensus protocol model is the only PARAM benchmark [23] that has true nondeterminism in the sense that its policy depends on the parameter values. The protocol was introduced by Aspnes and Herlihy [3]. The 2-parameter model is exactly the same as the PARAM model, see [23] for all details. The 1-parameter model depends on a parameter  $p$  and is obtained by setting  $p_1 = p$  and  $p_2 = 1 - p$ , i.e., it is a bias parameter with average  $1/2$ . We used  $N = K = 2$  and the target state is the state in which consensus is reached with the preferred value.

Figures 3 and 4 show the experimental results of the robot on a grid, and the consensus protocol benchmarks, respectively. We only ran PBVI as IP was too slow and in both cases notice that the runtime grows exponentially with the horizon. The grid pMDP model has 10 states and 2 parameters, hence the

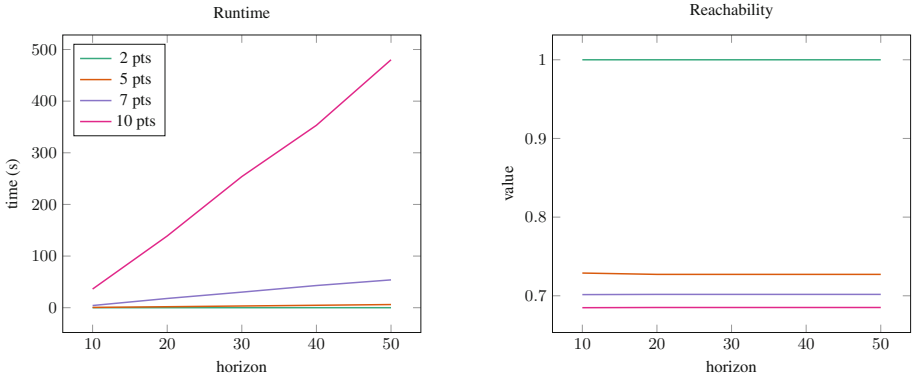


Fig. 3. PBVI results for robot on a grid

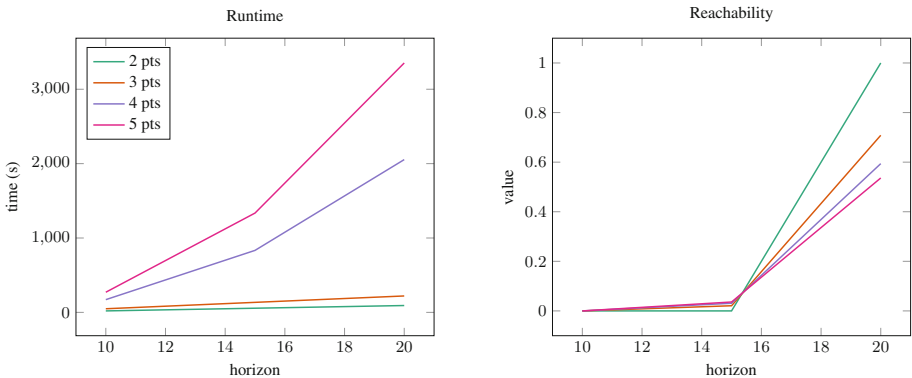


Fig. 4. PBVI results for consensus

induced POMDP for 10 parameter points has 1000 states. The consensus protocol model has 273 states and hence the induced POMDP for 5 parameter points has 1365 states.

## 6 Conclusion

We have presented a way to compute parameter-independent strategies that are expectation  $\varepsilon$ -optimal for pMDP by encoding the problem as to compute  $\varepsilon$ -solutions in POMDPs. We have implemented this approach using Storm [20] and AI-Toolbox [7] and we have evaluated on different case studies. Future work will focus on improving the efficiency of the current algorithms (for better scalability) by taking into account the particular POMDP structure resulting from the encoding.

## References

1. Arming, S., Bartocci, E., Chatterjee, K., Katoen, J., Sokolova, A.: Parameter-independent strategies for pMDPs via POMDPs. arXiv 1806.05126 (2018). <http://arxiv.org/abs/1806.05126>
2. Arming, S., Bartocci, E., Sokolova, A.: SEA-PARAM: exploring schedulers in parametric MDPs. In: Proceedings of the QAPL 2017. EPTCS, vol. 250, pp. 25–38 (2017)
3. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. *J. Algorithms* **11**(3), 441–461 (1990)
4. Baier, C., Größer, M., Bertrand, N.: Probabilistic  $\omega$ -automata. *J. ACM* **59**(1), 1:1–1:52 (2012)
5. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. Baldi, M., et al.: A probabilistic small model theorem to assess confidentiality of dispersed cloud storage. In: Bertrand, N., Bortolussi, L. (eds.) QEST 2017. LNCS, vol. 10503, pp. 123–139. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66335-7\\_8](https://doi.org/10.1007/978-3-319-66335-7_8)
7. Bargiacchi, E.: AI-Toolbox. <https://github.com/Svalorzen/AI-Toolbox/>
8. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 326–340. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-19835-9\\_30](https://doi.org/10.1007/978-3-642-19835-9_30)
9. Beyer, D., Löwe, S., Wendler, P.: Benchmarking and resource measurement. In: Fischer, B., Geldenhuys, J. (eds.) SPIN 2015. LNCS, vol. 9232, pp. 160–178. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-23404-5\\_12](https://doi.org/10.1007/978-3-319-23404-5_12)
10. Cassandra, A.R., Littman, M.L., Zhang, N.L.: Incremental pruning - a simple, fast, exact method for partially observable Markov decision processes. In: Proceedings of the UAI 1997, pp. 54–61 (1997)
11. Chatterjee, K., Doyen, L., Henzinger, T.A.: Qualitative analysis of partially-observable Markov decision processes. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 258–269. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15155-2\\_24](https://doi.org/10.1007/978-3-642-15155-2_24)
12. Chatterjee, K., Chmelik, M.: POMDPs under probabilistic semantics. *Artif. Intell.* **221**, 46–72 (2015)

13. Chatterjee, K., Chmelik, M., Davies, J.: A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In: Proceedings of the AAAI 2016, pp. 3225–3232 (2016)
14. Chatterjee, K., Chmelik, M., Gupta, R., Kanodia, A.: Optimal cost almost-sure reachability in POMDPs. *Artif. Intell.* **234**, 26–48 (2016)
15. Chatterjee, K., Doyen, L., Gimbert, H., Henzinger, T.A.: Randomness for free. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 246–257. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-15155-2\\_23](https://doi.org/10.1007/978-3-642-15155-2_23)
16. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M.Z., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: Proceedings of the TASE 2013, pp. 85–92 (2013)
17. Cubuktepe, M.: Sequential convex programming for the efficient verification of parametric MDPs. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 133–150. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54580-5\\_8](https://doi.org/10.1007/978-3-662-54580-5_8)
18. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 280–294. Springer, Heidelberg (2005). [https://doi.org/10.1007/978-3-540-31862-0\\_21](https://doi.org/10.1007/978-3-540-31862-0_21)
19. Dehnert, C., et al.: PROPhESY: a probabilistic parameter synthesis tool. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 214–231. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-21690-4\\_13](https://doi.org/10.1007/978-3-319-21690-4_13)
20. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A STORM is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_31](https://doi.org/10.1007/978-3-319-63390-9_31)
21. Hahn, E.M., Han, T., Zhang, L.: Probabilistic reachability for parametric Markov models. *STTT* **13**(1), 3–19 (2011)
22. Hahn, E.M., Han, T., Zhang, L.: Synthesis for PCTL in parametric Markov decision processes. In: Bobaru, M., Havelund, K., Holzmann, G.J., Joshi, R. (eds.) NFM 2011. LNCS, vol. 6617, pp. 146–161. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-20398-5\\_12](https://doi.org/10.1007/978-3-642-20398-5_12)
23. Hahn, E.M., Hermanns, H., Zhang, L., Wachter, B.: PARAM case studies (2015). <https://depend.cs.uni-saarland.de/tools/param/casestudies>
24. Halmos, P.R.: *Measure Theory*. Springer, New York (1974). <https://doi.org/10.1007/978-1-4684-9440-2>
25. Jansen, N., et al.: Accelerating parametric probabilistic verification. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 404–420. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10696-0\\_31](https://doi.org/10.1007/978-3-319-10696-0_31)
26. Junges, S., Jansen, N., Wimmer, R., Quatmann, T., Winterer, L., Katoen, J., Becker, B.: Finite-state controllers of POMDPs via parameter synthesis. In: Proceedings of the UAI 2018 (2018)
27. Kreinovich, V., Lakeyev, A., Rohn, J., Kahl, P.: *Computational Complexity and Feasibility of Data Processing and Interval Computations, Applied Optimization*, vol. 10. Springer, Boston (1998). <https://doi.org/10.1007/978-1-4757-2793-7>
28. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
29. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Parametric probabilistic transition systems for system design and analysis. *Form. Asp. Comput.* **19**(1), 93–109 (2007)

30. Lukina, A., et al.: ARES: adaptive receding-horizon synthesis of optimal plans. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 286–302. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54580-5\\_17](https://doi.org/10.1007/978-3-662-54580-5_17)
31. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. *Artif. Intell.* **147**(1–2), 5–34 (2003)
32. Medina Ayala, A.I., Andersson, S.B., Belta, C.: Probabilistic control from time-bounded temporal logic specifications in dynamic environments. In: Proceedings of the ICRA 2012, pp. 4705–4710. IEEE (2012)
33. Pathak, S., Ábrahám, E., Jansen, N., Tacchella, A., Katoen, J.-P.: A greedy approach for the efficient repair of stochastic models. In: Havelund, K., Holzmann, G., Joshi, R. (eds.) NFM 2015. LNCS, vol. 9058, pp. 295–309. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-17524-9\\_21](https://doi.org/10.1007/978-3-319-17524-9_21)
34. Pineau, J., Gordon, G.J., Thrun, S.: Point-based value iteration - an anytime algorithm for POMDPs. In: Proceedings of the IJCAI 2003, pp. 1025–1032 (2003)
35. Polgreen, E., Wijesuriya, V.B., Haesaert, S., Abate, A.: Automated experiment design for data-efficient verification of parametric Markov decision processes. In: Bertrand, N., Bortolussi, L. (eds.) QEST 2017. LNCS, vol. 10503, pp. 259–274. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66335-7\\_16](https://doi.org/10.1007/978-3-319-66335-7_16)
36. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.-P.: Parameter synthesis for Markov models: faster than ever. In: Artho, C., Legay, A., Peled, D. (eds.) ATVA 2016. LNCS, vol. 9938, pp. 50–67. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-46520-3\\_4](https://doi.org/10.1007/978-3-319-46520-3_4)
37. Russell, S., Norvig, P.: *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River (2009)
38. Sennott, L.I.: *Stochastic Dynamic Programming and the Control of Queueing Systems*. Wiley, New York (1998)
39. Spaan, M.T.J., Vlassis, N.: Perseus: randomized point-based value iteration for POMDPs. *J. Artif. Intell. Res.* **24**, 195–220 (2011)



# On the Verification of Weighted Kripke Structures Under Uncertainty

Giovanni Bacci<sup>(✉)</sup>, Mikkel Hansen, and Kim Guldstrand Larsen

Department of Computer Science, Aalborg University, Aalborg, Denmark  
{giovbacci,mhan,kg1}@cs.aau.dk

**Abstract.** We study the problem of checking *weighted CTL* properties for weighted Kripke structures in presence of imprecise weights. We consider two extensions of the notion of weighted Kripke structures, namely (i) *parametric weighted Kripke structures*, having transitions weights modelled as affine maps over a set of parameters and, (ii) *weight-uncertain Kripke structures*, having transition labelled by real-valued random variables as opposed to precise real valued weights.

We address this problem by using *extended parametric dependency graphs*, a symbolic extension of dependency graphs by Liu and Smolka. Experiments performed with a prototype tool implementation show that our approach outperforms by orders of magnitude an adaptation of a state-of-the-art tool for WKSs.

## 1 Introduction

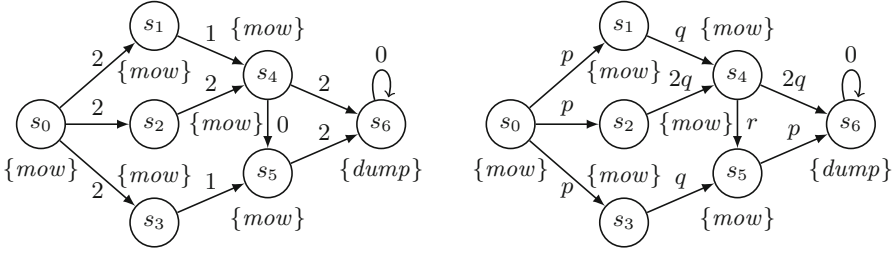
The rapid diffusion of cyber-physical systems (CPSs) poses the challenge of handling their growing complexity, while meeting requirements on correctness, predictability, performance without compromising time- and cost-to-market. In this respect, model-driven development is a promising approach that allows for early design and verification and may be used as the basis for systematic testing of a final product. The verification of cyber-physical systems should not only address functional properties but also a number of non-functional properties related to the quantitative aspects that are typical of such systems.

In the area of model checking, a number of modelling formalisms have emerged, allowing for quantitative aspects to be expressed. Among these, Weighted Kripke structures (WKSs) were proposed as a natural extension of the usual notion of Kripke structures with a (real-valued) weighted transition relation [8].

Interesting properties of WKSs may be expressed by means of quantitative extensions of CTL. There are different ways of extending CTL with quantitative information. Fahrenberg et al. [8] proposed to generalise the classical Boolean interpretation of CTL to a map that assigns to states and temporal formulas a real-valued distance describing the degree of satisfaction. This paper considers weighted CTL (WCTL), an extension of CTL with weight-constrained modalities, because it is an expressive logic with efficient tool support for WKSs [9].

---

Work supported by the Advanced ERC Grant nr. 867096 (LASSO).



**Fig. 1.** (Left) A lawn mower example from [9]; (Right) the lawn mower model with weights parametric in  $p$ ,  $q$  and  $r$ .

Consider the WKS in Fig.1(left) representing a grass field with different routes a lawn mower can take from the starting state  $s_0$  to  $s_6$  where the grass can be dumped. The weights on the transitions represent the amount of grass that is accumulated in the container when selecting a particular route. Assume that the lawn mower breaks when it is forced to store more than 6.5 units of grass, then the property “the grass is always dumped before the lawn mower breaks, irrelevant of the selected route” is expressed in WCTL as  $\forall(mow \mathcal{U}_{\leq 6.5} dump)$ .

The above example models the accumulated grass by means of precise weight values. This is an unrealistic simplification, since the amount of mowed grass may vary depending on different factors (e.g., distribution of the grass in the field, meteorologic conditions, etc.) that cannot be modelled with precise values. The same argument applies to CPSs, that typically rely on sensor measurements which are inherently imprecise.

Typically, there are two ways for dealing with uncertain sensor measurements: (i) determine the precision of the instrument and associate an error  $\epsilon$  with each measurement, or (ii) perform estimation statistics (e.g., by recursive Bayesian estimation [14]) and associate a probability distribution with each measurement.

In this paper we aim at providing adequate formal basis and tool-support for the verification of WKSs in presence of imprecise weights. We consider two extensions of the notion of WKS: (i) *parametric weighted Kripke structures* (pWKSs), having weights depending on a set of parameters (*cf.* Fig.1(right)) and, (ii) *weight-uncertain Kripke structures* (WUKSs), having as weights real-valued random variables as opposed to precise real values. On the one hand, verification of pWKSs is done by inferring constraints over its parameters characterising the valuations that ensure correctness then, verify the robustness of the model within the given precision. On the other hand, verifying WUKSs consists of measuring the degree of satisfaction of the model w.r.t. the given specification.

Our contribution is twofold. First, we extend and improve the model checking algorithm of [5] for pWKSs. In contrast with [5], our method supports negation and implements an efficient termination condition. In line with [5,6,9], our algorithm uses an extension of dependency graphs by Liu and Smolka [11] to model-check pWKSs. Specifically, we integrate cover-edges from [9] and negation-edges from [6] and, lift the computation of fixed points from the boolean domain to



that of non-negative real-valued maps to cope with parametric weights and the non-monotonic reasoning necessary to deal with negation.

As for our second contribution, we introduce the notion of weight-uncertain Kripke structures and address two natural problems related to their analysis: (i) checking whether the expected behaviour of the model satisfies a given specification and, (ii) measuring the probability that a concrete realisation of the model satisfies a given WCTL formula.

The proposed model checking framework has been implemented on a prototype tool. Experiments show that our approach considerably improves w.r.t. the PVT00L from [5] and outperforms an adaptation of the WKTOOL from [9].

We refer to the full version of this paper [2] for the omitted proofs.

*Related Work.* Our paper fits within the area of weighted automata [7] where weights come as elements of a semi-ring. By combining the tropical and the probability semi-rings, one obtains probabilistic weighted automata (PWA) [1, 4]. There, transitions are labelled with a cost and a probability and the weight that the PWA assigns to a word is the expected accumulated costs of the runs producing the word. A similar approach is seen with Markov reward models whose analysis consider the computation of the expected reward for reachability properties or their verification against probabilistic reward CTL [3]. In contrast to PWAs and Markov reward models, where transitions are executed probabilistically and the weights are fixed, WUKSs choose transitions non-deterministically and generate weights according to the given probability distributions.

Fahrenberg et al. [8] consider the verification of WKSs with respect to two interpretations of WCTL where the satisfaction of a formula by a model is no longer interpreted in the Boolean domain, but rather assigns to a state a truth value in the domain of extended non-negative reals where a smaller value means a better match of the specified weights in the formula. Differently from [8], we keep the classical boolean interpretation of WCTL and measure how likely is the model to be correct. In this respect, our approach resembles that of probabilistic LTL model checking for Markov chains [3, 15].

## 2 Weighted Kripke Structures and Weighted CTL

In this section we present *weighted Kripke structures* (WKSs) as an expressive modelling formalism for quantitative systems, and *weighted CTL* (WCTL), an extension of computation tree logic (CTL) with weight-constrained modalities, interpreted with respect to WKSs.

We denote by  $\mathbb{R}$ ,  $\mathbb{Q}$ , and  $\mathbb{N}$  respectively the sets of real numbers, rational numbers, and natural numbers. We write  $\mathbb{R}_{\geq 0}$  (resp.  $\mathbb{Q}_{\geq 0}$ ) to denote the set of non-negative real (resp. rational) numbers.

**Definition 1 (WKS).** *A weighted Kripke structure is a tuple  $\mathcal{K} = (S, R, \ell)$ , where  $S$  is a finite nonempty set of states,  $R \subseteq S \times \mathbb{R}_{\geq 0} \times S$  is a finite weighted transition relation and  $\ell: S \rightarrow 2^{\mathcal{AP}}$  is a function labelling the states with subsets of  $\mathcal{AP}$ , where  $\mathcal{AP}$  is a fixed set of atomic propositions.*

Let  $\mathcal{K} = (S, R, \ell)$  be a WKS. We write  $s \xrightarrow{w} s'$  to indicate that  $(s, w, s') \in R$  and, we denote with  $\omega(\mathcal{K}) \in \mathbb{R}_{\geq 0}^m$  the vector of weighs of  $\mathcal{K}$ , where  $m = |R|$ . A run in  $\mathcal{K}$  from  $s_0 \in S$  is a (finite or infinite) sequence  $\pi = (w_i, s_i)_{i \in I}$ , such that  $w_0 = 0$  and  $I$  is an interval of  $\mathbb{N}$  containing 0 where, for all  $i \in I \setminus \{0\}$ ,  $s_{i-1} \xrightarrow{w_i} s_i$ . The *accumulated weight* of a run  $\pi = (w_i, s_i)_{i \in I}$  at position  $j \in I$  is defined as  $\mathcal{W}(\pi, j) = \sum_{i=0}^j w_i$ .

We write  $|\pi|$  for the length of  $\pi$  (the cardinal of  $I$ ); and, for  $i \in I$ , we write  $\pi[i]$  for the  $i$ -th state in  $\pi$ , i.e.,  $\pi[i] = s_i$ . A run is *maximal* if it has infinite length ( $|\pi| = \omega$ ) or its last state has no outgoing transitions.  $Run(\mathcal{K}, s_0)$  denotes the set of all maximal runs from  $s_0$  in  $\mathcal{K}$ .

We can now define WCTL with upper-bounds on weights. WCTL allows for *state formulas* describing properties about states in the system and *path formulas* describing properties about runs in a WKS. State formulas  $\Phi, \Psi$  and path formulae  $\varphi$  are constructed over the following abstract syntax

$$\Phi, \Psi ::= t \mid a \mid \neg\Phi \mid \Phi \wedge \Psi \mid \exists\varphi \mid \forall\varphi. \quad \varphi ::= \mathcal{X}_{\leq q}\Phi \mid \Phi \mathcal{U}_{\leq q}\Psi$$

where  $a \in \mathcal{AP}$  and  $q \in \mathbb{Q}_{\geq 0}$ .

Given a WKS  $\mathcal{K} = (S, R, \ell)$ , a state  $s \in S$ , and a run  $\pi \in Run(\mathcal{K}, s)$ , we denote by  $\mathcal{K}, s \models \Phi$  (resp.  $\mathcal{K}, \pi \models \varphi$ ) the fact that the state  $s$  satisfies the state formula  $\Phi$  (resp. the path  $\pi$  satisfies the path formula  $\varphi$ ). Formally, the *satisfiability relation*  $\models$  is inductively defined as:

$$\begin{array}{ll} \mathcal{K}, s \models t & \text{always holds} \\ \mathcal{K}, s \models a & \text{if } p \in \ell(s) \\ \mathcal{K}, s \models \neg\Phi & \text{if } \mathcal{K}, s \not\models \Phi \\ \mathcal{K}, s \models \Phi \wedge \Psi & \text{if } \mathcal{K}, s \models \Phi \text{ and } \mathcal{K}, s \models \Psi \\ \mathcal{K}, s \models \exists\varphi & \text{if there exists } \pi \in Run(\mathcal{K}, s) \text{ such that } \mathcal{K}, \pi \models \varphi \\ \mathcal{K}, s \models \forall\varphi & \text{if for all } \pi \in Run(\mathcal{K}, s) \text{ it holds that } \mathcal{K}, \pi \models \varphi \\ \mathcal{K}, \pi \models \mathcal{X}_{\leq q}\Phi & \text{if } |\pi| > 0, \mathcal{W}(\pi, 1) \leq q, \text{ and } \mathcal{K}, \pi[1] \models \Phi \\ \mathcal{K}, \pi \models \Phi \mathcal{U}_{\leq q}\Psi & \text{if there exists } j \leq |\pi| \text{ such that } \mathcal{K}, \pi[j] \models \Psi, \\ & \mathcal{W}(\pi, j) \leq q, \text{ and } \mathcal{K}, \pi[j'] \models \Phi \text{ for all } j' < j \end{array}$$

As usual, we can derive the logical operators  $\text{ff}$ ,  $\vee$  and  $\rightarrow$  as follows:  $\text{ff} \stackrel{\text{def}}{=} \neg t$ ,  $\Phi \vee \Psi \stackrel{\text{def}}{=} \neg(\neg\Phi \wedge \neg\Psi)$  and,  $\Phi \rightarrow \Psi \stackrel{\text{def}}{=} \neg\Phi \vee \Psi$ .

*Example 2.* Consider the WKS  $\mathcal{K}$  in Fig. 1(left) described before. The WCTL state formulas  $\Phi = \forall(mow \mathcal{U}_{\leq 6} dump)$  and  $\Phi' = \exists(mow \mathcal{U}_{\leq 4} dump)$  express respectively the properties “the grass is always dumped before the lawn accumulates more that 6 grass units, irrelevant of the selected route” and “there exists a mowing route that accumulates at most 4 grass units before dumping”. Clearly  $\mathcal{K}, s_0 \models \Phi$  holds true because all paths from  $s_0$  to  $s_6$  accumulate at most 6 grass units, whereas  $\mathcal{K}, s_0 \models \Phi'$  doesn't hold true, because each path from  $s_0$  to  $s_6$  accumulates at least 5 grass units.  $\square$

### 3 Parametric Weighted Kripke Structures

In this section we present *parametric weighted Kripke structures* and demonstrate how they can be employed for verifying the robustness of WKSs in presence of imprecise weights.

Parametric weighted Kripke structures (pWKSs) model families of WKSs that rely on the same graph structure, but differ in the concrete transition weights, which are specified as expressions built over a set of parameters.

Let  $\mathbf{x} = (x_1, \dots, x_k)$  be a vector of real-valued parameters. We denote by  $\mathcal{E}$  the set of affine maps  $f: \mathbb{R}^k \rightarrow \mathbb{R}$  of the form  $f(\mathbf{x}) = \mathbf{a} \cdot \mathbf{x} + b$ , with  $\mathbf{a} = (a_1, \dots, a_k) \in \mathbb{Q}_{\geq 0}^k$  and  $b \in \mathbb{Q}_{\geq 0}$ , i.e.,  $f(x_1, \dots, x_k) = (\sum_{i=1}^k a_i x_i) + b$ . Hereafter we may denote the map  $f$  by means of the augmented vector<sup>1</sup>  $(\mathbf{a}, b) \in \mathbb{N}^{k+1}$ . Accordingly, for  $f, g \in \mathcal{E}$  the map addition  $(f + g)(\mathbf{x}) = f(\mathbf{x}) + g(\mathbf{x})$  is encoded as the vector addition.

**Definition 3.** A parametric weighted Kripke structure is a tuple  $\mathcal{P} = (S, R, \ell)$ , where  $S$  is a finite nonempty set of states,  $R \subseteq S \times \mathcal{E} \times S$  is a finite parametric weighted transition relation and  $\ell: S \rightarrow 2^{\mathcal{A}^{\mathcal{P}}}$  is a labelling function.

Intuitively, a pWKS  $\mathcal{P} = (S, R, \ell)$  defines a family of WKSs arising by plugging in concrete values for the parameters. A parameter valuation  $\mathbf{v} \in \mathbb{R}^k$  is said to be *admissible* for  $\mathcal{P}$  if for each transition  $(s, f, s') \in R$  we have  $f(\mathbf{v}) \geq 0$ . Let  $\mathcal{V}_{\mathcal{P}}$ , or just  $\mathcal{V}$  when  $\mathcal{P}$  is clear from the context, denote the set of admissible valuations for  $\mathcal{P}$ . Given  $\mathbf{v} \in \mathcal{V}$ , we denote  $\mathcal{P}(\mathbf{v})$  the WKS associated with  $\mathbf{v}$ . In this respect, it will be convenient to think at  $\mathcal{P}$  as a partial function  $\mathcal{P}: \mathbb{R}^k \rightarrow \text{WKS}$  with domain  $\mathcal{V}_{\mathcal{P}}$ . The semantics of  $\mathcal{K}$ , written  $[\mathcal{P}]$ , is defined as the image of  $\mathcal{P}$ , i.e.,  $[\mathcal{P}] = \{\mathcal{P}(\mathbf{v}) \mid \mathbf{v} \in \mathcal{V}\}$ .

A task typically addressed in the analysis of parametric Kripke structures is that of finding symbolic representations of the set of parameter valuations for which a given WCTL formula holds [5].

Formally, given a pWKS  $\mathcal{P} = (S, R, \ell)$ , a state  $s \in S$  and a state formula  $\Phi$ , the set of admissible valuations for which  $\Phi$  holds at  $s$  is

$$\llbracket \mathcal{P}, s \models \Phi \rrbracket \stackrel{\text{def}}{=} \{\mathbf{v} \in \mathcal{V} \mid \mathcal{P}(\mathbf{v}), s \models \Phi\}. \quad (1)$$

*Example 4.* Consider the pWKS  $\mathcal{P}$  depicted in Fig. 1(right) representing a family of lawn mower models parametric in  $p$ ,  $q$  and,  $r$ . Its parameters represent the amount of grass measured in different parts of the field. The admissible valuations for  $\mathcal{P}$ , i.e.,  $\mathcal{V}_{\mathcal{P}}$ , are represented by the constraint

$$\alpha(p, q, r) = p \geq 0 \wedge q \geq 0 \wedge r \geq 0. \quad (2)$$

Let  $\Phi = \forall(\text{mow} \mathcal{U}_{\leq 6.5} \text{dump})$  be our specification. The set of valuations satisfying  $\Phi$ , i.e.,  $\llbracket \mathcal{P}, s \models \Phi \rrbracket$ , is represented by the following constraint

$$\beta(p, q, r) = \alpha(p, q, r) \wedge p + 4q \leq 6.5 \wedge 2p + 2q + r \leq 6.5. \quad (3)$$

<sup>1</sup> Our is a special case of the so called *affine transformation matrix* (or *projective transformation matrix*) representation for generic affine transformations.

Assume that we have measured  $p \cong 2 \pm \epsilon$ ,  $q \cong 1 \pm \epsilon$  and,  $r \cong 0 \pm \epsilon$  where  $\epsilon > 0$  is the measurement error. One can determine if  $\mathcal{P}$  is robust w.r.t.  $\Phi$  by checking that all possible measurement values lay in  $\llbracket \mathcal{P}, s \models \Phi \rrbracket$ , formally

$$\mathcal{V}_{\mathcal{P}} \cap \{(p, q, r) \mid |p - 2| \leq \epsilon, |q - 1| \leq \epsilon, |r| \leq \epsilon\} \subseteq \llbracket \mathcal{P}, s \models \Phi \rrbracket.$$

The above can be expressed as first-order formula in theory of linear real arithmetic

$$\forall p \in [0, 2 + \epsilon]. \forall q \in [0, 1 + \epsilon]. \forall r \in [0, \epsilon]. \beta(p, q, r). \quad (4)$$

By performing quantifier elimination (e.g., using MJOLLNIR [12]) we reduce (4) to  $\epsilon \leq 0.1$ , indicating that robustness for  $\mathcal{P}$  is ensured iff  $\epsilon$  does not exceed 0.1.

In Example 4 we showed how to exploit pWKSs to verify a simple WKSs against a given specification up-to some error.

Clearly, with an increasing complexity of the model (or the formula) it becomes necessary to have an automatic procedure to resolve (1). The following two sections are devoted to present a generalization of the model checking algorithm presented in [5] that can also accept WCTL formulas with negation.

## 4 Extended Parametric Dependency Graphs

Dependency graphs as originally introduced by Liu and Smolka [11] can be applied to model-checking of the alternation-free modal  $\mu$ -calculus, including its sub-logics like CTL. Jensen et al. [9] proposed to extend the dependency graphs framework using *cover-edges* and weighted *hyper-edges* for the verification of WKSs against negation-free WCTL formulas. Later, Christoffersen et al. [5] further generalised their approach to pWKSs by using parametric *hyper-edges* and *cover-edges*.

In this section we present an extension of the parametric dependency graph framework by incorporating a new type of edges, called *negation-edges*. Negation-edges were originally used in [6] for extending the applicability of dependency graphs w.r.t. CTL model checking of Petri Nets.

**Definition 5.** An Extended Parametric Dependency Graph (EPDG) is a tuple  $\mathcal{G} = (V, H, N, C)$  where  $V$  is a nonempty set of configurations and

- $H \subseteq V \times 2^{\mathcal{E} \times V}$  is a set of hyper-edges,
- $N \subseteq V \times V$  is a set of negation-edges, and
- $C \subseteq V \times \mathbb{Q}_{\geq 0} \times V$  is a set of cover-edges.

For  $v, u \in V$ , we write  $v \xrightarrow{f} u$  if  $(v, T) \in H$  and  $(f, u) \in T$ ;  $v \Rightarrow u$  if  $(v, u) \in N$ ;  $v \xrightarrow{q} u$  if  $(v, q, u) \in C$  and  $v$  and  $u$  are said resp. the *source* and the *target* configurations of the edge. We write  $v \rightsquigarrow u$  if  $v$  and  $u$  are respectively the source and target configurations of some edge in  $\mathcal{G}$  and,  $\rightsquigarrow^*$  for the reflexive and transitive closure of  $\rightsquigarrow$ .

We identify a class of EPDGs having some convenient structural properties.

**Definition 6.** Let  $\mathcal{G} = (V, H, N, C)$  be an EPDG.  $\mathcal{G}$  is safe if

- (i) its components are finite and for all  $(v, T) \in H$ ,  $T$  is finite.
- (ii) for all  $v \in V$   $|\{(v, u) \in N\} \cup \{(v, q, u) \in C\}| \leq 1$  and if  $|\{(v, u) \in N\} \cup \{(v, q, u) \in C\}| = 1$  then  $\{(v, T) \in H\} = \emptyset$ .
- (iii) there are no  $u, v \in V$  such that  $v \xrightarrow{q} u$  and  $u \rightsquigarrow^* v$ , or  $v \Rightarrow u$  and  $u \rightsquigarrow^* v$ .

Intuitively, to be safe an EPDG  $\mathcal{G}$  needs to have (i) finitely many configurations and edges, and each hyper-edge needs to be finitely branching; (ii) each of its configurations admits at most one type of outgoing edges and no cover edges or negation edges share the same source configuration; (iii) finally, no loop in  $\mathcal{G}$  shall have any cover- or negation-edges.

In the rest of the section we fix  $\mathcal{G} = (V, H, N, C)$  to be a safe EPDG.

We assign to each configuration  $v \in V$  a distance  $d(v) \in \mathbb{N}$  counting the maximum number of negation- and cover-edges in the paths starting from  $v$

$$d(v) \stackrel{\text{def}}{=} \max \{0, \sup \{d(v'') + 1 \mid v' \Rightarrow v'' \text{ or } v' \xrightarrow{q} v'' \text{ for } v', v'' \in V \text{ s.t. } v \rightsquigarrow^* v'\} \}.$$

Notice that the distance is bounded because  $\mathcal{G}$  is assumed to be safe.

We define  $d(\mathcal{G}) = \max_{v \in V} d(v)$ . The distance value is used to identify some components  $\mathcal{C}_0, \dots, \mathcal{C}_{d(\mathcal{G})}$ , where  $\mathcal{C}_i = (V_i, H_i, N_i, C_i)$  is the sub-EPDG of  $\mathcal{G}$  induced by the configurations  $V_i = \{v \in V \mid d(v) \leq i\}$ . Note that by construction  $N_0 = C_0 = \emptyset$ .

A valuation  $\mathbf{v} \in \mathbb{R}^k$  is said *admissible* for  $\mathcal{G}$  if whenever  $v \xrightarrow{f} u$  we have  $f(\mathbf{v}) \geq 0$ . We denote by  $\mathcal{V}_{\mathcal{G}}$  the set of admissible valuations for  $\mathcal{G}$ .

**Definition 7.** An assignment  $A$  of  $\mathcal{G}$  is a function  $A : V \rightarrow (\mathcal{V}_{\mathcal{G}} \rightarrow \overline{\mathbb{R}}_{\geq 0})$  where  $\overline{\mathbb{R}}_{\geq 0} = \mathbb{R}_{\geq 0} \cup \{\infty\}$ . The set of all assignments of  $\mathcal{G}$  is denoted  $\mathcal{A}^{\mathcal{G}}$ .

We equip  $\mathcal{A}^{\mathcal{G}}$  with the partial order  $\sqsubseteq \subseteq \mathcal{A}^{\mathcal{G}} \times \mathcal{A}^{\mathcal{G}}$  defined as

$$A_1 \sqsubseteq A_2 \quad \text{iff} \quad \forall v \in V. \forall \mathbf{v} \in \mathcal{V}_{\mathcal{G}}. A_1(v)(\mathbf{v}) \geq A_2(v)(\mathbf{v}).$$

$(\mathcal{A}^{\mathcal{G}}, \sqsubseteq)$  forms a complete lattice, with bottom element  $A_{\perp}$  and top element  $A_{\top}$  respectively defined as  $A_{\perp}(v)(\mathbf{v}) = \infty$  and  $A_{\top}(v)(\mathbf{v}) = 0$  for all  $v \in V$  and  $\mathbf{v} \in \mathcal{V}_{\mathcal{G}}$ . Given  $E \subseteq \mathcal{A}^{\mathcal{G}}$  such that  $E \neq \emptyset$  the greatest lower bound  $\prod E$  and least upper bound  $\bigsqcup E$  are defined, for arbitrary  $v \in V$  and  $\mathbf{v} \in \mathcal{V}_{\mathcal{G}}$ , as<sup>2</sup>

$$(\prod E)(v)(\mathbf{v}) = \sup_{A \in E} A(v)(\mathbf{v}), \quad (\bigsqcup E)(v)(\mathbf{v}) = \inf_{A \in E} A(v)(\mathbf{v}).$$

We are now ready to define the least fixed-point assignment of an EPDG  $\mathcal{G}$ .

**Definition 8.** The least fixed-point assignment for  $\mathcal{G}$ , denoted  $A_{\min}^{\mathcal{G}}$ , is defined inductively on its components  $\mathcal{C}_0, \dots, \mathcal{C}_{d(\mathcal{G})}$ . For  $0 \leq i \leq d(\mathcal{G})$ ,  $A_{\min}^{\mathcal{C}_i}$  is the least fixed-point of the function  $F_i : \mathcal{A}^{\mathcal{C}_i} \rightarrow \mathcal{A}^{\mathcal{C}_i}$ , defined as

$$F_i(A)(v)(\mathbf{v}) = \begin{cases} \chi(A_{\min}^{\mathcal{C}_{i-1}}(u)(\mathbf{v}) > 0) & \text{if } v \Rightarrow u \\ \chi(A_{\min}^{\mathcal{C}_{i-1}}(u)(\mathbf{v}) \leq q) & \text{if } v \xrightarrow{q} u \\ \min_{(v,T) \in H_i} \max_{(f,u) \in T} A(u)(\mathbf{v}) + f(\mathbf{v}) & \text{otherwise} \end{cases}$$

<sup>2</sup> As usual,  $\prod \emptyset = A_{\top}$  and  $\bigsqcup \emptyset = A_{\top}$ .

where  $\chi(p) = 0$  if the predicate  $p$  holds,  $\infty$  otherwise. We assume that  $\max \emptyset = 0$  and  $\min \emptyset = \infty$ .

**Lemma 9.** *Let  $i \in \{0, \dots, d(\mathcal{G})\}$  and  $\{A_j\}_{j \in \mathbb{N}} \subseteq \mathcal{A}^{C_i}$  be an ascending chain. Then,  $F_i(\bigsqcup_{j \in \mathbb{N}} A_j) = \bigsqcup_{j \in \mathbb{N}} F_i(A_j)$ , i.e.,  $F_i$  is  $\omega$ -continuous.*

**Corollary 10.**  *$F_i$  is monotonic for all  $i \in \{0, \dots, d(\mathcal{G})\}$ .*

By Knaster-Tarski's fixed-point theorem,  $A_{min}^{C_i}$  exists for all  $i \leq d(\mathcal{G})$ , moreover, by Kleene's fixed-point theorem, it is the limit of the ascending chain  $A_{\perp} \sqsubseteq F_i(A_{\perp}) \sqsubseteq F_i(F_i(A_{\perp})) \sqsubseteq \dots \sqsubseteq F_i^n(A_{\perp}) \sqsubseteq \dots$ , i.e.,  $\bigsqcup_{n \in \mathbb{N}} F_i^n(A_{\perp})$ .

The following result states that the limit of the above chain is reached within  $|V_i|$  steps. This result is essential for our algorithm.

**Lemma 11.** *Let  $i \in \{0, \dots, d(\mathcal{G})\}$  and  $k = |V_i|$ . Then,  $F_i^k(A_{\perp}^{C_i}) = A_{min}^{C_i}$ .*

By Lemma 11, we can compute  $A_{min}^{\mathcal{G}}$  symbolically by repeated application of  $F$  until we are sure that the fixed-point has been reached. It is worth noting that our termination condition only depends on the number of configurations of the EPDG. Therefore, in contrast with [5], we don't need to perform any symbolic comparison of the assignments to check whether a fixed-point has been reached. Not only does it simplify the algorithm, but it also reduces the overhead caused by symbolic comparison.

**Lemma 12.** *For any safe EPDG  $\mathcal{G} = (V, H, N, C)$  and component  $C_i$  of  $\mathcal{G}$ , the symbolic computation of the least fixed-point assignment,  $A_{min}^{C_i}$ , by repeated application of the function  $F_i$  on  $A_{\perp}^{C_i}$  runs in time  $\mathcal{O}(|V_i| \cdot (|H_i| + |N_i| + |C_i|))$ .*

## 5 Model Checking Parametric WKSS Using EPDGs

In this section we present a reduction from the model checking problem of WCTL on pWKSs to the computation of least fixed-point assignments for EPDGs. Then, we show how to obtain from those assignments a symbolic representation of (1) as a (quantifier-free) first-order formula in the linear theory of the reals.

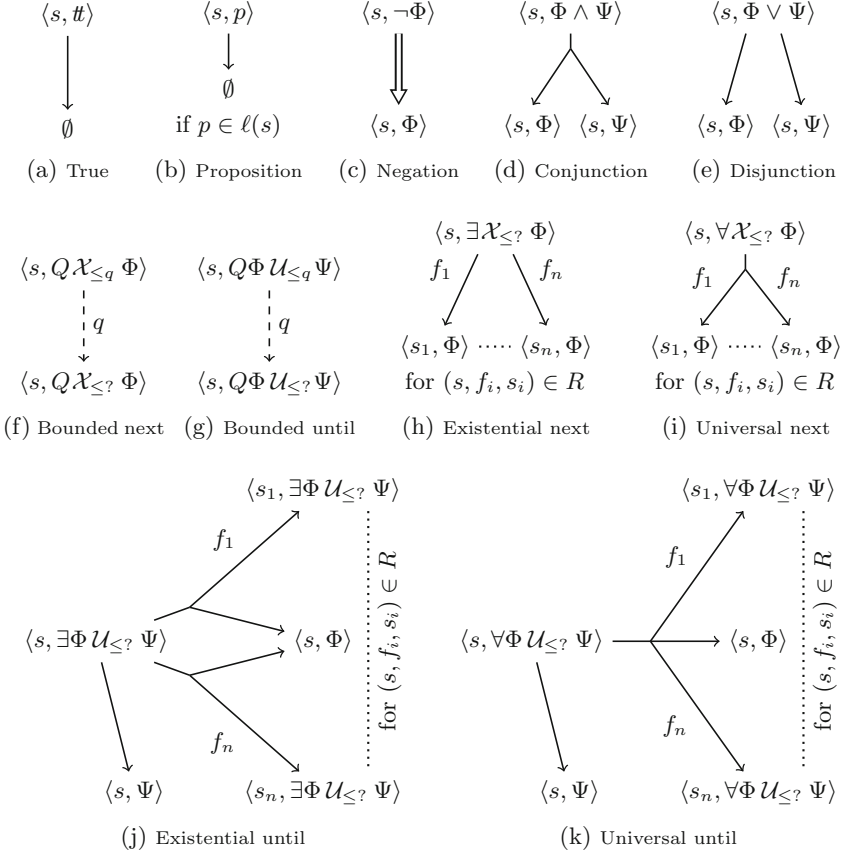
Given a pWKS  $\mathcal{P} = (S, R, \ell)$ , a state  $s \in S$  and a WCTL formula  $\Phi$ , we construct an EPDG  $\mathcal{G}$  where every configuration is a pair consisting of a state and a formula. Starting from the initial pair  $\langle s, \Phi \rangle$ ,  $\mathcal{G}$  is constructed according to the rules given in Fig. 2.

It is worth noting that the size of  $\mathcal{G}$  does not depend on the actual weight values of  $\Phi$  or  $\mathcal{P}$  but only on the size of  $\mathcal{P}$  and the number of sub-formulas of  $\Phi$ .

The following result ensures that the EPDG framework described in Sect. 4 can be applied to the EPDGs constructed according to the rules in Fig. 2.

**Lemma 13.** *The EPDG  $\mathcal{G}$  rooted at  $\langle s, \Phi \rangle$  is safe.*

In  $\mathcal{G}$  we distinguish two types of configurations: *concrete* configurations have concrete WCTL formulas, while *symbolic* configurations have *symbolic formulas* of the form  $Q \mathcal{X}_{\leq ?} \Phi$  or  $Q \Phi \mathcal{U}_{\leq ?} \Psi$  where  $Q \in \{\exists, \forall\}$  and  $\Phi, \Psi$  are concrete WCTL formulas. Given a symbolic formula  $\Phi$  and  $q \in \mathbb{Q}_{\geq 0}$ , we denote by  $\Phi_q$  the corresponding concrete formula with bound  $q$ .



**Fig. 2.** EPDG construction rules. Here  $Q \in \{\exists, \forall\}$  and hyper-edges without labels shall be assumed to be labelled with the constant weight map  $\mathbf{0}$ .

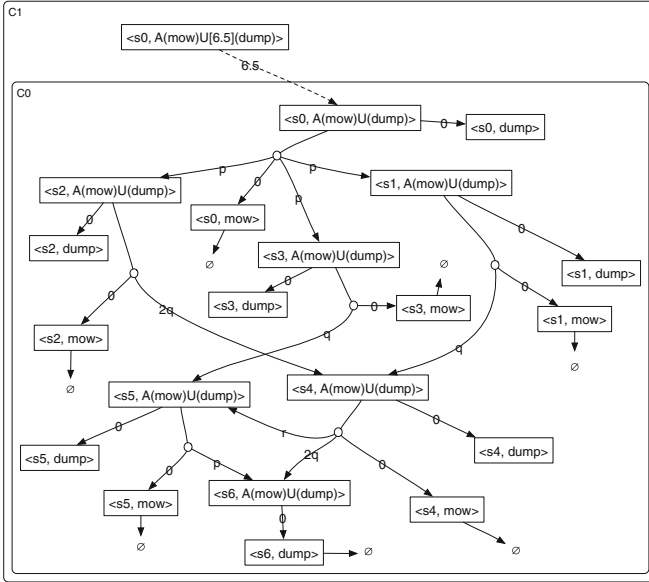
**Lemma 14.** *Let  $v = \langle s, \Phi \rangle$  be a concrete configuration of  $\mathcal{G}$  and  $\mathbf{v} \in \mathcal{V}_{\mathcal{G}}$  an admissible valuation. Then,  $A_{\min}^{\mathcal{G}}(v)(\mathbf{v}) \in \{0, \infty\}$ .*

The next theorem states that the set of correct valuations  $\llbracket \mathcal{P}, s \models \Phi \rrbracket$  corresponds to the set  $\{\mathbf{v} \in \mathcal{V}_{\mathcal{G}} \mid A_{\min}^{\mathcal{G}}(\langle s, \Phi \rangle)(\mathbf{v}) \leq 0\}$ . This reduces the model checking problem to the computation of least fixed-point assignments for EPDGs.

**Theorem 15.** *Let  $v = \langle s, \Phi \rangle$  be a configuration of  $\mathcal{G}$  and  $\mathbf{v} \in \mathcal{V}_{\mathcal{G}}$  an admissible valuation. Then, the following hold*

- (1) *if  $v$  is concrete, then  $A_{\min}^{\mathcal{G}}(v)(\mathbf{v}) = 0$  iff  $\mathcal{P}(\mathbf{v}), s \models \Phi$  and,*
- (2) *if  $v$  is symbolic, then for all  $q \in \mathbb{Q}$ ,  $A_{\min}^{\mathcal{G}}(v)(\mathbf{v}) \leq q$  iff  $\mathcal{P}(\mathbf{v}), s \models \Phi_q$ .*

We showed that  $A_{\min}^{\mathcal{G}}(\langle s, \Phi \rangle)$  can be computed symbolically as a partially evaluated expression. During the computation one can perform some simplifications (e.g.,  $\min \emptyset = \infty$  or  $\max \emptyset = 0$ ), nevertheless, the parts of the expression that depend on the actual value of the parameters are left unevaluated.



**Fig. 3.** EPDG rooted at  $\langle s_0, \forall(mow U_{\leq 6.5} dump) \rangle$  (cf. Example 16).

By Theorem 15 we are interested in a symbolic representation of the valuations  $\mathbf{v}$  such that  $A_{min}^{\mathcal{G}}(\langle s, \Phi \rangle)(\mathbf{v}) \leq 0$ . As anticipated in Example 4, this can be done by means of a (quantifier-free) first-order formula in the linear theory of the reals. In practice, such a formula is obtained as  $\Gamma(A_{min}^{\mathcal{G}}(\langle s, \Phi \rangle) \leq 0)$  where  $\Gamma$  is defined by cases as follows<sup>3</sup>, for  $\bowtie \in \{\leq, >\}$ ,  $m \in \{\min, \max\}$  and,  $q \in \mathbb{Q}_{\geq 0}$

$$\begin{aligned} \Gamma(\max\{e_1, \dots, e_n\} \bowtie q) &= \Gamma(e_1 \bowtie q) \wedge \dots \wedge \Gamma(e_n \bowtie q) \\ \Gamma(\min\{e_1, \dots, e_n\} \bowtie q) &= \Gamma(e_1 \bowtie q) \vee \dots \vee \Gamma(e_n \bowtie q) \\ \Gamma(\chi(b) \leq q) &= \Gamma(b) & \Gamma(\chi(b) > q) &= \neg\Gamma(b) \\ \Gamma(e + m\{e_1, \dots, e_n\} \bowtie q) &= \Gamma(m\{e + e_1, \dots, e + e_n\} \bowtie q) \\ \Gamma(e \bowtie q) &= e \bowtie q. & & \text{(if } e \text{ has no occurrence of } \min, \max \text{ or } \chi) \end{aligned}$$

*Example 16.* Consider the pWKS  $\mathcal{P}$  and the formula  $\Phi = \forall(mow U_{\leq 6.5} dump)$  from Example 4. In Fig. 3 is depicted the EPDG  $\mathcal{G}$  rooted at  $\langle s_0, \Phi \rangle$ . By running our symbolic algorithm we obtain the following expression

$$A_{min}^{\mathcal{G}}(\langle s_0, \Phi \rangle) = \chi(\max\{p + q + \max\{p + r, 2q\}, p + 2q + \max\{p + r, 2q\}, 2p + q\} \leq 6.5).$$

The above expression can be then turned into the following formula

$$2p + q + r \leq 6.5 \wedge p + 3q \leq 6.5 \wedge 2p + 2q + r \leq 6.5 \wedge p + 4q \leq 6.5 \wedge 2p + q \leq 6.5,$$

that, in conjunction with  $p \geq 0 \wedge q \geq 0 \wedge r \geq 0$  (cf. (2)) simplifies to (3).  $\square$

<sup>3</sup> To simplify the exposition, here unevaluated expressions are assumed to be modulo commutativity and associativity of  $+$ .



## 6 Weight-Uncertain Kripke Structures

In Sect. 3 we have seen how to use pWKSs for modelling and verifying the robustness of WKSs when the imprecision of the weights is quantified by means of an absolute accuracy error  $\epsilon$ . However, for an experimental weight value  $w$ , not all values in the interval  $w \pm \epsilon$  are equally likely to occur in practice.

It's common practice to model experimental measurements by means of real-valued random variables distributed according to well studied family of distribution (e.g., normal or student's T). In this section we introduce the notion of *weight-uncertain Kripke structures* (WUKSs), where weights are modelled as random variables and present a WCTL model checking framework for them.

Before we start let us recall some notions from measure theory.

*Measure Theory.* Let  $\Omega$  be a set. A family  $\Sigma \subseteq 2^\Omega$  is called  $\sigma$ -algebra if it contains the empty set  $\emptyset$  and is closed under complement and countable unions, in this case  $(\Omega, \Sigma)$  is said *measurable space* and elements of  $\Sigma$  *measurable sets*. If  $\Omega$  is given a topology then  $\mathcal{B}(\Omega)$  denotes the Borel  $\sigma$ -algebra of  $\Omega$ , i.e., the smallest  $\sigma$ -algebra having all open subsets of  $\Omega$ . We say that  $\Omega$  is a *Borel space* to indicate the measurable space  $(\Omega, \mathcal{B}(\Omega))$ , and elements of  $\mathcal{B}(\Omega)$  are called *Borel sets*. As an example,  $\mathbb{R}$  is assumed to have the usual Euclidean topology and  $\mathcal{B}(\mathbb{R})$  denotes the induced Borel  $\sigma$ -algebra which makes  $\mathbb{R}$  a Borel space.

A *measure* on  $(\Omega, \Sigma)$  is a  $\sigma$ -additive function  $\mu: \Sigma \rightarrow \mathbb{R}$ , i.e, a map satisfying  $\mu(\bigcup_{i \in I} E_i) = \sum_{i \in I} \mu(E_i)$  for any countable family of pairwise disjoint measurable sets  $(E_i)_{i \in I}$ , in this case  $(\Omega, \Sigma, \mu)$  is said *measure space*. If  $\mu$  additionally satisfies  $\mu(\Omega) = 1$ , it is called *probability measure* and  $(\Omega, \Sigma, \mu)$  *probability space*.

For  $(\Omega, \Sigma)$  and  $(Y, \Theta)$  measurable spaces, the map  $f: \Omega \rightarrow Y$  is *measurable* if for all  $E \in \Theta$ ,  $f^{-1}(E) = \{x \mid f(x) \in E\} \in \Sigma$ . Given a measurable map  $f: \Omega \rightarrow Y$  and a measure  $\mu$  on  $(\Omega, \Sigma)$  we define the measure  $\mu[f]$  on  $(Y, \Theta)$  as  $\mu[f](E) = \mu(f^{-1}(E))$ , for  $E \in \Theta$ , a.k.a. the *push forward of  $\mu$  under  $f$* .

A real-valued *random variable*  $X: \Omega \rightarrow \mathbb{R}$  is a measurable function from a probability space  $(\Omega, \Sigma, P)$  to the Borel space  $\mathbb{R}$ . Intuitively,  $X$  can be understood as the outcome value of an experiment (e.g., measuring some sensor value). Given a “test”  $A \in \mathcal{B}(\mathbb{R})$ , we write  $P[X \in A]$  for the probability that  $X$  has value in  $A$ , i.e.,  $P[X \in A] = P[X](A)$ . A random variable  $X$  is associated with its *cumulative distribution function* (CDF)  $F_X: \mathbb{R} \rightarrow [0, 1]$  defined as  $F_X(x) = P[X \in (-\infty, x]]$ ; and a *probability density function* (PDF)  $f_X$ , a non-negative Lebesgue-integrable function satisfying  $P[X \in [a, b]] = \int_a^b f_X(x)dx$ . The expected value of  $X$ , written  $E[X]$  is intuitively understood as the long-run average value of repetitions of the experiment  $X$ , formalised by the Lebesgue integral  $\int_{\Omega} X \, dP$  (corresponding to  $\int_{\mathbb{R}} f_X(x)dx$  when  $X$  admits density function  $f_X$ ).

In the rest of the section we fix the probability space  $(\Omega, \Sigma, P)$  representing the environment where the experiments are performed, and we use  $\mathbb{Y}$  to denote the set of real-valued random variables of the form  $Y: \Omega \rightarrow \mathbb{R}$ .

We are now ready to define the concept of weight-uncertain Kripke structure.

**Definition 17.** A weight-uncertain Kripke structure is a tuple  $\mathcal{J} = (S, R, \ell)$ , where  $S$  is a finite nonempty set of states,  $R \subseteq S \times \mathbb{Y} \times S$  is a finite random weighted transition relation and  $\ell: S \rightarrow 2^{A^P}$  is a labelling function.

Consider the WUKS  $\mathcal{J} = (S, R, \ell)$ . We denote by  $\text{WKS}_{\mathcal{J}}$  the set of all WKSs having the same underlying graph as  $\mathcal{J}$ . We construct the  $\sigma$ -algebra  $\Sigma_{\mathcal{J}}$  as the family of sets  $A \subseteq \text{WKS}_{\mathcal{J}}$  whose corresponding set of weights is Borel measurable in  $\mathbb{R}^m$  ( $m = |R|$ ). Formally,

$$A \in \Sigma_{\mathcal{J}} \quad \text{iff} \quad A \subseteq \text{WKS}_{\mathcal{J}} \text{ and } \{\omega(\mathcal{K}) \mid \mathcal{K} \in A\} \in \mathcal{B}(\mathbb{R}^m).$$

$\mathcal{J}$  can be seen as a measurable function  $\mathcal{J}: \Omega \rightarrow \text{WKS}_{\mathcal{J}}$ , where  $\mathcal{J}(\omega)$  is the WKS associated with  $\omega \in \Omega$ , justifying the intuition that it represents an experiment whose outcomes are WKSs. Accordingly, the semantics of  $\mathcal{J}$  is the probability space  $(\text{WKS}_{\mathcal{J}}, \Sigma_{\mathcal{J}}, P[\mathcal{J}])$ .

Given a WUKS  $\mathcal{J}$ , a state  $s \in S$ , and a WCLT property  $\Phi$ , two natural model checking questions are (i) whether the expected behaviour of  $\mathcal{J}$  satisfies  $\Phi$  at  $s$ , informally “ $E[\mathcal{J}], s \models \Phi$ ”, (ii) and how likely is that a concrete instance of  $\mathcal{J}$  satisfies  $\Phi$  at  $s$ , denoted by  $P[\mathcal{J}, s \models \Phi]$ .

We address the above problems for a subclass of WUKSs having random variables  $(Y: \Omega \rightarrow \mathbb{R}) \in \mathcal{E}_{\mathbf{X}}$  of the form  $Y(\omega) = \mathbf{a} \cdot \mathbf{X}(\omega) + b$ , with  $\mathbf{a} \in \mathbb{Q}_{\geq 0}^k$ ,  $b \in \mathbb{Q}_{\geq 0}$  and, where  $\mathbf{X} = (X_1, \dots, X_k)$  is vector of pairwise *independent* non-negative real-valued random variables<sup>4</sup>. Observe that, elements in  $\mathcal{E}_{\mathbf{X}}$  may not be independent from each other.

From here on we consider the WUKS  $\mathcal{J} = (S, \mathcal{E}, R, \ell)$  with  $R \subseteq S \times \mathcal{E}_{\mathbf{X}} \times S$ , and we use  $\mathcal{P}$  to refer to the pWKS obtained by replacing the random variables  $X_i$  in  $\mathcal{J}$  with the parameters  $x_i$  (for  $i = 1..k$ ).

Let’s consider the first question, namely “ $E[\mathcal{J}], s \models \Phi$ ”. There,  $E[\mathcal{J}]$  was informally denoting the WKS obtained by replacing each transition weight in  $\mathcal{J}$  with the corresponding expected value. Formally,  $E[\mathcal{J}]$  is defined as the unique  $\mathcal{K} \in \text{WKS}_{\mathcal{J}}$  such that  $\omega_i(\mathcal{K}) = \int_{\text{WKS}_{\mathcal{J}}} \omega_i \, dP[\mathcal{J}]$  for all  $i \in \{1, \dots, m\}$  where  $\omega_i: \text{WKS}_{\mathcal{J}} \rightarrow \mathbb{R}_{\geq 0}$  is the function that returns the  $i$ -th weight from a given WKS.

The assumption made on the weights in  $\mathcal{J}$  allows us to rephrase  $E[\mathcal{J}], s \models \Phi$  as a model checking problem for  $\mathcal{P}$ .

**Lemma 18.**  $E[\mathcal{J}], s \models \Phi$  if and only if  $E[\mathbf{X}] \in \llbracket \mathcal{P}, s \models \Phi \rrbracket$ .

We are now ready to address the second question, that is formalised as follows

$$P[\mathcal{J}, s \models \Phi] \stackrel{\text{def}}{=} P[\mathcal{J}](\{\mathcal{K} \in \text{WKS}_{\mathcal{J}} \mid \mathcal{K}, s \models \Phi\}). \quad (5)$$

For the above definition to be well-defined the set  $\{\mathcal{K} \in \text{WKS}_{\mathcal{J}} \mid \mathcal{K}, s \models \Phi\}$  needs to be a measurable event in  $\Sigma_{\mathcal{J}}$ . The following result ensures that.

**Lemma 19.**  $\{\mathcal{K} \in \text{WKS}_{\mathcal{J}} \mid \mathcal{K}, s \models \Phi\} \in \Sigma_{\mathcal{J}}$

<sup>4</sup> In fact, the vector  $\mathbf{X}$  is a multivariate random variable  $\mathbf{X}: \Omega \rightarrow \mathbb{R}^n$  with marginals  $X_i: \Omega \rightarrow \mathbb{R}_{\geq 0}$  ( $i = 1..n$ ).

The following theorem characterizes the model checking problem for the WUKS  $\mathcal{J}$  in terms of the model checking problem of its associated pWKS  $\mathcal{P}$ .

**Theorem 20.**  $P[\mathcal{J}, s \models \Phi] = P[\mathbf{X} \in \llbracket \mathcal{P}, s \models \Phi \rrbracket]$ .

*Remark 21.* For the sake of clarity, so far we have assumed that  $\mathbf{X}$  is non-negative real-valued random vector. However, provided that  $P[\mathbf{X} \in \mathcal{V}_{\mathcal{P}}] > 0$ , the non-negativity assumption can be dropped by replacing the probability distribution  $P[\mathbf{X}]$  with the conditional probability  $P[\mathbf{X} | \mathbf{X} \in \mathcal{V}_{\mathcal{P}}]$ .

By Theorem 20 we can estimate the value  $p$  of (5) by applying Monte Carlo simulation techniques. For this, we sample  $n$  independent repetitions of  $\mathbf{X}$ , associating with each repetition a Bernoulli random variable  $B_i$ . A realisation  $b_i$  of  $B_i$  is 1 if the corresponding sampled value of  $\mathbf{X}$  lays in  $\llbracket \mathcal{P}, s \models \Phi \rrbracket$ , and 0 otherwise. Finally, we estimate  $p$  by means of the observed relative success rate  $\tilde{p} = (\sum_{i=1}^n b_i)/n$ . The absolute error  $\varepsilon$  of the estimation can be bound with a certain degree of confidence  $\delta \in (0, 1]$  by tuning the number of required simulations based on the inequality  $P(|\tilde{p} - p| \geq \varepsilon) \leq \delta$  where  $\delta = e^{-2n\varepsilon^2}$  (cf. [10, 13]). Therefore the required number  $n$  of samples is obtained as

$$n = \left\lceil -\frac{\ln(\delta)}{2\varepsilon^2} \right\rceil. \quad (6)$$

*Example 22.* Consider the WUKS  $\mathcal{J}$  depicted in Fig. 1(right), where  $p$ ,  $q$  and,  $r$  shall now be interpreted as real-valued random variables distributed as  $p \sim \mathcal{N}(2, \epsilon)$ ,  $q \sim \text{unif}(1 - \epsilon, 1 + \epsilon)$ , and  $r \sim \mathcal{N}(0, \epsilon)$  for  $\epsilon = 0.1$ . We can estimate  $P[\mathcal{J}, s_0 \models \Phi] = 0.959$  with an error  $\varepsilon = 0.003$  and confidence of 99,9% (i.e.,  $\delta = 0.001$ ) by generating  $n = 383765$  samples.

## 7 Experimental Results

To evaluate the performance of the algorithms discussed in this paper, we developed a prototype tool suite for WCTL model checking of WKSs under uncertain weights. The tool suite consists of two parts: a back-end, called PVT<sub>TOOL</sub>2<sup>5</sup> and a front-end, called UV<sub>TOOL</sub><sup>6</sup>. UV<sub>TOOL</sub> supports the verification of pWKSs and WUKSs as described in Sects. 5 and 6 making use of the PVT<sub>TOOL</sub>2 which implements the EPDG construction and the symbolic fixed-point computation.

We have evaluated the PVT<sub>TOOL</sub>2 and the UV<sub>TOOL</sub> separately.

*Evaluation of the PVT<sub>TOOL</sub>2.* We compared the performance of the PVT<sub>TOOL</sub>2 with the PVT<sub>TOOL</sub> from [5]. For a fair comparison we used as benchmarks the vacuum cleaner models from [5] checking them against the WCTL formula  $\exists(\forall \text{dirty } \mathcal{U}_{\leq 10} \text{clean}) \mathcal{U}_{\leq 1000} \text{done}$ . The table depicted in Fig. 4a reports the results obtained by increasing the number of rooms in the vacuum cleaning model.

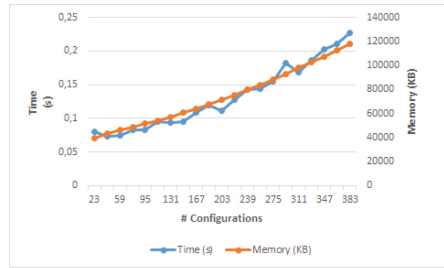
<sup>5</sup> The PVT<sub>TOOL</sub>2 is available at <https://github.com/AcId9381/PVTool>.

<sup>6</sup> The UV<sub>TOOL</sub> is implemented using Mathematica [16] and is available at <http://people.cs.aau.dk/~giovbacci/tools.html>.

The first and the second columns respectively present the number of states of the model and the number of configurations of the resulting EPDG, while the last two columns present respectively the computation time and the memory consumption of the two tools. The results of the experiments show that the PVTOOL2 performs worse than the PVTOOL on small models but it scales way better than the PVTOOL both in terms of computation time and memory consumption. This may be due to the fact that our algorithm does not perform any comparison of the symbolic assignments during the fixed-point computation. The improvement is measured when the overhead of the symbolic comparison exceeds the cost of the additional iterations required by the PVTOOL2.

Figure 4b shows how the computation time and memory consumption of the PVTOOL2 grows linearly in the number of configurations of the EPDG.

Model # states	EPGD # conf.	Time (s)		Memory (KB)	
		v1	v2	v1	v2
7	41	0.0015	0.073	1,004	43,616
13	77	0.017	0.083	1,504	48,686
19	113	0.190	0.095	3,808	54,274
25	149	0.250	0.096	14,264	60,780
31	185	35	0.119	60,548	67,806
34	203	781	0.112	263,832	71,667
40	239	N/A	0.142	N/A	79,530
46	275	N/A	0.155	N/A	88,252
52	311	N/A	0.168	N/A	97,889
58	347	N/A	0.202	N/A	107,574
64	383	N/A	0.227	N/A	118,044



(a) Comparison with the PVTOOL from [5]

(b) Performance of the PVTOOL2

**Fig. 4.** Experiments on an Intel i7 (5<sup>th</sup> gen.) 2.6 GHz processor with 12 GB RAM

*Evaluation of the UVTOOL.* For the verification of WUKS, our algorithm first samples valuations from  $\mathbf{X}$ , then estimates the relative number of valuation-samples that are correct in the sense of (1). Alternatively, one could first sample WKSs from the given WUKS and then estimate the relative number of models that satisfy the specification. In the second approach one could employ the WKTOOL<sup>7</sup> and exploit the efficient local algorithm from [9].

We compared the two approaches on the WUKS of Example 22 and performed the evaluation with increasing precision and accuracy of the estimation. The results are presented in Table 1. The first three columns report the error, the confidence and the number of generated samples (*cf.* Eq. (6)), and the last two columns present the computation time respectively for the UVTOOL and the adaptation of the WKTOOL. It is worth mentioning that the values reported in the last column do not consider the time required to sample and generate the models, but only the total time used for the model checking. The results clearly show that our approach outperforms the second one by several orders of magnitude, showing that computing the symbolic representation of the correct valuations in advance gives a huge speed-up in the overall computation time.

<sup>7</sup> The WKTOOL is available at <https://github.com/jonasfj/WKTool>.

**Table 1.** Experiments on an Intel Core i5 3.1 GHz with 8 GB RAM.

Error $\varepsilon$	Confidence $\delta$	# samples	UVTOOL (s)	WKT00L (s)
0.02	0.01	5,757	0.137	181.009
0.01	0.01	23,026	0.533	724.206
0.01	0.001	34,539	0.828	1086.88
0.005	0.001	138,156	3.231	4,347.96
0.003	0.001	383,756	8.876	5,886.670

## 8 Conclusion and Future Work

We addressed the model checking problem of weighted Kripke structures under uncertainty. We proposed to employ parametric weighted Kripke structures and weight-uncertain Kripke structures for modelling WKSs with imprecise real-valued weights. For the verification of pWKSs against WCTL formulas we developed a model checking algorithm that, compared with [5], implements an improved termination condition and accepts formulas with negation. The algorithm, given a pWKS and a WCTL formula, and produces a quantifier free first-order formula in the linear theory of the reals representing the set of parameter valuations satisfying the specification. The outcome formula is then used as underlying ingredient for verifying the robustness of WKSs. If the imprecision of the weights by means of an absolute accuracy error the verification can be performed via quantifier elimination (*cf.* Example 4). Otherwise, if the imprecision is quantified by mean of random variables, the probability of satisfying the specification is estimated via Monte Carlo simulation techniques (*cf.* Example 22).

In the future we plan to consider an alternative semantic interpretation for WUKSs where the random weights are dynamically sampled while unfolding the model, thus modelling WKSs with an infinite state space. This alternative semantics would fit well in the contexts of reactive systems that respond to external stimuli whose values are uncertain. Another direction for future work would be to consider the model checking of weighted LTL properties under uncertainty.

## References

1. Avni, G., Kupferman, O.: Stochastization of weighted automata. In: Italiano, G.F., Pighizzini, G., Sannella, D.T. (eds.) MFCS 2015. LNCS, vol. 9234, pp. 89–102. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-48057-1\\_7](https://doi.org/10.1007/978-3-662-48057-1_7)
2. Bacci, G., Hansen, M., Larsen, K.G.: On the verification of weighted Kripke structures under uncertainty. Full version, Aalborg University (2018). <http://people.cs.aau.dk/~giobvacci/papers/uncertwks-full.pdf>
3. Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press, Cambridge (2008)
4. Chatterjee, K., Doyen, L., Henzinger, T.A.: Probabilistic weighted automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 244–258. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-04081-8\\_17](https://doi.org/10.1007/978-3-642-04081-8_17)

5. Christoffersen, P., Hansen, M., Mariegaard, A., Ringsmose, J.T., Larsen, K.G., Mardare, R.: Parametric verification of weighted systems. In: 2nd International Workshop on Synthesis of Complex Parameters, SynCoP 2015, April 11, 2015, London, United Kingdom, pp. 77–90 (2015)
6. Dalsgaard, A.E., et al.: Extended dependency graphs and efficient distributed fixed-point computation. In: van der Aalst, W., Best, E. (eds.) PETRI NETS 2017. LNCS, vol. 10258, pp. 139–158. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-57861-3\\_10](https://doi.org/10.1007/978-3-319-57861-3_10)
7. Droste, M., Kuich, W., Vogler, H. (eds.): Handbook of Weighted Automata, 1st edn. Springer Publishing Company, Incorporated, Heidelberg (2009). <https://doi.org/10.1007/978-3-642-01492-5>
8. Fahrenberg, U., Larsen, K.G., Thrane, C.: A quantitative characterization of weighted Kripke structures in temporal logic. *Comput. Inf.* **29**, 1311–1324 (2010)
9. Jensen, J.F., Larsen, K.G., Srba, J., Oestergaard, L.K.: Efficient model-checking of weighted CTL with upper-bound constraints. *STTT* **18**(4), 409–426 (2016)
10. Singh Kambo, N., Kotz, S.: On exponential bounds for binomial probabilities. *Ann. Inst. Stat. Math.* **18**(1), 277 (1966)
11. Liu, X., Smolka, S.A.: Simple linear-time algorithms for minimal fixed points (extended abstract). In: Proceedings of the Automata, Languages and Programming, 25th International Colloquium, ICALP 1998, Aalborg, Denmark, July 13–17, 1998, pp. 53–66 (1998)
12. Monniaux, D.: Quantifier elimination by Lazy model enumeration. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 585–599. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14295-6\\_51](https://doi.org/10.1007/978-3-642-14295-6_51)
13. Okamoto, M.: Some inequalities relating to the partial sum of binomial probabilities. *Ann. Inst. Stat. Math.* **10**(1), 29–35 (1959)
14. Srkk, S.: Bayesian Filtering and Smoothing. Cambridge University Press, New York (2013)
15. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: 26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21–23 October 1985, pp. 327–338. IEEE Computer Society (1985)
16. Wolfram Research, Inc., Mathematica, Version 11.2. Champaign, IL (2017)



# Hospital Inventory Management Through Markov Decision Processes @runtime

Marco Biagi, Laura Carnevali<sup>(✉)</sup>, Francesco Santoni, and Enrico Vicario

Department of Information Engineering, University of Florence, Florence, Italy  
{marco.biagi,laura.carnevali,francesco.santoni,enrico.vicario}@unifi.it

**Abstract.** Stock management in a hospital requires the achievement of a trade-off between conflicting criteria, with mandatory requirements on the quality of patient care as well as on purchasing and logistics costs. We address daily drug ordering in a ward of an Italian public hospital, where patient admission/discharge and drug consumption during the sojourn are subject to uncertainty. To derive optimal control policies minimizing the overall purchasing and stocking cost while avoiding drug shortages, the problem is modeled as a Markov Decision Process (MDP), fitting the statistics of hospitalization time and drug consumption through a discrete phase-type (DPH) distribution or a Hidden Markov Model (HMM). A planning algorithm that operates at run-time iteratively synthesizes and solves the MDP over a finite horizon, applies the first action of the best policy found, and then moves the horizon forward by one day. Experiments show the convenience of the proposed approach with respect to baseline inventory management policies.

**Keywords:** Inventory management · Markov decision processes  
Receding horizon · Discrete phase-type distributions  
Hidden markov models

## 1 Introduction

Stock replenishment in a hospital is a complex and critical task, facing a number of major conflicting challenges [16, 23]. On the one hand, the quality of patient care is subject to mandatory requirements, even allowing expensive emergency orders to avoid stock-out danger. On the other hand, hospitals are subject to limitations in budget and storage space. Additional factors of complexity include uncertainty on patient admission/discharge, limited predictability of drug consumption, imprecise monitoring of stock state, and delays in drug delivery. Despite these complexities, orders are typically performed by nurses on a daily basis and in a manual fashion, resulting in a time consuming and yet imperfect process, with frequent emergency orders. According to this, optimization of inventory management has been advocated as a means to improve the efficiency of healthcare services while maintaining the same clinical guarantees [10], motivating the investigation in quantitative approaches to decision support.

Markov Decision Processes (MDPs) [20] have been widely used to solve inventory control problems in a variety of contexts [24], supporting determination of the optimal reorder points and quantities over the decision-making horizon. An MDP inventory problem is formulated in [5] considering perishable commodities with selling cost depending on their lifetime, deriving the policy that maximizes the average profit per period. In [6], solution of an MDP yields the optimal ordering policy for a retailer with two suppliers, one perfectly reliable and one less reliable but offering a lower price. An integrated approach is proposed in [12] to coordinate inventory management in the stages of supply, manufacturing, and distribution of a supply chain subject to customer demand uncertainty and lead time variability, solving an MDP through a reinforcement learning algorithm. As a common trait, all these approaches formulate an *infinite-horizon* optimization problem, often too computationally-intensive to be solved online.

In run-time environments, complexity is managed by repeatedly solving the optimization problem over a *finite-horizon*, leveraging a *model@run.time* [8] that adapts its behavior in response to new observations and measurements. The result of the optimization is exploited according to the *receding horizon* philosophy typical of Model Predictive Control (MPC) [11], applying the first move of the optimal command sequence while discarding the remaining optimal moves, and then solving a new optimal control problem at the next time step. In [15], MPC is applied to minimize the stock-out probability in a hospital pharmacy while limiting the overall cost and the number of orders. Demand is modeled as a stochastic disturbance process, yet without deriving a probabilistic characterization from historical data. Inventory control problems for the management of distribution chains are solved in [7] by combining MPC with min-max optimization, considering the uncertainty on long-term predictions of customer demand.

The feedback loop of receding horizon control perfectly fits the MAPE-K pattern (*Monitor, Analyze, Plan, Execute, Knowledge*), initially proposed by IBM for autonomic computing systems [1]. Despite the different application context, self-adaptation of software-intensive systems under environment uncertainty relies on a control problem formulation similar to that of inventory management with stochastic supply and demand. In [18], an optimal control policy for self-adaptive systems is derived solving an MDP over a finite horizon, considering the latency in the execution of different adaptation tactics. An MPC self-adaptation strategy is computed in [14] for software systems described by queuing networks, exploiting a linearization technique to efficiently solve the non-linear optimization problem and continuously meet the desired performance requirements.

In this paper, we present a probabilistic approach to stock replenishment in a hospital ward, aimed to minimize the overall cost for purchasing and stocking pharmaceuticals while preventing stock-outs. The approach is developed with reference to the case of a ward in a public Italian hospital, where patient sojourn time and drug consumption are subject to uncertainty, and drugs have a purchasing cost and a stocking cost, with no additional expenses for standard orders but with a charge on emergency orders. The problem is modeled as a Markov Decision Process (MDP), fitting the statistics of sojourn time and drug consumption through a discrete phase-type (DPH) distribution or a Hidden Markov



Model (HMM). At run-time, the MDP parameters are derived and the model is solved over a finite horizon, applying the first action of the best policy found and then moving the horizon forward by one day. A sensitivity analysis is performed on a real data set with respect to the stocking cost per drug unit, showing the convenience of the approach compared to baseline policies.

Among reviewed works, the proposed method has similarities with the approach of [18] in the way an MDP is used to build a receding horizon controller. However, this paper addresses a much different application context and proposes a completely different methodology to synthesize the MDP, not using an autoregressive time series predictor as in [18], but rather leveraging domain knowledge to define the states of the model, and exploiting the statistics of hospitalization time and drug consumption to derive probabilistic transitions between them.

The rest of the paper is organized in four sections. Section 2 formulates the problem of stock replenishment with reference to the considered application scenario; Sect. 3 presents the solution approach; and, Sect. 4 illustrates the experimental results. Finally, conclusions are drawn in Sect. 5.

## 2 Application Context

We address stock replenishment in a ward of a public hospital in Tuscany, Italy. Information and data were collected as a part of the LINFA project (Intelligent Pharmaceutical Logistics) [4], funded by the Regional Government of Tuscany for the investigation of technologies and methods supporting smart stock replenishment in wards of the Tuscan healthcare system. In the following, we formulate the problem with reference to the specific application context (Sect. 2.1) and we illustrate available data on drug consumption (Sect. 2.2).

### 2.1 Problem Formulation

In the public healthcare system of Tuscany, small-sized hospital wards independently issue orders to a central supplier. There is neither a specific employee in charge of inventory management, nor a decision support system or a training course that can help health professionals in performing such task: orders are placed by nurses on a daily basis and in a manual fashion, depending on their work shift and personal experience. Each drug has a purchasing cost per unit in case of standard orders, and an increased cost in case of emergency orders, with no fixed cost for the order itself. Each drug has also a stocking cost per unit per day, representing a penalty on drug units stored and not yet consumed.

The amount of drug units consumed in a ward depends on several factors, notably including the number of patients, the duration of the stay of each patient, the healthcare protocol assigned to each patient as well as his/her physical characteristics and response to the treatment. Depending on the type of ward, patient arrivals are distinguished in scheduled arrivals, which are known in advance, and urgent arrivals (e.g., in an emergency ward, no arrival is scheduled). The number of patient arrivals in a ward is also limited by the maximum number  $P_{\max}$  of

available beds. The number of days of hospitalization of a patient has no maximum and a minimum equal to zero, which corresponds to the case that the patient is discharged the same day of admission.

We address the problem of stock replenishment for a single drug over a time horizon of  $H$  days, with the aim of minimizing the overall purchasing and stocking cost while avoiding stock-outs. To this end, let  $c_d$ ,  $c_u > c_d$ , and  $c_s$  be the standard purchasing cost per unit, the purchasing cost per unit in case of emergency orders, and the stocking cost per unit per day, respectively. And, let  $q_d(h)$  and  $q_u(h)$  be the number of drug units restocked on the  $h$ -th day of the horizon through a standard order and an emergency order, respectively, and let  $q_s(h)$  be the number of drug units stocked at the beginning of the  $h$ -th day of the horizon. According to this notation, the objective function is defined as follows:

$$\min \sum_{h=1}^H q_d(h) c_d + q_u(h) c_u + q_s(h) c_s \quad (1)$$

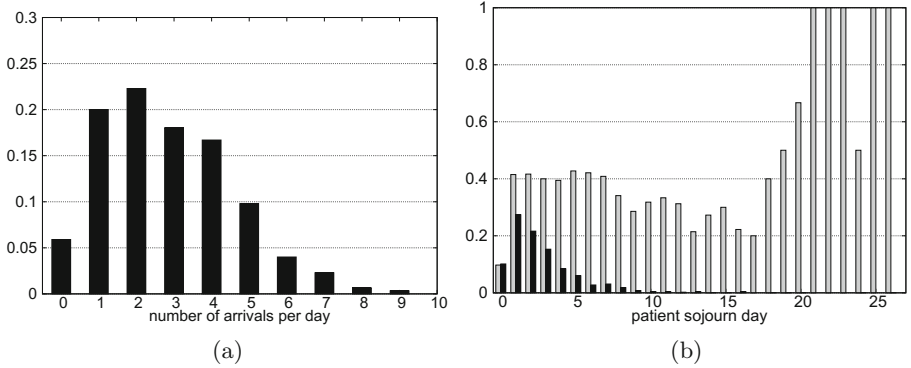
where  $0 \leq q_d(h) + q_u(h) + q_s(h) \leq S_{max}$  guarantees that the number of drug units restocked plus the number of those already stocked and not yet used does not exceed the maximum number  $S_{max}$  of drug units that can be stored.

## 2.2 Drug Consumption Data

Data on drug consumption were collected in a ward of a public hospital in Tuscany, Italy, under a non-disclosure agreement, and therefore they can be described only in an aggregate form. The data set is composed of nearly 30 000 entries describing drug units consumed by nearly 1000 patients hospitalized in the ward within about a year. The data set has an entry for each drug intake of each patient. Each entry is a tuple  $\langle \text{id}, \mathbf{t}_{\text{intake}}, \mathbf{t}_{\text{end}}, \mathbf{n}, \mathbf{d} \rangle$ , where  $\text{id}$  is the anonymous unique identifier of the patient,  $\mathbf{t}_{\text{intake}}$  is the hospitalization day during which the drug intake was consumed,  $\mathbf{t}_{\text{end}}$  is the discharge date of the patient,  $\mathbf{n}$  is the number of consumed drug units, and  $\mathbf{d}$  is the drug type.

While treatment protocols applied to patients are not always known and thus cannot be explicitly used to predict drug usage and support stock replenishment, statistics on drug consumption can be derived from collected data. Figure 1a shows the Probability Mass Function (PMF) of the number of patient arrivals per day, with average around 2.80 and maximum equal to 9. Though in the considered ward arrivals can be either scheduled or emergency, the data set does not distinguish between them. The average number of patients in the ward per day (not shown in Fig. 1a) is approximately equal to 7.62.

Figure 1b shows the PMF of the number of hospitalization days (black curve), with average approximately equal to 2.72 and maximum equal to 26. Figure 1b also shows the probability to consume a unit of a specific drug in each day of hospitalization (gray curve), computed for the most widely used drug in the ward during the period in which data were collected, which is the drug considered in the experiments reported in this paper. The plot shows that the probability of drug consumption from day 1 to day 7 of hospitalization is almost constant and



**Fig. 1.** (a) Probability mass function of the number of patient arrivals per day. (b) Probability mass function of the number of hospitalization days (black), and probability to consume a unit of a specific drug in each day of hospitalization (gray), computed for the most consumed drug in the ward.

equal to 0.4. The maximum number of drug units consumed by each patient within a single day (not shown in Fig. 1b) is equal to 1.

### 3 Solution Method

We solve the problem of daily stock replenishment by repeatedly performing probabilistic model checking of an MDP over a finite horizon. In the following, we define the MDP model and we present the solution method (Sect. 3.1), illustrating how the statistics of hospitalization time and drug consumption can be fitted through a DPH distribution (Sect. 3.2) or an HMM (Sect. 3.3).

#### 3.1 Probabilistic Model Checking of an MDP @runtime

MDPs [20] are popular for modeling systems that feature both probabilistic and nondeterministic state transitions, supporting the derivation of the optimal sequence of nondeterministic choices with respect to a given reward function.

**Definition 1.** An MDP is a tuple  $\mathcal{M} = \langle S, s_0, A, P, R \rangle$  where:  $S$  is a finite set of states;  $s_0 \in S$  is the initial state;  $A$  is a set of nondeterministic actions;  $P : S \times A \times S \rightarrow [0, 1]$  is the state transition probability matrix such that (i)  $\sum_{s' \in S} P(s, a, s') \in \{0, 1\} \forall s \in S$  and  $\forall a \in A$ , and (ii)  $\forall s \in S \exists a \in A$  such that  $\sum_{s' \in S} P(s, a, s') = 1$ ; and,  $R : S \times A \rightarrow \mathbb{R}_{\geq 0}$  is a reward function, assigning to each state-action pair a non-negative reward.

In each state  $s \in S$ , a *nondeterministic choice* is made in the set of enabled actions  $A(s) = \{a \in A \mid \sum_{s' \in S} P(s, a, s') = 1\}$ . A reward of  $R(s, a)$  is obtained, and a successor state  $s'$  is chosen with probability  $P(s, a, s')$  through a probabilistic choice. A *path* is an execution that resolves both nondeterministic and

probabilistic choices, i.e.,  $\gamma = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots$ , where  $s_i \in S$ ,  $a_i \in A(s_i)$ , and  $P(s_i, a_i, s_{i+1}) > 0 \forall i \in \mathbb{N}$ . A *policy* resolves nondeterministic choices only, selecting which action to take in each state.

**Definition 2.** A *policy* of an MDP is a function  $\lambda : \mathcal{P}^* \rightarrow \mathcal{D}(A)$  s.t.  $\lambda(\gamma, a) = 0 \forall a \notin A(\text{last}(\gamma))$ , where  $\mathcal{P}^*$  is the set of infinite paths;  $\mathcal{D}(a)$  is the set of discrete probability distributions over  $A$ ;  $\text{last}(\gamma)$  is the last state of path  $\gamma$ ; and  $\lambda(\gamma, a)$  is the probability of choosing  $a$  at state  $\text{last}(\gamma)$  under  $\lambda$ .

The problem of stock replenishment defined in Sect. 2 can be formulated as an MDP where states represent the number of stored drug units in each day of the horizon, nondeterministic actions model the choice to order a certain number of drug units, and probabilistic transitions account for drug consumption by the patients. The MDP can be specified through the high-level PRISM language [17] as a module with constants and variables. Listing 1.1 shows a fragment of the PRISM code, consisting of a module termed `WARD` (lines 12–48) and a reward termed `cost` (lines 50–59). The code is instantiated for a finite horizon of  $H = 5$  days (constant `horizonDuration`, line 5), standard purchasing cost per drug unit  $c_d = 10$  (constant `drugCost`, line 6), emergency purchasing cost per drug unit  $c_u = 40$  (constant `urgentDrugCost`, line 7), stocking cost per drug unit per day  $c_s = 2$  (constant `stockCost`, line 8), and maximum number  $S_{max} = 30$  of stored drug units (constant `maxStock`, line 9).

**States.** The set of states of the MDP is the set of reachable combinations of values of the integer variables `day` (representing the current day of the horizon, line 13), `stock` (modeling the number of stored drug units, line 14), and `s` (comprising an index variable used to repeat the commands of drug ordering, drug consumption, and reward evaluation for each day of the horizon, line 15). Specifically,  $S \subseteq [0, \text{horizonDuration}] \times [-\text{maxMissed}, \text{maxStock}] \times [0, 7] = [0, 5] \times [-30, 30] \times [0, 7]$ , where constant `maxMissed` = 30 (line 10) is the maximum number of consumed drug units per day, obtained as the product of the maximum number of hospitalized patients (i.e., 30) and the maximum number of consumed drug units per patient per day (i.e., 1). Negative values of `stock` account for missing drug units to be supplied through an emergency order.

The initial state is  $s_0 = \langle \text{day} = 0, \text{stock} = 6, \mathbf{s} = 0 \rangle$ , where the value of `stock` is the best threshold obtained for the order-up-to  $(R, T)$  policy in the experiments of Sect. 4. Starting from  $s_0$ , a nondeterministic choice is performed representing the standard order issued on day 0 (lines 17–23), and the corresponding cost is evaluated (lines 51–56). Then, the following steps indexed by variable `s` are repeated for each day  $h \in \{1, \dots, \text{horizonDuration}\}$ : (1) time is moved forward by one day increasing variable `day` by 1 (`s` = 1, line 25); (2) a probabilistic transition representing drug consumption on day  $h$  is performed (`s` = 2, lines 27–31); (3) an urgent order is issued in case of missing drug units (`s` = 3, line 32), and the corresponding cost is evaluated (line 57); (4) the stocking cost is evaluated (`s` = 4, line 35); (5) a standard order is issued on day  $h$  (`s` = 5, lines 36–42), and the corresponding cost is evaluated (lines 51–56); (6) termination is checked by comparing the value of variable `day` against constant `horizonDuration` (`s` = 6,

lines 45–47), repeating steps 1–6 if `day < horizonDuration`, otherwise reaching a final absorbing state where `s = 7`. In doing so, a standard order is issued on day 0, day 1, ..., and day `horizonDuration-1` based on the expected drug consumption on the next day, with the aim of minimizing the overall cost for standard purchasing, emergency purchasing, and stocking. Conversely, the last standard order is not followed by a next day, thus being associated with a purchasing cost but neither with an emergency cost nor with a stocking cost. Therefore, cost minimization yields a solution where no drug is ordered on day `horizonDuration`.

**Listing 1.1.** Fragment of the PRISM code.

```

1  mdp
2
3  label "horizonEnd" = s=7;
4
5  const int horizonDuration = 5;
6  const int drugCost = 10;
7  const int urgentDrugCost = 40;
8  const int stockCost = 2;
9  const int maxStock = 30;
10 const int maxMissed = 30;
11
12 module WARD
13   day : [0..horizonDuration] init 0;
14   stock : [-maxMissed..maxStock] init 6;
15   s : [0..7] init 0;
16   // 0) Standard order
17   [order0] s=0 -> (s'=1);
18   [order5] s=0 -> (stock'=min(maxStock, stock+5)) & (s'=1);
19   [order10] s=0 -> (stock'=min(maxStock, stock+10)) & (s'=1);
20   [order15] s=0 -> (stock'=min(maxStock, stock+15)) & (s'=1);
21   [order20] s=0 -> (stock'=min(maxStock, stock+20)) & (s'=1);
22   [order25] s=0 -> (stock'=min(maxStock, stock+25)) & (s'=1);
23   [order30] s=0 -> (stock'=min(maxStock, stock+30)) & (s'=1);
24   // 1) Move time forward of one day
25   [] s=1 -> (s'=2) & (day'=day+1);
26   // 2) Drug consumption
27   [] s=2&(day=1) -> 0.018:(stock'=stock)&(s'=3) + 0.982:(stock'=stock-1)&(s'=3);
28   [] s=2&(day=2) -> ... <omitted> ...;
29   [] s=2&(day=3) -> ... <omitted> ...;
30   ... <omitted> ...
31   [] s=2&(day=horizonDuration) -> ... <omitted> ...;
32   // 3) Urgent order
33   [missingDrugs] s=3 -> (stock'=max(stock, 0)) & (s'=4);
34   // 4) Stocking cost evaluation
35   [storageDrugs] s=4 -> (s'=5);
36   // 5) Standard order
37   [order0] s=5 -> (s'=6);
38   [order5] s=5 -> (stock'=min(maxStock, stock+5)) & (s'=6);
39   [order10] s=5 -> (stock'=min(maxStock, stock+10)) & (s'=6);
40   [order15] s=5 -> (stock'=min(maxStock, stock+15)) & (s'=6);
41   [order20] s=5 -> (stock'=min(maxStock, stock+20)) & (s'=6);
42   [order25] s=5 -> (stock'=min(maxStock, stock+25)) & (s'=6);
43   [order30] s=5 -> (stock'=min(maxStock, stock+30)) & (s'=6);
44   // 6) Check termination
45   [] (s=6)&(day<horizonDuration) -> (s'=1);
46   [] (s=6)&(day=horizonDuration) -> (s'=7);
47   [] s=7 -> (s'=7);
48 endmodule
49
50 rewards "cost"
51 [order5] true : (5*drugCost);
52 [order10] true : (10*drugCost);
53 [order15] true : (15*drugCost);
54 [order20] true : (20*drugCost);
55 [order25] true : (25*drugCost);
56 [order30] true : (30*drugCost);
57 [missingDrugs] true : urgentDrugCost*( - min(stock, 0) );
58 [storageDrugs] true : stockCost*stock;
59 endrewards

```

**Nondeterministic Actions.** The set of actions of the MDP collects nondeterministic choices on the number of drug units to order, which can be specified in the PRISM language by multiple concurrently enabled guarded commands of the form  $[action] guard \rightarrow update$ ; changing the value of variables according to the *update* if the *guard* condition is satisfied. In Listing 1.1, the

minimum number of drug units that can be ordered is 5, Hence, actions model the choice of ordering 0, 5, 10, 15, 20, 25, or 30 drug units (lines 17–23 and 37–43), i.e.,  $A = \{\text{order0}, \text{order5}, \text{order10}, \text{order15}, \text{order20}, \text{order25}, \text{order30}\}$ . Each action increases variable `stock` by the corresponding number of drug units, e.g., `order5` increases `stock` by 5.

**Probabilistic Transitions.** The state transition probability matrix of the MDP depends on the probability  $p_{\text{intake}}^{\text{day}}(i|h)$  that  $i$  drug units are consumed by the hospitalized patients during the  $h$ -th day of the horizon. Specifically, for any pair of states  $s_u, s_v \in S$  and for any action  $a \in A(s_u)$ , the transition probability from  $s_u$  to  $s_v$  through  $a$  is defined as:

$$P(s_u, a, s_v) = \begin{cases} p_{\text{intake}}^{\text{day}}(i|h) & \text{if } s_u = \langle h, r, 2 \rangle \\ & \wedge s_v = \langle h, r - i, 3 \rangle \\ \\ 1 & \text{if } (s_u = \langle \cdot, r, \mathbf{s} \rangle \\ & \wedge s_v = \langle \cdot, \min\{\text{maxStock}, r + t\}, \mathbf{s} + 1 \rangle \\ & \wedge \mathbf{s} \in \{0, 5\}) \\ & \vee (s_u = \langle h, \cdot, 1 \rangle \wedge s_v = \langle h + 1, \cdot, 2 \rangle) \\ & \vee (s_u = \langle \cdot, \cdot, 3 \rangle \wedge s_v = \langle \cdot, \max\{r, 0\}, 4 \rangle) \\ & \vee (s_u = \langle \cdot, \cdot, 4 \rangle \wedge s_v = \langle \cdot, \cdot, 5 \rangle) \\ & \vee (s_u = \langle h, \cdot, 6 \rangle \wedge s_v = \langle h, \cdot, 1 \rangle \wedge h < 5) \\ & \vee (s_u = \langle \text{horizonDuration}, \cdot, 6 \rangle \\ & \wedge s_v = \langle \text{horizonDuration}, \cdot, 7 \rangle) \\ & \vee (s_u = \langle \cdot, \cdot, 7 \rangle \wedge s_v = \langle \cdot, \cdot, 7 \rangle) \\ \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where  $t$  is the number of ordered drug units when  $\mathbf{s} \in \{0, 5\}$ .

Probabilistic transitions can be specified in the PRISM language by guarded commands of the form  $[action] guard \rightarrow prob_1 : update_1 + \dots + prob_n : update_n$ ; associating each update with a nonzero probability; when there is a single *update*, consequently with probability 1, the preceding "prob :" can be omitted. In particular, commands modeling consumption of drug units are defined for each day of the horizon (lines 27–31) with at most  $S_{\text{max}} + 1$  updates, representing the probability of consuming 0, 1,  $\dots$ ,  $S_{\text{max}}$  drug units, respectively. For example, in Listing 1.1, at most one drug unit can be consumed on day 1 with probability 0.982, i.e.,  $p_{\text{intake}}^{\text{day}}(0|1) = 0.018$ ,  $p_{\text{intake}}^{\text{day}}(1|1) = 0.982$ , and  $p_{\text{intake}}^{\text{day}}(i|1) = 0 \forall i > 1$ , which yields the following command:  $\square \text{ s}=2 \& (\text{day}=1) \rightarrow 0.018 : (\text{stock}'=\text{stock}) \& (\mathbf{s}'=3) + 0.982 : (\text{stock}'=\text{stock}-1) \& (\mathbf{s}'=3);$ . The probability distribution  $p_{\text{intake}}^{\text{day}}(\cdot|h)$  is estimated for each day  $h \in \{1, \dots, H\}$  of the horizon from the statistics of drug consumption and hospitalization time, using either a DPH distribution (Sect. 3.2) or an HMM (Sect. 3.3).

**Reward.** The policy that minimizes the objective function of Eq. (1) can be derived using the PRISM tool to perform probabilistic model checking of the

property `R"cost"min=?[F "horizonEnd"]`, requiring that the expected cumulated cost of the paths reaching the last day of the horizon is minimum.

**Receding Horizon Control.** The evaluation is repeatedly performed according to the receding horizon paradigm [11]: (i) an MDP model of the stock replenishment problem is synthesized, using data on drug consumption and hospitalization time to estimate transition probabilities; (ii) the MDP is solved to derive the policy that minimizes the overall cost over a finite horizon of  $H$  days; (iii) the first action of the best policy found is applied, ordering the number of drug units suggested for the current day while discarding the subsequent actions; (iv) the horizon is moved forward by one day. In doing so, the MDP model can adapt at runtime to variations of drug consumption and hospitalization time, which may be due to changes in clinical protocols or modifications in prescribed drugs.

### 3.2 Estimating Drug Consumption Probabilities Through a DPH

The probability  $p_{\text{intake}}^{\text{day}}(i|h)$  that  $i$  drug units are consumed by the hospitalized patients during the  $h$ -th day of the horizon can be estimated by fitting the statistic of hospitalization time through an acyclic DPH distribution [9, 19], i.e., the distribution of the time until absorption in a Discrete-Time Markov Chain (DTMC) with a finite number of transient states (termed *phases*), one absorbing state, and no cycle. Let  $\mathbb{D}$  be the obtained DTMC and  $\Sigma = \{\sigma_1, \dots, \sigma_T\}$  be the set of its states. On the one hand, for each state  $\sigma \in \Sigma$ , we derive the probability  $p_{\text{intake}}^{\text{DTMC}}(i, \sigma)$  that  $i$  drug units are consumed by a patient during a single day given that the elapsed hospitalization time of the patient is described by state  $\sigma$  of  $\mathbb{D}$ . On the other hand, for each vector  $\mathbf{n} = \langle n_1, \dots, n_T \rangle \in \mathbb{N}^T$  such that  $\|\mathbf{n}\|_1 = \sum_{t=1}^T n_t \leq P_{\max}$ , we compute the probability  $p_{\text{patients}}(\mathbf{n}, h)$  that, on day  $h$  of the horizon, state  $\sigma_t \in \Sigma$  describes the elapsed hospitalization time of  $n_t$  patients for each  $t \in \{1, \dots, T\}$ . Therefore,  $p_{\text{intake}}^{\text{day}}(i|h)$  can be derived as:

$$p_{\text{intake}}^{\text{day}}(i|h) = \sum_{\substack{\mathbf{n} \in \mathbb{N}^T \\ \|\mathbf{n}\|_1 \leq P_{\max}}} p_{\text{patients}}(\mathbf{n}, h) \sum_{\substack{\mathbf{u} \in \mathbb{N}^{\|\mathbf{n}\|_1} \\ \|\mathbf{u}\|_{\infty} \leq Q_{\max} \\ \|\mathbf{u}\|_1 = i}} \prod_{q=1}^{\|\mathbf{n}\|_1} p_{\text{intake}}^{\text{DTMC}}(\gamma(\mathbf{u}, q), \phi(\mathbf{n}, q)) \quad (3)$$

where  $\mathbf{u} = \langle u_1, \dots, u_{\|\mathbf{n}\|_1} \rangle$  is a vector of integer values representing the number of drug units consumed by each of  $\|\mathbf{n}\|_1$  patients,  $Q_{\max}$  is the maximum number of drug units that a patient may intake,  $\gamma(\mathbf{u}, q)$  is the number of drug units consumed by the  $q$ -th considered patient, and  $\phi(\mathbf{n}, q)$  is the state of  $\mathbb{D}$  that describes the elapsed hospitalization time of the  $q$ -th considered patient. In doing so,  $p_{\text{intake}}^{\text{day}}(i|h)$  is derived by summing, over all feasible allocations of patients to states of  $\mathbb{D}$  (outer summation) and over all feasible combinations of the number of drug units consumed by each patient (inner summation), the product of the probabilities that each patient consumes the specific number of drug units given

the specific state of  $\mathbb{D}$  that describes his/her elapsed hospitalization time (note that drug consumptions by patients are independent events).

**Derivation of  $p_{\text{intake}}^{\text{DTMC}}(i, \sigma)$ .** By the law of total probability, we obtain the following expression for  $p_{\text{intake}}^{\text{DTMC}}(i, \sigma)$ , i.e., the probability that the number  $G$  of drug units consumed by a patient during a single day is equal to  $i$  given that the state  $V$  of  $\mathbb{D}$  that describes the elapsed hospitalization time of the patient is  $\sigma$ :

$$\begin{aligned} p_{\text{intake}}^{\text{DTMC}}(i, \sigma) &:= P\{G = i | V = \sigma\} \\ &= \sum_{l=0}^{K_{\max}} P\{G = i | V = \sigma, K = l\} P\{K = l | V = \sigma\} \end{aligned} \quad (4)$$

where  $K \in \{0, \dots, K_{\max}\}$  is the hospitalization day ( $K_{\max} = 26$  in Fig. 1b).

Given that the number of drug units consumed by a patient does not depend on the state of  $\mathbb{D}$  that describes its elapsed hospitalization time, we obtain that  $P\{G = i | V = \sigma, K = l\} = P\{G = i | K = l\}$ , where  $P\{G = i | K = l\}$  can be easily computed from the hospitalization time statistic as the ratio between the number of times that a patient intakes  $i$  drug units during the  $l$ -th hospitalization day and the number of times that a patient has been hospitalized for at least  $l$  days.

Finally, by the Bayes theorem,  $P\{K = l | V = \sigma\}$  can be expressed as:

$$\begin{aligned} P\{K = l | V = \sigma\} &= \frac{P\{V = \sigma | K = l\} P\{K = l\}}{P\{V = \sigma\}} \\ &= \frac{P\{V = \sigma | K = l\} P\{K = l\}}{\sum_{m=0}^{K_{\max}} P\{V = \sigma | K = m\} P\{K = m\}} \end{aligned} \quad (5)$$

where  $P\{V = \sigma | K = l\}$  can be computed from the one-step transition probability matrix of  $\mathbb{D}$  as the probability of being in state  $\sigma$  after  $l$  transitions conditioned to not having reached the absorbing state through any of the previous  $l - 1$  transitions, while  $P\{K = l\}$  can be computed as the ratio between the number of times that a patient has been hospitalized for at least  $l$  days and the total number of hospitalization days of all patients.

**Derivation of  $p_{\text{patients}}(\mathbf{n}, h)$ .** For each vector  $\mathbf{n} = \langle n_1, \dots, n_T \rangle \in \mathbb{N}^T$  such that  $\|\mathbf{n}\|_1 = \sum_{t=1}^T n_t \leq P_{\max}$  and for each day  $h \in \{1, \dots, H\}$  in the considered horizon, we exploit  $p_{\text{DTMC}}^{\text{day}}(\sigma, h) := P\{V = \sigma | K = h\}$ , i.e., the probability of being in state  $\sigma$  given that the current day of the horizon is  $h$ , to derive  $p_{\text{patients}}(\mathbf{n}, h)$ , i.e., the probability that, on day  $h$  of the horizon, state  $\sigma_t \in \Sigma$  describes the elapsed hospitalization time of  $n_t$  patients for each  $t \in \{1, \dots, T\}$ .

The derivation of  $p_{\text{patients}}(\mathbf{n}, h)$  is performed through an iterative algorithm that takes into account all the possible allocations of patients to states of  $\mathbb{D}$  and the corresponding probability. The overall algorithm is not shown due to space limitations. As an example, Algorithm 1 shows a pseudo-code fragment that computes a quantity used in the evaluation of  $p_{\text{patients}}(\mathbf{n}, h)$ , namely the



**Algorithm 1.** Calculation of  $p_{\text{init}}(\mathbf{n})$ 


---

```

 $p_{\text{tmp}}(\mathbf{n}) := \begin{cases} 1 & \text{if } \mathbf{n} = [0, \dots, 0] \\ 0 & \text{otherwise} \end{cases}$ 
for  $i = 1$  to  $N_{\text{init}}$  do
  for all  $\mathbf{n} = \langle n_1, \dots, n_T \rangle \in \mathbb{N}^T$  such that  $\|\mathbf{n}\|_1 = \sum_{t=1}^T n_t \leq P_{\text{max}}$  do
     $p_{\text{new}}(\mathbf{n}) = \sum_{j=1}^T p_{\text{tmp}}(\langle n_1, \dots, (n_j - 1), \dots, n_T \rangle) \cdot p_{\text{DTMC}}^{\text{day}}(\sigma_j, l_i)$ 
     $p_{\text{tmp}}(\cdot) = p_{\text{new}}(\cdot)$ 
  end for
end for
 $p_{\text{init}}(\cdot) := p_{\text{new}}(\cdot)$ 

```

---

probability  $p_{\text{init}}(\mathbf{n})$  that, for each  $t \in \{1, \dots, T\}$ , state  $\sigma_t \in \Sigma$  describes the elapsed hospitalization time of  $n_t$  of the initial  $N_{\text{init}}$  patients, with  $\langle l_1, \dots, N_{\text{init}} \rangle$  being the vector containing the current hospitalization day of each patient.

### 3.3 Estimating Drug Consumption Probabilities Through an HMM

The probability  $p_{\text{intake}}^{\text{day}}(i | h)$  that  $i$  drug units are consumed by the patients during the  $h$ -th day of the horizon can be estimated by fitting the statistics of hospitalization time and drug consumption through an HMM [21], i.e., a DTMC with unobserved (*hidden*) states, generating an *observation* at each state transition. We use hidden states to account for the hospitalization time of patients and observations to model drug consumption. Since in general an HMM does not have a final absorbing state indicating the end of the hospitalization time, the set of observations is extended into a new set where, for each symbol  $q_j$  representing the consumption of  $j$  drug units by a patient, a new symbol  $q_j^E$  is added modeling not only the drug consumption but also the discharge of the patient from the ward. Given a set of such observations and a selected number of desired hidden states, the *Expectation-Maximization* (EM) algorithm permits to automatically evaluate the HMM parameters that maximize the probability that the considered observation sequences are generated by the model.

Following the notation of Sect. 3.2, let  $\mathbb{D}$  be the DTMC of the HMM and  $\Sigma = \{\sigma_1, \dots, \sigma_T\}$  be the set of its states. For each vector  $\mathbf{n} = \langle n_1, \dots, n_T \rangle \in \mathbb{N}^T$  such that  $\|\mathbf{n}\|_1 = \sum_{t=1}^T n_t \leq P_{\text{max}}$ , the initial state probabilities and the transition probability matrix of the HMM permit to derive  $p_{\text{patients}}(\mathbf{n}, h)$ , i.e., the probability that, on day  $h$  of the horizon, state  $\sigma_t \in \Sigma$  describes the elapsed hospitalization time of  $n_t$  patients for each  $t \in \{1, \dots, T\}$ . The derivation is performed through an iterative algorithm similar to that used for DPHs, which is also not reported due to space limitations. Once  $p_{\text{patients}}(\mathbf{n}, h)$  is known,  $p_{\text{intake}}^{\text{day}}(i | h)$  can be computed according to Eqs. 3–5, as discussed in Sect. 3.2.

Note that the EM algorithm yields the joint probability distribution of drug consumption and discharge time. While leading to better predictions, taking advantage of this joint distribution in the MDP model requires to maintain in

memory the number of consumptions produced in each phase, massively increasing the state space. According to this, we used the marginal distributions of drug consumption and discharge time both to maintain a manageable state space and to obtain a more comparable model with respect to the DPH fitting.

## 4 Experiments

The approach is experimented in combination with the techniques described in Sects. 3.2 and 3.3 for estimating drug consumption probabilities, based on DPH distributions (*DPH approach*) and HMMs (*HMM approach*), respectively. Experiments are performed on the data set introduced in Sect. 2.2, comparing results with those obtained with two standard inventory management policies.

### 4.1 Experimental Setup

The data set is split into a training set of nearly 10 months and a test set of nearly 2 months. The training set is used to perform parameter tuning and hyperparameter optimization, whereas the test set is used to evaluate the performance of the approaches in terms of total cost  $C$  obtained at the end of the test period.

**Baseline Competitors.** As baseline competitors, we consider the inventory policies  $(Q, r)$  and  $(R, T)$  [22], which decide when and how much to order based on the quantity of drugs in the inventory only, neither considering the number of patients in the ward, nor their sojourn times. In addition, both policies are subject to a periodic review constraint, being able to order only once per day. Specifically, the  $(Q, r)$  policy, also termed *reorder point, fixed quantity* policy, issues an order of a predefined fixed quantity  $Q$  when the number of stocked drugs falls below a threshold  $r$ . Conversely, the  $(R, T)$  policy, also termed *order-up-to*  $(R, T)$  policy, issues an order equal to the minimum quantity required to get back over the threshold  $R$  at each periodic interval  $T$  (each day in our case).

For the two baselines, we select the hyperparameters producing the minimum cost  $C$  on the training set. As for the values of  $r$  and  $R$ , we test all integers between 1 and 15 (in the data set, the maximum number of drug units consumed in a single day is 12). For  $(Q, r)$ , for all the possible values of  $r$ , the fixed quantity to be reordered  $Q$  is selected among the multiples of  $\Delta \in \{1, \dots, 20\}$ .

**DPH Approach.** In the DPH approach, the hyperparameter to fit is the number of phases  $\Phi$ , selected using a cross-validation approach. Over 10 rounds, the training set is split into a reduced training set of 9 months, and a validation set of 1 month. We let  $\Phi \in \{1, \dots, 6\}$ , which is a reasonable choice since the mean sojourn time of patients is 2.72 days and a higher number of phases would result in a much higher complexity in the evaluation of  $p_{\text{intake}}^{\text{day}}(i|h)$ . In each round  $r$  of the cross-validation, for each possible value of  $\Phi$ , we fit parameters on the reduced training set with the *PhFit* tool [13], and we evaluate the cost on the validation set. Such cost is evaluated using a horizon of  $H = 5$  days, with the

possibility to order between 0 and 20 drug units (that is, with 21 nondeterministic choices in the *MDP*): the number of phases is thus chosen as the value that minimizes the average cost over the 10 rounds. We remark that *PhFit* follows a gradient-based approach, which can stop in local optima. Therefore, a multi-start methodology is employed, by repeating parameter fitting a number of times with different random starting points, and by selecting the distribution that best fits samples.

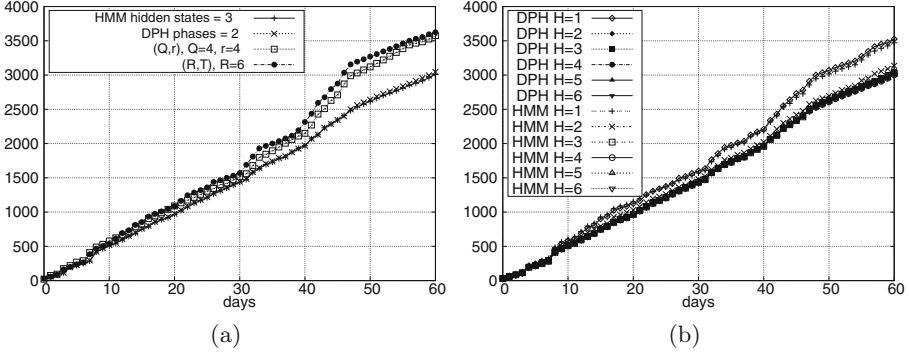
**HMM Approach.** The hyperparameter to fit in HMMs is the number of hidden states, which is selected in  $\{2, \dots, 7\}$  to guarantee a fair the comparison with the DPH approach (the number of phases of a DPH does not include the final absorbing state, so that the DPH approach with  $\phi$  phases compares with the HMM approach with  $\phi+1$  states). Given that also HMMs suffer from the problem of local minima, also in this case we exploit a multi-start methodology. The Python library *HMMlearn* [3] is used for HMM training.

Both in the DPH approach and in the HMM approach, we use the Java Apache Velocity template engine [2] for the automatic construction of the MDP model at runtime, and the Prism API [17] for the synthesis of the optimal policy.

## 4.2 Experimental Results

All experiments were performed on a 64-bit Intel I7 4690k @4GHz CPU with 16 GB RAM. The selected hyperparameters are the following: a threshold  $r = 4$  and a fixed quantity  $Q = 4$  for the  $(Q, r)$  policy; a threshold  $R = 6$  for the  $(R, T)$  policy; equivalently 1 or more phases for the DPH; 3 hidden states for the HMM. For the DPH approach, since sojourn times are almost geometrically distributed when excluding a null stay (as shown in Fig. 1b), a single phase is already sufficient to approximate the distribution with high precision. For comparison with the HMM approach, the number of phases selected for the DPH approach is 2. Training required 14ms both for the  $(Q, r)$  and the  $(R, T)$  policies. The DPH and HMM approaches required 116 s and 151 s, respectively, most of which spent in the evaluation of  $p_{\text{intake}}^{\text{day}}(i, h)$ , with PRISM requiring  $<1$  s.

Figure 2a shows day-by-day cumulated costs on the test set for all approaches, considering a horizon of  $H = 5$  days for the MDP and assuming the following costs (so that the ratio between them is realistic for the context of use): standard cost  $c_d = 10$ , urgent cost  $c_u = 40$ , and stocking cost  $c_s = 5$ . Results show that the total cost of the HMM and DPH approaches is considerably lower than that of the inventory level policies, proving the effectiveness of utilizing information on the ward state. In particular, all approaches initially have a comparable cumulated cost, but, after nearly 40 days, costs start diverging due to a sudden increase in the number of patients in the ward and the consequent higher drug consumption. Since the inventory level approaches do not consider this information, they end up requiring urgent orders to meet the increasing demand. The HMM costs are lower than the DPH costs by around 1%, but the HMM approach requires a running time (29s) slightly larger than that of the DPH approach (22s). We

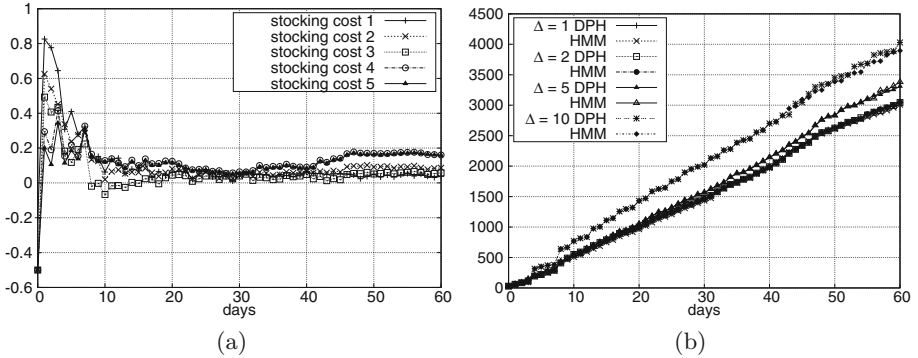


**Fig. 2.** (a) Cumulative cost on the test set. (b) Cumulative cost on the test set for the DPH approach with 2 phases and the HMM approach with 3 hidden states.

remark that the time required by the DPH approach could be sensibly lowered by setting the number of phases equal to 1, since there are no significant differences in results, while the state space would be much smaller.

Figure 2b shows how the cumulated cost varies with the considered horizon  $H$  for the DPH approach with 2 phases and the HMM approach with 3 hidden states. Specifically, the cumulative cost reduces as  $H$  increases until  $H = 3$ , with no significant variations for  $H \geq 3$ : reasonably, a horizon of 3 days turns out to be sufficient to achieve good performance due to the fact that the mean hospitalization time is 2.72 days, as discussed in Sect. 2.2. The complexity of the approach increases with  $H$ : specifically, for  $H = 1, 2, 3, 4, 5$ , the running times of the DPH approach on the test set are 3 s, 6 s, 11 s, 16 s, and 21 s, respectively, while for the HMM approach they are 3 s, 8 s, 14 s, 20 s, and 26 s, respectively.

Another factor with significant impact on results is the stocking cost  $c_s$ . If it is low, all approaches tend to maintain a high number of drugs as safety stock, to avoid to issue urgent orders. As the stocking cost increases, it becomes more relevant to maintain a trade-off between having a low number of drug units in stock and avoiding urgent orders. According to this, the advantage of the HMM approach and the DPH approach over the inventory level policies increases with the stocking cost. In Fig. 3a, we compare the cost of the HMM approach with 3 hidden states and  $H = 5$  with the cost of the  $(Q, r)$  policy, which are the best performing methods for the two different classes of approaches, varying the stocking cost  $c_s$  between 1 and 5. In particular, the hyperparameters of both approaches have been re-computed for different stocking costs. While the number of hidden states of the HMM remains equal to 3, the  $(Q, r)$  thresholds vary: for  $c_s$  increasing from 1 to 5 we have  $(Q, r) = (6, 5), (5, 4), (5, 4), (4, 4)$  and  $(4, 4)$ , respectively. The relation between the costs of the HMM approach and the  $(Q, r)$  policy is expressed through  $\rho(d) := (C_{(Q,r)}(d) - C_{\text{HMM}}(d))/C_{(Q,r)}(d)$  in the y-axis, where  $C_{\text{HMM}}(d)$  is the cumulative HMM cost until day  $d$  and  $C_{(Q,r)}(d)$  is the cost for  $(Q, r)$ . After the instability of the first few days due to



**Fig. 3.** (a) Profit  $\rho(d)$  of the HMM approach over the  $(Q, r)$  policy as a function of days. (b) Cumulative cost on the test set for the DPH approach with 2 phases and the HMM approach with 3 hidden states.

delayed purchases, the advantage of the HMM approach increases with higher stocking costs: HMM saves almost 20% of the cost when  $c_s = 5$ .

Figure 3b shows the cumulative costs of the HMM approach with 3 hidden states and the DPH approach with 2 phases varying the granularity  $\Delta$  of the admissible orders in  $\{1, 2, 5, 10\}$  (and  $H = 5$ ): as the granularity decreases, two approaches tune orders with more precision, while reducing the overall cost. The best performances are in fact achieved with  $\Delta = 1$  and  $\Delta = 2$ , with only slightly differences. With  $\Delta = 10$ , the performance drops by more than 30%. We can therefore conclude that flexibility in order granularity plays an important role in reducing costs in the drug restocking problem.

## 5 Conclusions

We formulate the problem of stock management in a hospital ward as a Markov Decision Process (MDP), where nondeterministic actions model the choice of ordering drug units, while probabilistic transitions represent drug consumption. The model is iteratively solved at run-time, applying only the first action of the best policy found and moving the horizon forward by one day. Experiments on a real-world case study, with data collected from a ward of a public Italian hospital, show the convenience of the approach with respect to standard baselines.

The problem formulation could be extended in various aspects, for instance by considering a fixed cost for orders or a different purchasing cost depending on the supplier. It would be interesting also to compare the proposed approach with reorder point policies that issue a variable-size order whenever stock drops below a given threshold, so as to bring it up to the pre-determined level [16]. A notable challenge would be accounting for dependencies in the use of different drugs, which would require joint optimization of the orders of multiple drugs. To this end, integration with efficient solution techniques based on Model Predictive Control (MPC) such as those used in [14] comprises a direction to explore.

## References

1. An Architectural Blueprint for Autonomic Computing. Technical report, IBM (2006)
2. Apache velocity (2018). <http://velocity.apache.org/>
3. HMMLearn python library (2018). <http://hmmlearn.readthedocs.io/en/latest/>
4. Project LINFA (2018). <http://www.linfasystem.it/>
5. Adachi, Y., Nose, T., Kuriyama, S.: Optimal inventory control policy subject to different selling prices of perishable commodities. *Int. J. Prod. Econ.* **60–61**, 389–394 (1999)
6. Ahiska, S.S., Appaji, S.R., King, R.E., Warsing, D.P.: A Markov decision process-based policy characterization approach for a stochastic inventory control problem with unreliable sourcing. *Int. J. Prod. Econ.* **144**(2), 485–496 (2013)
7. Alessandri, A., Gaggero, M., Tonelli, F.: Min-max and predictive control for the management of distribution in supply chains. *IEEE Trans. Control Syst. Technol.* **19**(5), 1075–1089 (2011)
8. Blair, G., Bencomo, N., France, R.B.: Models@ run. time. *Computer* **42**, 10 (2009)
9. Bobbio, A., Horváth, A., Scarpa, M., Telek, M.: Acyclic discrete phase type distributions: properties and a parameter estimation algorithm. *Perform. Eval.* **54**(1), 1–32 (2003)
10. de Vries, J.: The shaping of inventory systems in health services: a stakeholder analysis. *Int. J. Prod. Econ.* **133**(1), 60–69 (2011)
11. García, C.E., Pretti, D.M., Morari, M.: model predictive control: theory and practice—a survey. *Automatica* **25**(3), 335–348 (1989)
12. Giannoccaro, I., Pontrandolfo, P.: Inventory management in supply chains: a reinforcement learning approach. *Int. J. Prod. Econ.* **78**(2), 153–161 (2002)
13. Horváth, A., Telek, M.: PhFit: a general phase-type fitting tool. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) *TOOLS 2002*. LNCS, vol. 2324, pp. 82–91. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-46029-2\\_5](https://doi.org/10.1007/3-540-46029-2_5)
14. Incerto, E., Tribastone, M., Trubiani, C.: Software performance self-adaptation through efficient model predictive control. In: *ASE, Piscataway, NJ, USA*, pp. 485–496. IEEE Press (2017)
15. Jurado, I., et al.: Stock management in hospital pharmacy using chance-constrained model predictive control. *Comput. Biol. Med.* **72**, 248–255 (2016)
16. Kelle, P., Woosley, J., Schneider, H.: Pharmaceutical supply chain specifics and inventory solutions for a hospital case. *Oper. Res. Health Care* **1**(2), 54–63 (2012)
17. Kwiatkowska, M., Norman, G., Parker, D.: PRISM: probabilistic symbolic model checker. In: Field, T., Harrison, P.G., Bradley, J., Harder, U. (eds.) *TOOLS 2002*. LNCS, vol. 2324, pp. 200–204. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-46029-2\\_13](https://doi.org/10.1007/3-540-46029-2_13)
18. Moreno, G.A., Cámara, J., Garlan, D., Schmerl, B.: Proactive self-adaptation under uncertainty: a probabilistic model checking approach. In: *Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015*, pp. 1–12. ACM (2015)
19. Neuts, M.F.: *Probability Distributions of Phase Type*. Purdue University, Baltimore (1974). Department of Statistics
20. Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York (2014)
21. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. *Proc. IEEE* **77**(2), 257–286 (1989)

22. Tempelmeier, H.: Inventory Management in Supply Networks Problems, Models, Solutions. Books on Demand, Norderstedt (2011)
23. Volland, J., Fügener, A., Schoenfelder, J., Brunner, J.O.: Material logistics in hospitals: a literature review. *Omega* **69**, 82–101 (2017)
24. White, D.J.: A survey of applications of markov decision processes. *J. Oper. Res. Soc.* **44**(11), 1073–1096 (1993)



# Guaranteed Error Bounds on Approximate Model Abstractions Through Reachability Analysis

Luca Cardelli<sup>1,2</sup>, Mirco Tribastone<sup>3</sup>, Max Tschaikowski<sup>3</sup>,  
and Andrea Vandin<sup>4</sup>(✉)

<sup>1</sup> Microsoft Research, Cambridge, UK  
luca@microsoft.com

<sup>2</sup> University of Oxford, Oxford, UK

<sup>3</sup> IMT School for Advanced Studies Lucca, Lucca, Italy  
mirco.tribastone@gmail.com, max.tschaikowski@imtlucca.it

<sup>4</sup> DTU Compute, Kongens Lyngby, Denmark  
anvan@dtu.dk

**Abstract.** It is well known that exact notions of model abstraction and reduction for dynamical systems may not be robust enough in practice because they are highly sensitive to the specific choice of parameters. In this paper we consider this problem for nonlinear ordinary differential equations (ODEs) with polynomial derivatives. We introduce approximate differential equivalence as a more permissive variant of a recently developed exact counterpart, allowing ODE variables to be related even when they are governed by nearby derivatives. We develop algorithms to (i) compute the largest approximate differential equivalence; (ii) construct an approximate quotient model from the original one via an appropriate parameter perturbation; and (iii) provide a formal certificate on the quality of the approximation as an error bound, computed as an over-approximation of the reachable set of the perturbed model. Finally, we apply approximate differential equivalences to study the effect of parametric tolerances in models of symmetric electric circuits.

## 1 Introduction

Ordinary differential equations (ODEs) are a prominent model of dynamical systems across many branches of science and engineering, and have enjoyed increasing popularity in computer science, for instance, in computational systems biology [4, 13, 30], as an approximation to large-scale Markov models and as the laws of continuous motion in hybrid systems [7]. This has motivated techniques for the *comparison* and *minimization* of ODEs based on behavioral relations, along the lines of other foundational quantitative models of computation, e.g. [25]. Here we consider *differential equivalence* [11], recently developed as an equivalence over ODE variables yielding a quotient that preserves the dynamics of the original one. However, differential equivalences (reviewed in Sect. 2) are exact, hence highly sensitive to parameter values and initial conditions. This may hinder their practical



usability in some applications domains, for instance due to parameter uncertainty arising from finite-precision measurements in biology or the tolerances of electric components in electrical engineering.

Our objective is to develop approximate variants of differential equivalence (in Sect. 3). We study models with derivatives given by multivariate polynomials (over the ODE variables) of any degree, thus restricting the scope of [11]. However we remark that this is still quite a generous class since it includes chemical reaction networks (CRNs) with mass-action kinetics and linear/affine systems, thus covering, e.g., continuous-time Markov chains through their Kolmogorov equations.

Considering polynomial derivatives allows us to introduce a notion of equivalence that just concerns the ODE “syntax” (while the nonlinear class of ODEs of [11] required symbolic SMT-based checks). Our main idea is to consider a threshold parameter  $\varepsilon \geq 0$ , which intuitively captures perturbations in polynomial coefficients. This allows relating ODE variables that would be distinct otherwise. Like in other established approaches such as behavioral pseudometrics (e.g., [1, 9]),  $\varepsilon = 0$  corresponds to an exact differential equivalence of [11]. In addition to defining criteria for approximate differential equivalences, we provide an algorithm for obtaining the largest one. This is done via partition refinement, computing the coarsest refinement of a given initial partition of ODE variables for a given “structural” tolerance  $\varepsilon$ .

A quotient ODE system can be constructed from a *reference model*, obtained through a perturbation of the coefficients of the original model which makes the given approximate differential equivalence an exact one. By considering a metric (the Euclidean norm) to measure the degree of perturbation, the reference model is the one which minimizes such perturbation. This can be done efficiently by solving an optimization problem which runs polynomially with the size of the ODE system [22]. This approach is analogous to optimal approximate lumping for Markov chains (e.g., [16]), although our theory can be applied to other choices of reference models.

The bound of the error produced by the reference model with respect to the original system can be computed by studying the reachable set of the reference model from an uncertain set of initial conditions that covers the applied perturbation. Since the reference model subsumes any behavior of the quotient, the bound formally relates the quotient model to the original model. Section 4 presents a bound which relies on a linearization of the reference model (which can be efficiently computed in the case of polynomial ODE systems). First, we bound the reachable set of the linearized model using closed form solutions, similarly to [24]. Then, we provide a conservative condition (i.e., an over-approximation) that ensures that the linearized model describes the original nonlinear behavior dynamics well. Our bound is given in terms of an  $\varepsilon$ - $\delta$  argument (similar in spirit to the ones routinely used in calculus). Informally, it states the following: for any choice of the structural tolerance  $\varepsilon$ , there exist a degree of perturbation  $\delta$  and an *amplifier*  $\lambda$  such that, for any ODE system obtained by applying a perturbation

to the reference model of at most  $\delta$ , at all time points the difference between the solution of the reference model and the perturbed one is at most  $\lambda$  times the perturbation.

Being based on a linearization, it is perhaps not surprising that the as-computed  $\delta$  will account only for *small* perturbations of the parameters. Yet numerical experiments in Sect. 5 show that these can be enough to explain quasi-symmetric behavior due to parametric tolerances in components of real electric network designs [31]. By comparing to over-approximation techniques supported by state-of-the-art tools C2E2 [17], CORA [2,3] and Flow\* [12], we show that our bounding technique can complement them in that it can scale to larger systems while being more conservative in the size of the initial uncertain set that it supports.

*Further Related Work.* Differential equivalence is promising when the ODE system is composed of several identical subsystems that depend on some common context [33]. It is related but not comparable to bisimulation for differential systems [18,20] since it partitions ODE variables rather than the state space. Likewise, it complements [8] that captures nonlinear relations between ODE variables but does not enjoy a polynomial time algorithm like [10].

A classic approximation approach relies on Lyapunov-like functions [15,27] known from stability theory of ODEs. However, for nonlinear systems the automatic computation of Lyapunov-like functions remains a challenging task. Restricting to special classes of Lyapunov-like functions (e.g., sum-of-squares polynomials [18]) leads to efficient construction algorithms which may provide tight bounds, but existence is not guaranteed. On the other hand, approximations with differential inequalities [34] can be computed efficiently, but may be loose. Abstraction, supported by CORA and Flow\*, locally approximates the nonlinear model by a multivariate polynomial or an affine system, see [5,12] and references therein. Similarly in spirit we linearize across a reference trajectory. A closer approach to ours is discussed in [17] and supported by the corresponding tool C2E2. It combines local Lyapunov-like functions and techniques based on sensitivity analysis [24]. Our bound is however different because the nonlinear part is bounded analytically by restricting to polynomial derivatives.

More in general, research on approximate quotients of ODE systems spans many disciplines. In chemistry, it can be traced back to Kuo and Wei [23]. They studied monomolecular reaction networks, which give rise to affine ODE systems. The approximation consists in *nearly exact lumping*, i.e., a linear transformation of the state space that would be exact up to a perturbation of the parameters (hence we are similar in spirit). The approximation, however, only applies when the transition matrix underlying the linear system is diagonalizable. Li and Rabitz extend approximate lumping to general CRNs [26], but an explicit error bound is not given. In a similar vein, approximate quotients in ecology have been studied from the point of view of finding a reduced ODE system whose derivatives are as close as possible (in norm) to the derivatives of the original ODE system, where the 0-distance induce the exact quotient [21]. The justification that variables underlying similar ODEs have nearby solutions

is grounded on Gronwall's inequality which is also at the basis of more recent quotient constructions [19], which however are not algorithmic, unlike in this paper.

*Notation and Basic Definitions.* We denote the infinity norm  $\|\cdot\|_\infty$  by  $\|\cdot\|$ , while  $\|\cdot\|_2$  is the Euclidian norm. Whenever convenient, for a given partition of variables  $\mathcal{H}$ , we write  $H = \{x_{H,1}, \dots, x_{H,|H|}\}$  for any  $H \in \mathcal{H}$ . We denote by  $\psi[t/s]$  the term that arises by replacing each occurrence of  $t$  in  $\psi$  by  $s$ . Let  $S$  be a set, and  $\mathcal{H}_1, \mathcal{H}_2$  two partitions of  $S$ . Then,  $\mathcal{H}_1$  is a *refinement* of  $\mathcal{H}_2$  if for any block  $H_1 \in \mathcal{H}_1$  there exists a block  $H_2 \in \mathcal{H}_2$  such that  $H_1 \subseteq H_2$ . For any partition  $\mathcal{H}$  of  $S$ , let  $\sim_{\mathcal{H}}$  denote the unique equivalence relation with  $\mathcal{H} = S/\sim_{\mathcal{H}}$ . The transitive closure of a relation  $\sim$  is denoted by  $\sim^*$ .

## 2 Background

Throughout the paper we consider a polynomial initial value problem (PIVP) over the set of ODE variables  $\mathcal{S} = \{x_1, \dots, x_n\}$ . It is defined by the ODEs  $\dot{x}_i = q_i$ ,  $1 \leq i \leq n$ , where  $q_i$  is a multivariate polynomial over  $\mathcal{S}$ . The initial condition of the PIVP is given by  $\sigma : \mathcal{S} \rightarrow \mathbb{R}$ ;  $x_i(t)$  denotes the unique solution for variable  $x_i$  at time point  $t$  starting from  $x_i(0) = \sigma(x_i)$ . We consider PIVPs that do not exhibit finite explosion times, i.e., whose solutions do not have a singularity at any finite point in time. This property is shared by the vast majority of practical models and can be efficiently checked via numerical ODE solvers.

A polynomial  $q_i$  is given in the normal form if each monomial  $x^\alpha \equiv \prod_{x_i \in \mathcal{S}} x_i^{\alpha_{x_i}}$ , where  $\alpha \in \mathbb{N}_0^{\mathcal{S}}$  is a multi-index, appears in  $q_i$  at most once. The normal form of a polynomial  $q_i$  is denoted by  $\mathcal{N}(q_i)$ . Without loss of generality, we assume that the polynomials  $q_i$  of a PIVP  $\dot{x} = q(x)$ , where  $q = (q_i)_{x_i \in \mathcal{S}}$ , are given in normal form. For a polynomial  $q_i$  in normal form with variables in  $\mathcal{S}$ , let  $c(q_i, x^\alpha)$  denote the coefficient of the monomial  $x^\alpha$ , where  $\alpha \in \mathbb{N}_0^{\mathcal{S}}$ .

*Example 1.* We use the following ODE system, with variables  $\mathcal{S} = \{x_1, x_2, x_3\}$ , as a running example.

$$\dot{x}_1 = -4.00x_1 + x_2 + x_3 \quad \dot{x}_2 = 1.99x_1 - x_2 \quad \dot{x}_3 = 2.01x_1 - x_3 \quad (1)$$

In [11] two variants of differential equivalence were introduced for IDOL, a class of nonlinear ODE systems covering derivatives more general than polynomials. Here we find it convenient to restate them for a PIVP. (The proofs for this correspondence are straightforward hence we omit them).

We begin with *backward differential equivalence* (BDE), which relates variables that have the same solutions at all time points. The definition of BDE for PIVP makes pairwise comparisons between the coefficients of any two variables in the same equivalence class.

**Definition 1 (BDE).** Fix a PIVP, a partition  $\mathcal{H}$  of  $\mathcal{S}$  and write  $x_i \sim_{\mathcal{H}}^B x_j$  if all coefficients of the following polynomial are zero,

$$\wp_{i,j}^{\mathcal{H}} := (q_i - q_j)[x_{H',1}/x_{H'}, \dots, x_{H',|H'|}/x_{H'} : H' \in \mathcal{H}]$$

i.e., when

$$\sum_{\alpha \in \mathbb{N}_0^S} |c(\wp_{i,j}^{\mathcal{H}}, x^\alpha)| = 0. \tag{2}$$

A partition  $\mathcal{H}$  is a BDE if  $\mathcal{H} = \mathcal{S}/(\sim_{\mathcal{H}}^{B^*} \cap \sim_{\mathcal{H}})$ .

Essentially, establishing that a candidate partition is a BDE consists in comparing the coefficients of the monomials of the ODEs of related variables, up to the natural equivalence class induced on monomials by the equivalence relation through  $\wp_{i,j}^{\mathcal{H}}$ —for instance, the partition  $\{\{x_1\}, \{x_2, x_3\}\}$  will equate the monomials  $x_1x_2$  and  $x_1x_3$ . Then, for any two variables in the same block it must hold that the differences between the coefficients of the same monomials (modulo the induced equivalence class) are zero.

*Example 2.* In our running example let us consider the partition of variables  $\mathcal{H} = \{H_1, H_2\}$ , with  $H_1 = \{x_1\}$  and  $H_2 = \{x_2, x_3\}$ . Then  $\mathcal{H}$  is not a BDE because

$$\wp_{2,3}^{\mathcal{H}} = -0.02x_1 \quad \text{and} \quad c(\wp_{2,3}^{\mathcal{H}}, x_1) = -0.02 \neq 0.$$

*Forward differential equivalence (FDE)* identifies a partition that induces a quotient ODE that tracks sums of variables in each equivalence class. For instance, for any initial condition we have that  $\{\{x_1\}, \{x_2, x_3\}\}$  is an FDE for (1) because we can find an ODE system for  $x_2 + x_3$ :

$$\dot{x}_1 = -4.00x_1 + (x_2 + x_3) \quad (x_2 + x_3) = 4.00x_1 - (x_2 + x_3)$$

The change of variable  $x_{23} = x_2 + x_3$  gives us the quotient ODE  $\dot{x}_1 = -4.00x_1 + x_{23}$ ,  $\dot{x}_{23} = 4.00x_1 - x_{23}$ . From this we conclude that the solution satisfies  $x_{23}(t) = x_2(t) + x_3(t)$  for all time  $t$  if this holds for the initial condition, i.e.,  $x_{23}(0) = x_2(0) + x_3(0)$ .

For a PIVP, FDE can be checked by requiring that the evaluation of the polynomial that represents the quotient derivative for an equivalence class is invariant with respect to a redistribution of the values of any two variables within that equivalence class.

**Definition 2 (FDE).** Fix a PIVP, a partition  $\mathcal{H}$  of  $\mathcal{S}$  and write  $x_i \sim_{\mathcal{H}}^F x_j$  if all coefficients of the polynomial  $\sum_{H \in \mathcal{H}} \wp_{i,j}^H$  are zero, where

$$\wp_{i,j}^H := \sum_{x_k \in H} q_k - \sum_{x_k \in H} q_k [x_i/s(x_i + x_j), x_j/(1-s)(x_i + x_j)]$$

That is, when

$$\sum_{k=1}^m \sum_{\alpha \in \mathbb{N}_0^{S \cup \{s\}}} |c(\wp_{i,j}^{H_k}, x^\alpha)| = 0. \tag{3}$$

$\mathcal{H} = \{H_1, \dots, H_m\}$  is an FDE when  $\mathcal{H} = \mathcal{S}/(\sim_{\mathcal{H}}^{F^*} \cap \sim_{\mathcal{H}})$ .

---

**Algorithm 1.** Template partition refinement algorithm for the computation of the coarsest  $\varepsilon$ -FDE/ $\varepsilon$ -BDE partition that refines a given initial partition  $\mathcal{G}$ .

---

**Require:** A PIVP over variables  $\mathcal{S}$ , a partition  $\mathcal{G}$  of  $\mathcal{S}$ , a threshold  $\varepsilon \geq 0$ , and a mode  $\chi \in \{F, B\}$ .

```

 $\mathcal{H} \leftarrow \mathcal{G}$ 
while true do
     $\mathcal{H}' \leftarrow \mathcal{S} / (\sim_{\mathcal{H}, \varepsilon}^{\chi*} \cap \sim_{\mathcal{H}})$ 
    if  $\mathcal{H}' = \mathcal{H}$  then
        return  $\mathcal{H}$ 
    else
         $\mathcal{H} \leftarrow \mathcal{H}'$ 
    end if
end while
    
```

---

### 3 Approximate Differential Equivalences

*Definitions.* Approximate differential equivalence relaxes the equality conditions (2)–(3) of Definitions 1 and 2 to inequalities with respect to a tolerance level  $\varepsilon$ .

**Definition 3 (Approximate BDE).** Fix a PIVP, a partition  $\mathcal{H} = \{H_1, \dots, H_m\}$  of  $\mathcal{S}$ , and  $\varepsilon \geq 0$ . We write  $x_i \sim_{\mathcal{H}, \varepsilon}^B x_j$  if  $\sum_{\alpha \in \mathbb{N}_0^s} |c(\wp_{i,j}^{\mathcal{H}}, x^\alpha)| \leq \varepsilon$ , where  $\wp_{i,j}^{\mathcal{H}}$  is as in Definition 1. A partition  $\mathcal{H}$  is an  $\varepsilon$ -BDE if  $\mathcal{H} = \mathcal{S} / (\sim_{\mathcal{H}, \varepsilon}^B \cap \sim_{\mathcal{H}})$ .

**Definition 4 (Approximate FDE).** Fix a PIVP, a partition  $\mathcal{H} = \{H_1, \dots, H_m\}$  of  $\mathcal{S}$ , and  $\varepsilon \geq 0$ . We write  $x_i \sim_{\mathcal{H}, \varepsilon}^F x_j$  if  $\sum_{k=1}^m \sum_{\alpha \in \mathbb{N}_0^{S \cup \{s\}}} |c(\wp_{i,j}^{H_k}, x^\alpha)| \leq \varepsilon$ , where  $\wp_{i,j}^{H_k}$  is as in Definition 2. A partition  $\mathcal{H}$  is an  $\varepsilon$ -FDE when  $\mathcal{H} = \mathcal{S} / (\sim_{\mathcal{H}, \varepsilon}^F \cap \sim_{\mathcal{H}})$ .

Setting  $\varepsilon = 0$  recovers the exact counterparts in both cases. That is,  $\mathcal{H}$  is an BDE (resp., FDE) partition if and only if  $\mathcal{H}$  is a 0-BDE (resp., 0-FDE) partition. The two approximate differential equivalences are not comparable since their exact counterparts are not [11]. Since these two notions have similar structure in the rest of this paper we will illustrate only approximate BDE using small examples. Instead, both notions will be discussed in more detail for the numerical evaluation of Sect. 5.

*Example 3.* Let us consider our running example (1). Then, the partition  $\{\{x_1\}, \{x_2, x_3\}\}$  is a 0.02-BDE partition, as can be easily seen from Example 2.

The next two results show the existence of largest approximate differential equivalences and of a partition-refinement algorithm to compute it.

**Theorem 1.** Fix a PIVP, a partition  $\mathcal{G}$  of  $\mathcal{S}$ , and  $\varepsilon \geq 0$ . Then, there exists a unique coarsest  $\varepsilon$ -FDE ( $\varepsilon$ -BDE) partition refining  $\mathcal{G}$ .

**Theorem 2.** *Fix a PIVP, a partition  $\mathcal{G}$  of  $\mathcal{S}$ , and  $\varepsilon \geq 0$ . Then, Algorithm 1 computes the coarsest  $\varepsilon$ -FDE ( $\varepsilon$ -BDE) that refines  $\mathcal{G}$  if  $\chi = F$  ( $\chi = B$ ).*

We now study how efficiently the conditions for approximate differential equivalence can be computed. The notions are defined with respect to the coefficients of the polynomials  $\wp_{i,j}^H$  and  $\wp_{i,j}^{\mathcal{H}}$  and thus require the computation of their normalization. In the case of  $\varepsilon$ -FDE, this yields exponential complexity due to term replacement. To see this, consider for instance the PIVP  $\dot{x}_1 = x_2^k, \dot{x}_2 = x_1^k$ , for some  $k > 0$ . Then, for  $\mathcal{H} = \{\{x_1, x_2\}\}$ , the term  $q_1[x_1/s(x_1 + x_2), x_2/(1-s)(x_1 + x_2)]$  will be of size  $\mathcal{O}(2^k)$ . This stands in stark contrast to  $\varepsilon$ -BDE, where the conditions involve a difference between polynomial terms with no term rewritings. This discussion can be formalized as follows.

**Theorem 3.** *There exists a polynomial  $\Pi$  such that, under the assumptions of Theorem 2, the number of steps done by Algorithm 1 is  $\mathcal{O}(\Pi(2^d \cdot p))$  if  $\chi = F$  and  $\mathcal{O}(\Pi(p))$  if  $\chi = B$ , respectively, where  $d$  is the maximum degree of the polynomial and  $p$  is the number of monomials present in the PIVP.*

In practice,  $d$  is usually not large. For example, mass-action CRNs feature ODEs with degree-two polynomials because in nature at most two species interact in every reaction. An experimental comparison between the reduction runtimes of  $\varepsilon$ -FDE and  $\varepsilon$ -BDE will be presented in Sect. 5. We also remark that since 0-FDE/BDE coincides with FDE/BDE, the above result provides a complexity bound for a subclass of ODE systems considered in [11].

Other computational considerations motivate the choice of the definitions of approximate differential equivalence given in this paper. Another natural definition could have involved the computation of the maximal distance between derivatives “semantically”, i.e., under all possible evaluations within a given domain of interest. For example, consider the PIVP  $\dot{x}_1 = x_1^3 - x_2, \dot{x}_2 = x_1 - x_2^3$ . Establishing that  $\{\{x_1, x_2\}\}$  is an  $\varepsilon$ -BDE would require checking that the difference between the derivatives satisfies

$$|\dot{x}_1 - \dot{x}_2| = |x_1^3 - x_1 + x_2^3 - x_2| \leq \varepsilon, \text{ for all } 0 \leq x_1, x_2 \leq C \quad (4)$$

for some finite  $C > 0$  that represents some bounded domain where the trajectories are assumed to live. Since this example shows that this question is in general equivalent to solving a nonconvex optimization problem, we infer that the problem is NP-hard [29].

However it can be easily shown that our approximate differential equivalence, defined through the coefficients of the polynomials, corresponds to checks such as (4) in the following sense: If a partition  $\mathcal{H}$  satisfies constraints similar to (4) with respect to some  $\varepsilon > 0$ , then there exists an  $\varepsilon' > 0$  such that  $\mathcal{H}$  is an  $\varepsilon'$ -FDE/BDE, and vice versa. The basic idea is to observe that a polynomial is the zero function if and only if its coefficients are all zero.

Finally, we remark that our structural/syntactic criteria can be used for PIVPs only. It is the lack of analogous conditions in the case of more general functions like minima or roots which prevents our approximate differential equivalences to be extended in a straightforward way to the full class of nonlinear ODEs of [11].

*Reference PIVP.* Given a partition of variables that represents an approximate differential equivalence, we construct a *reference PIVP* by finding a “perturbation” of the original PIVP—i.e., a modification of the initial condition  $\sigma$  and the coefficients present in  $q_1, \dots, q_n$ —which ensures that very partition becomes an exact differential equivalence. On this reference PIVP one can use the quotienting algorithms for FDE/BDE developed in [11] (and not restated here formally for brevity). Therefore, the as-obtained quotient represents an approximate reduction of the original PIVP.

We obtain the desired perturbation by treating the original initial conditions and polynomial coefficients uniformly as initial conditions on an *extended PIVP* where every coefficient is parameterized and turned into a new ODE variable.

**Definition 5.** *The parameterization of a polynomial  $q_i$  in normal form with variables  $\mathcal{S}$  is denoted by  $\hat{q}_i$  and arises from  $q_i$  by replacing, for each  $\alpha \in \mathbb{N}_0^S$ , the constant  $c(q_i, x^\alpha)$  with the parameter  $\mathfrak{c}(\hat{q}_i, x^\alpha)$ .*

*Example 4.* The polynomials  $q_2 = 1.99x_1 - x_2$  and  $q_3 = 2.01x_1 - x_3$  from Example 1 give rise to the parameterized polynomials  $\hat{q}_2 = \mathfrak{c}(\hat{q}_2, x_1)x_1 + \mathfrak{c}(\hat{q}_2, x_2)x_2$  and  $\hat{q}_3 = \mathfrak{c}(\hat{q}_3, x_1)x_1 + \mathfrak{c}(\hat{q}_3, x_3)x_3$ , respectively.

**Definition 6 (Extended PIVP).** *For a PIVP  $\mathcal{P}$  with variables  $\mathcal{S}$ , set  $\Theta = \{\mathfrak{c}(\hat{q}_i, x^\alpha) \mid 1 \leq i \leq n, \alpha \in \mathbb{N}_0^S\}$ . Its extended version  $\hat{\mathcal{P}}$  has variables  $\mathcal{S} \cup \Theta$  and is given by  $\dot{x}_i = \hat{q}_i$  and  $\dot{\mathfrak{c}}(\hat{q}_i, x^\alpha) = 0$ , where  $x_i \in \mathcal{S}$  and  $\alpha \in \mathbb{N}_0^S$ . For a given  $\hat{\sigma} \in \mathbb{R}^{\mathcal{S} \cup \Theta}$ , let  $\hat{\mathcal{P}}(\hat{\sigma})$  denote the PIVP which arises from  $\hat{\mathcal{P}}$  by replacing each  $v \in \mathcal{S} \cup \Theta$  by the corresponding real value  $\sigma(v) \in \mathbb{R}$  in  $\hat{\mathcal{P}}$ . In particular, let  $\hat{\sigma}_0 \in \mathbb{R}^{\mathcal{S} \cup \Theta}$  be such that  $\mathcal{P}(\sigma) = \hat{\mathcal{P}}(\hat{\sigma}_0)$ .*

*Example 5.* If  $\mathcal{P}$  is the PIVP from Example 1, its extended version  $\hat{\mathcal{P}}$  is

$$\begin{aligned} \dot{x}_1 &= \mathfrak{c}(\hat{q}_1, x_1)x_1 + \mathfrak{c}(\hat{q}_1, x_2)x_2 + \mathfrak{c}(\hat{q}_1, x_3)x_3, & \dot{\mathfrak{c}}(\hat{q}_1, x_i) &= 0, & i &= 1, 2, 3, \\ \dot{x}_2 &= \mathfrak{c}(\hat{q}_2, x_1)x_1 + \mathfrak{c}(\hat{q}_2, x_2)x_2, & \dot{\mathfrak{c}}(\hat{q}_2, x_i) &= 0, & i &= 1, 2, 3, \\ \dot{x}_3 &= \mathfrak{c}(\hat{q}_3, x_1)x_1 + \mathfrak{c}(\hat{q}_3, x_2)x_2, & \dot{\mathfrak{c}}(\hat{q}_3, x_i) &= 0, & i &= 1, 2, 3. \end{aligned}$$

The corresponding  $\hat{\sigma}_0$  satisfies  $\hat{\sigma}_0(x_i) = \sigma(x_i)$  for  $1 \leq i \leq 3$  and

$$\begin{aligned} \hat{\sigma}_0(\mathfrak{c}(\hat{q}_1, x_1)) &= -4.00, & \hat{\sigma}_0(\mathfrak{c}(\hat{q}_1, x_2)) &= 1.00, & \hat{\sigma}_0(\mathfrak{c}(\hat{q}_1, x_3)) &= 1.00, \\ \hat{\sigma}_0(\mathfrak{c}(\hat{q}_2, x_1)) &= 1.99, & \hat{\sigma}_0(\mathfrak{c}(\hat{q}_2, x_2)) &= -1.00, \\ \hat{\sigma}_0(\mathfrak{c}(\hat{q}_3, x_1)) &= 2.01, & \hat{\sigma}_0(\mathfrak{c}(\hat{q}_3, x_2)) &= -1.00. \end{aligned}$$

The following is needed for the definition of the reference PIVP.

**Definition 7.** *Given constant free polynomial  $\hat{\phi}$  (i.e., such that  $\hat{\phi}(0) = 0$ ) and  $\Xi \subseteq \mathcal{S} \cup \Theta \cup \{s\}$ , let  $\mathfrak{t}(\hat{\phi}, x^\alpha, \Xi)$  denote the coefficient term of  $x^\alpha$  in  $\mathcal{N}(\hat{\phi}, \Xi)$ , where  $\alpha \in \mathbb{N}_0^{\Xi}$  and  $\mathcal{N}(\hat{\phi}, \Xi)$  is the normal form of  $\hat{\phi}$  where variables outside  $\Xi$  are treated as parameters.*

*Example 6.* With  $\hat{q}_2$  and  $\hat{q}_3$  as in Example 4 and  $\Xi = \{x_1, x_2, x_3\}$ , the normal form  $\mathcal{N}(\hat{q}_2 - \hat{q}_3, \Xi)$  is given by  $(\mathfrak{c}(\hat{q}_2, x_1) - \mathfrak{c}(\hat{q}_3, x_1))x_1 + (\mathfrak{c}(\hat{q}_2, x_2) - \mathfrak{c}(\hat{q}_3, x_2))x_2$ , while  $\mathfrak{t}(\hat{q}_2 - \hat{q}_3, x_1, \Xi) = \mathfrak{c}(\hat{q}_2, x_1) - \mathfrak{c}(\hat{q}_3, x_1)$ .

**Definition 8.** Given a PIVP with variables  $\mathcal{S}$  and an  $\varepsilon$ -FDE partition  $\mathcal{H}$  of  $\mathcal{S}$ , the set of linear constraints of  $\mathcal{H}$  is given by

$$\{\mathbf{t}(\tilde{\varphi}_{i,j}^H, x^\alpha, \mathcal{S} \cup \{s\}) = 0 \mid \alpha \in \mathbb{N}_0^{\mathcal{S} \cup \{s\}}, H \in \mathcal{H} \text{ and } x_i \sim_{\mathcal{H}} x_j\} \quad (5)$$

with  $\tilde{\varphi}_{i,j}^H = \sum_{x_k \in H} \hat{q}_k - \sum_{x_k \in H} \hat{q}_k [x_i/s(x_i + x_j), x_j/(1-s)(x_i + x_j)]$ .

If  $\mathcal{H}$  is an  $\varepsilon$ -BDE partition of  $\mathcal{S}$ , the corresponding set of linear constraints is

$$\begin{aligned} &\{\mathbf{t}(\tilde{\varphi}_{i,j}^{\mathcal{H}}, x^\alpha, \mathcal{S}) = 0 \mid \alpha \in \mathbb{N}_0^{\mathcal{S}}, x_i \sim_{\mathcal{H}} x_j\} \\ &\cup \{x_{i_j} - x_{i_{j+1}} = 0 \mid 1 \leq j \leq k-1 \text{ and } \{x_{i_1}, \dots, x_{i_k}\} \in \mathcal{S}/\sim_{\mathcal{H}}\}, \end{aligned} \quad (6)$$

where  $\tilde{\varphi}_{i,j}^{\mathcal{H}} = (\hat{q}_i - \hat{q}_j)[x_{H',1}/x_{H'}, \dots, x_{H',|H'|}/x_{H'} : H' \in \mathcal{H}]$ .

*Example 7.* From Example 2, we know that  $\mathcal{H} = \{\{x_1\}, \{x_2, x_3\}\}$  is a 0.02-BDE partition of the PIVP (1). The set of linear constraints underlying  $\mathcal{H}$  is given by  $\mathbf{c}(\hat{q}_2, x_1) - \mathbf{c}(\hat{q}_3, x_1) = 0$  and  $x_2 - x_3 = 0$ .

*Remark 1.* In line with its exact counterpart, an  $\varepsilon$ -BDE is “useful” under the further constraint that related variables have the same initial conditions in the reference model, as a necessary condition for having equal solutions at all time points. This translates into adding the constraints in (6) that perturbed initial conditions of related variables are equal. This leads, for instance, to the constraint  $x_2 - x_3 = 0$  in the running example. For  $\varepsilon$ -FDE, instead, only constraints on the parameters  $\Theta$  are made.

**Theorem 4.** Given a PIVP  $\mathcal{P}$  with variables  $\mathcal{S}$ , an  $\varepsilon$ -FDE/BDE partition  $\mathcal{H}$  and a configuration  $\hat{\sigma} \in \mathbb{R}^{\mathcal{S} \cup \Theta}$  that satisfies (5)/(6), it holds that  $\mathcal{H}$  is an FDE/BDE of  $\hat{\mathcal{P}}(\hat{\sigma})$ .

The linear system of constraints from Theorem 4 can be shown to be underdetermined, hence there are infinitely many perturbations that lead to an exact differential equivalence. This observation is an instance of the well-known fact that, in general, an approximate quotient is not unique. Here, we fix one candidate perturbation by assuming that nearby initial conditions yield nearby trajectories. This fact is asymptotically true due to Gronwall’s inequality, as mentioned in Sect. 1.

We are interested in finding a configuration  $\hat{\sigma}$  which satisfies the constraints of Theorem 4 and minimizes the distance  $\|\hat{\sigma} - \hat{\sigma}_0\|_2$ . Mathematically, this corresponds to the optimization problem

$$\hat{\sigma}_* = \underset{\hat{\sigma}: \text{Eq. (5)/(6) holds}}{\operatorname{argmin}} \|\hat{\sigma} - \hat{\sigma}_0\|_2 \quad (7)$$

Since the solution space of a linear system is convex, the Euclidian norm yields a convex quadratic program that can be solved in polynomial time [22].



*Example 8.* Let us continue Example 7 and assume that  $\sigma(x_2) = \sigma(x_3)$ . In such a case, it can be easily seen that  $\hat{\sigma}_*$  and  $\hat{\sigma}_0$  satisfy  $\hat{\sigma}_*(\mathbf{c}(\hat{q}_2, x_1)) = \hat{\sigma}_*(\mathbf{c}(\hat{q}_3, x_1)) = (\hat{\sigma}_0(\mathbf{c}(\hat{q}_2, x_1)) + \hat{\sigma}_0(\mathbf{c}(\hat{q}_3, x_1)))/2 = 2.00$  and coincide on all other parameters. In other words, the closest PIVP that enjoys an exact BDE relating  $x_2$  and  $x_3$  is given, as expected, by perturbing the coefficients 1.99 and 2.01 of (1) to their average value, yielding:

$$\dot{x}_1 = -4.00x_1 + x_2 + x_3 \quad \dot{x}_2 = 2.00x_1 - x_2 \quad \dot{x}_3 = 2.00x_1 - x_3$$

The above discussions are summarized in the following.

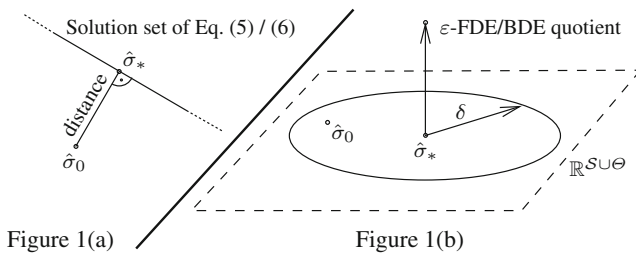
**Theorem 5.** *Given a PIVP,  $\varepsilon \geq 0$ , and an  $\varepsilon$ -FDE/BDE partition  $\mathcal{H}$ , the solution of (7) exists and can be computed in polynomial time.*

The solution of the optimization problem (7) stated in Theorem 5 is informally depicted in Fig. 1a.

The reference PIVP is the extended, exactly reducible PIVP with the optimum initial condition  $\hat{\sigma}_*$ , i.e.,  $\hat{\mathcal{P}}(\hat{\sigma}_*)$ . Its ODE solution is called the *reference trajectory*.

### 4 Error Bounds

The objective of this section is to provide a tight bound on the difference between the solution of the original PIVP and the reference. More specifically, we will show how to compute two values  $\delta > 0$  and  $\lambda > 0$  such that for all initial conditions  $\hat{\sigma}_1 \in \mathbb{R}^{S \cup \Theta}$  with  $\|x^{\hat{\sigma}_1}(0) - x^{\hat{\sigma}_*}(0)\| \leq \delta$ , it holds that  $\max_{0 \leq t \leq \hat{\tau}} \|x^{\hat{\sigma}_1}(t) - x^{\hat{\sigma}_*}(t)\| \leq \lambda \|x^{\hat{\sigma}_1}(0) - x^{\hat{\sigma}_*}(0)\|$ , where  $x^{\hat{\sigma}}$  denotes the solution underlying  $\hat{\mathcal{P}}(\hat{\sigma})$  and  $\hat{\tau} > 0$  is a previously fixed finite time horizon.



**Fig. 1.** Given a PIVP  $\mathcal{P}$ , a partition  $\mathcal{G}$  of  $\mathcal{S}$ , and an  $\varepsilon > 0$ , the coarsest  $\varepsilon$ -FDE/BDE partition  $\mathcal{H}$  that refines  $\mathcal{G}$  is constructed. Afterwards, the solution  $\hat{\sigma}_*$  of the optimization problem (7) is computed in Fig.1(a). This allows to compute the  $\varepsilon$ -FDE/BDE quotient  $\hat{\mathcal{P}}(\hat{\sigma}_*)$  of  $\mathcal{H}$ . With this,  $\lambda$  and  $\delta$  from Theorem 7 are calculated. In the case the distance between  $\hat{\sigma}_0$  and  $\hat{\sigma}_*$  does not exceed  $\delta$ , the tight bounds of Theorem 7 can be applied and relate the trajectories of  $\hat{\mathcal{P}}(\hat{\sigma}_*)$  and  $\hat{\mathcal{P}}(\hat{\sigma}_0) = \mathcal{P}(\sigma)$ , as depicted in Fig. 1(b).

The quantity  $\delta$  gives the size of the ball around the initial condition  $\hat{\sigma}_*$  of the reference PIVP, whereas  $\lambda$  is the *amplifier* that relates the maximum distance between trajectories to the distance between the initial conditions. Therefore, if the initial condition of the original PIVP  $\hat{\mathcal{P}}(\hat{\sigma}_0)$  falls within the prescribed  $\delta$  ball, then the above statement will provide a formal bound of the error made in approximating the original PIVP  $\hat{\mathcal{P}}(\hat{\sigma}_0)$  with the reference PIVP. This idea is visualized in Fig. 1(b).

Let us recall the notion of Jacobian matrix.

**Definition 9.** *Given an extended PIVP with variables  $\mathcal{S} \cup \Theta$ , the entries of the Jacobian matrix  $A = (A_{i,j})_{x_i, x_j \in \mathcal{S} \cup \Theta}$  are given by  $A_{i,j} = \partial_{x_j} \hat{q}_i$ , where  $\partial_x$  denotes the partial derivative with respect to  $x$ .*

Let  $A(t) \in \mathbb{R}^{\mathcal{S} \cup \Theta \times \mathcal{S} \cup \Theta}$  denote the Jacobian obtained by plugging in the reference trajectory  $x^{\hat{\sigma}^*}(t)$ . We will need the following result from the theory of ODEs.

**Theorem 6.** *There exists a family of matrices  $A(t_0, t_1)$ , with  $0 \leq t_0 \leq t_1 \leq \hat{\tau}$ , such that the solution of  $\dot{y}(t) = A(t)y(t)$ , where  $y(t_0) = y_0$  and  $t_0 \leq t \leq \hat{\tau}$ , is given by  $y(t) = A(t_0, t)y_0$  for all  $t_0 \leq t \leq \hat{\tau}$ .*

This is needed in the following.

**Theorem 7.** *Consider an extended PIVP  $\hat{\mathcal{P}}$  with variables  $\mathcal{S} \cup \Theta$  and define  $\lambda_0 = \max_{0 \leq t \leq \hat{\tau}} \|A(0, t)\|$  and  $\lambda_1 = \max_{0 \leq t_0 \leq t_1 \leq \hat{\tau}} \|A(t_0, t_1)\|$ . Further, define the remainder function  $r : [0; \hat{\tau}] \times \mathbb{R}^{\mathcal{S} \cup \Theta} \rightarrow \mathbb{R}^{\mathcal{S} \cup \Theta}$  via*

$$r(t, x - x^{\hat{\sigma}^*}(t)) = \hat{q}(x) - \hat{q}(x^{\hat{\sigma}^*}(t)) - A(t)(x - x^{\hat{\sigma}^*}(t))$$

and let  $0 \leq d_2, d_3, \dots$  be such that  $\|r(t, y)\| \leq \sum_{k=2}^{\deg(\hat{\mathcal{P}})} d_k \|y\|^k$  for all  $y \in \mathbb{R}^{\mathcal{S} \cup \Theta}$  and  $0 \leq t \leq \hat{\tau}$ . Then, with  $\lambda = 2\lambda_0$ , for any  $x^{\hat{\sigma}^1}(0) \in \mathbb{R}^{\mathcal{S} \cup \Theta}$ , it holds that

$$\|x^{\hat{\sigma}^1}(0) - x^{\hat{\sigma}^*}(0)\| \leq \delta \Rightarrow \max_{0 \leq t \leq \hat{\tau}} \|x^{\hat{\sigma}^1}(t) - x^{\hat{\sigma}^*}(t)\| \leq \lambda \|x^{\hat{\sigma}^1}(0) - x^{\hat{\sigma}^*}(0)\|$$

whenever  $\delta > 0$  satisfies  $\sum_{k=2}^{\deg(\hat{\mathcal{P}})} d_k (2\lambda_0 \delta)^{k-1} \leq (2\lambda_1 \hat{\tau})^{-1}$ .

Theorem 7 provides a bound on the difference  $x^{\hat{\sigma}^1}(t) - x^{\hat{\sigma}^*}(t)$  in terms of the initial perturbation  $x^{\hat{\sigma}^1}(0) - x^{\hat{\sigma}^*}(0)$  if the latter is sufficiently small, i.e., does not exceed  $\delta$ . We wish to point out that the maximal  $\delta$  satisfying  $\sum_{k=2}^{\deg(\hat{\mathcal{P}})} d_k (2\lambda_0 \delta)^{k-1} \leq (2\lambda_1 \hat{\tau})^{-1}$  is a root of a polynomial in one variable and thus can be efficiently approximated from below via Newton's method. Instead, the assumption  $\|r(s, y)\| \leq \sum_{k=2}^{\deg(\hat{\mathcal{P}})} d_k \|y\|^k$  on the remainder function  $r$  states essentially that, for any  $k \geq 2$ , the sum of all  $k$ -th order derivatives of  $r$  are bounded by  $d_k$  along the reference trajectory  $x^{\hat{\sigma}^*}$ .

The next result shows that the bound of Theorem 7 is tight and relies on the sharp bound available in the special case of linear systems (i.e.,  $\deg(\hat{\mathcal{P}}) = 1$ ) as discussed in [14].

**Theorem 8.** *If an extended PIVP  $\hat{\mathcal{P}}$  satisfies  $\deg(\hat{\mathcal{P}}) = 1$  and  $\lambda = 2\lambda_0$ , it holds that*

$$\max_{0 \leq t \leq \hat{\tau}} \|x^{\hat{\sigma}^1}(t) - x^{\hat{\sigma}^*}(t)\| \leq \frac{\lambda}{2} \|x^{\hat{\sigma}^1}(0) - x^{\hat{\sigma}^*}(0)\|$$

for any  $x^{\hat{\sigma}^1}(0) \in \mathbb{R}^{\mathcal{S} \cup \Theta}$ . The bound is tight in the sense that there exist  $0 \leq t \leq \hat{\tau}$  and  $x^{\hat{\sigma}^1}(0) \in \mathbb{R}^{\mathcal{S} \cup \Theta}$  such that  $\|x^{\hat{\sigma}^1}(t) - x^{\hat{\sigma}^*}(t)\| = \frac{\lambda}{2} \|x^{\hat{\sigma}^1}(0) - x^{\hat{\sigma}^*}(0)\|$ .

Note that the amplifier in Theorem 7 is twice as large as the amplifier in Theorem 8. This is because the proof of Theorem 7 has to estimate nonlinear terms present in the remainder function  $r$ . More importantly, Theorem 8 shows that the amplifier of Theorem 7 cannot be substantially improved.

*Remark 2.* We note that  $\lambda_0, \lambda_1$  can be estimated efficiently. Indeed, let  $e_{x_i} \in \mathbb{R}^{\mathcal{S} \cup \Theta}$  be the  $x_i$ -th unit vector in  $\mathbb{R}^{\mathcal{S} \cup \Theta}$ , i.e.,  $e_{x_i}(x_j) = \delta_{i,j}$  where  $\delta_{i,j}$  is the Kronecker delta. Then, if  $y(t_0) = e_{x_i}$ , Theorem 6 implies  $y(t_1) = \Lambda(t_0, t_1)e_{x_i}$ . Since  $\Lambda(0, t_1)e_{x_i}$  is the  $x_i$ -column of  $\Lambda(0, t_1)$  and  $\Lambda(t_0, t_1) = \Lambda(0, t_1)\Lambda(0, t_0)^{-1}$ , this shows that the matrices  $\Lambda(t_0, t_1)$  can be computed by solving  $|\mathcal{S} \cup \Theta|$  instances of the linear ODE system from Theorem 6 up to time  $\hat{\tau}$ .

By calculating a bound  $L > 0$  on  $\max_{0 \leq t \leq \hat{\tau}} \|\Lambda(t)\|$  and by computing the matrices  $\Lambda(t_i, t_j)$  for all time points  $t_k$  underlying a fixed discretization step  $\Delta t > 0$  of  $[0; \hat{\tau}]$ , the following can be shown.

**Lemma 1.** *Together with  $\lambda_0^+ = \max_i \|\Lambda(0, t_i)\|$  and  $\lambda_1^+ = \max_{i \leq j} \|\Lambda(t_i, t_j)\|$ , it holds that  $\lambda_0 \leq \lambda_0^+ e^{L\Delta t}$  and  $\lambda_1 \leq \lambda_1^+[1 + L\Delta t(e^{L\Delta t} + 1)]$ .*

The next result simplifies the constraints on  $\delta$  from Theorem 7 if  $\deg(\hat{\mathcal{P}}) \leq 3$ .

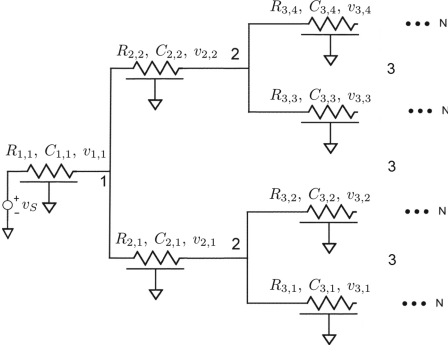
**Lemma 2.** *In the case where  $\deg(\hat{\mathcal{P}}) \leq 3$ , the constraint on  $\delta$  of Theorem 7 simplifies to  $\delta \leq \left[2\hat{\tau}\lambda_0\lambda_1 \left(d_2 + \sqrt{d_2^2 + \frac{2d_3}{\lambda_1\hat{\tau}}}\right)\right]^{-1}$ .*

The above lemma applies, for instance, to most biochemical systems, as discussed in Sect. 3. The next result, instead, allows for an efficient estimation of  $d_k$ , with  $2 \leq k \leq \deg(\hat{\mathcal{P}})$ .

**Lemma 3.** *Given an extended PIVP  $\hat{\mathcal{P}}$  with variables  $\mathcal{S} \cup \Theta$ , let  $\#_k(\hat{q}_i)$  be the number of degree  $k$  monomials in  $\mathcal{N}(\hat{q}_i)$  and  $D(\hat{q}_i, \hat{\sigma})$  the largest coefficient of  $\mathcal{N}(\hat{q}_i)$  for configuration  $\hat{\sigma} \in \mathbb{R}^{\mathcal{S} \cup \Theta}$ . With  $C = \max_{0 \leq t \leq \hat{\tau}} \|x^{\hat{\sigma}^*}(t)\|$ ,  $M = \max_{x_i \in \mathcal{S}} \max_{k \geq 2} \#_k(\hat{q}_i)$  and  $D = \max_{x_i \in \mathcal{S}} D(\hat{q}_i, \hat{\sigma}_*)$ , it suffices to set  $d_k$  in Theorem 7 to  $C^{\deg(\hat{\mathcal{P}})-k} MD$ .*

For the case of linear systems whose parameters are subject to perturbation, instead, the following lemma can be applied. It provides a sharper estimate on  $d_2$  but comes at the price of more involved computation.

**Lemma 4.** *Given an extended PIVP  $\hat{\mathcal{P}}$  with variables  $\mathcal{S} \cup \Theta$ , the Hessian matrix  $H^k = (H_{i,j}^k)_{x_i, x_j}$  of  $\hat{q}_k$  is given by  $H_{i,j}^k = \partial_{x_i} \partial_{x_j} \hat{q}_k$ . With this,  $d_2$  can be chosen as  $d_2 = \frac{1}{2} \cdot \max_{x_i \in \mathcal{S} \cup \Theta} \max_{0 \leq t \leq \hat{\tau}} \|H^i(x^{\hat{\sigma}^*}(t))\|$ .*



**Fig. 2.** H-tree network adapted from [31].

**Table 1.** Nominal parameters of electronic components at different depths  $i$ .

$i$	$R_i^*$ (m $\Omega$ )	$C_i^*$ (fF)
1	3.19	0.280
2	6.37	0.300
3	12.75	0.130
4	25.50	0.140
5	50.00	0.070
6	100.00	0.070
7	200.00	0.035
8	400.00	0.035

*Example 9.* Since  $\deg(\hat{\mathcal{P}}) = 2$  in Example 5, coefficients  $d_3, d_4, \dots$  are zero and we only need to bound  $d_2$ . Moreover, the constraint in Theorem 7 simplifies to  $\delta \leq (4\hat{\tau}\lambda_0\lambda_1d_2)^{-1}$  thanks to Lemma 2. By applying Lemma 3, instead, we see that it suffices to choose  $d_2 = 2.00$  because  $M = 2.00$  and  $D = 1.00$ . In the case of  $\hat{\tau} = 3.00$ , we thus get  $\lambda_0 = \lambda_1 = 1.40$  which yields  $\delta \leq 0.02$ .

## 5 Evaluation

We consider a simplified (inductance free) version of a power distribution electrical network from [31], arranged as a tree called *H-tree* (Fig. 2). We let  $N$  be the depth of the tree and denote the resistances and the capacitances at depth  $i$  by  $R_{i,k}$  and  $C_{i,k}$ , respectively. The source voltage is  $v_s$ , here assumed to be constant,  $v_s = 2.0V$ . Then, the voltage across  $C_{i,k}$ , denoted by  $v_{i,k}$ , obeys the affine ODE

$$\dot{v}_{1,1} = \frac{v_s - v_{1,1}}{R_{1,1}C_{1,1}} - \frac{v_{1,1} - v_{2,1}}{R_{2,1}C_{1,1}} - \frac{v_{1,1} - v_{2,2}}{R_{2,2}C_{1,1}}, \quad \dot{v}_{i,k} = \frac{v_{i-1,l} - v_{i,k}}{R_{i,k}C_{i,k}}, \quad (8)$$

where  $1 \leq i \leq N$ ,  $k = 1, \dots, 2^{i-1}$ , and  $l = \lceil k/2 \rceil$ , where  $\lceil \cdot \rceil$  denotes the ceil function. Here we considered networks with depth up to  $N = 8$ . For depths  $i \leq 4$ , the nominal parameter values  $R_i^*$  and  $C_i^*$  were taken from [31]; for  $i \geq 5$ , instead, we extrapolated them. The parameters are summarized in Table 1.

In [31] the authors show that when the values of resistors and capacitors of any depth are equal, i.e.,  $R_{i,\cdot} \equiv R_i^*$  and  $C_{i,\cdot} \equiv C_i^*$  then the network is *symmetric*. That is, the voltages at the capacitors in any level are equal at all time points. Indeed,  $\{\{v_{i,k} \mid 1 \leq k \leq 2^{i-1}\} \mid 1 \leq i \leq N\}$  is an exact BDE partition (with  $N$  equivalence classes).

We now study the robustness of the symmetry under the realistic assumption that resistances and capacitances are only approximately equal. In particular, we test whether it is possible to explain quasi-symmetries when the parameters have

**Table 2.** H-tree model results.

$N$	Reference model					Time (s) & Maximal Over-approximation					
	Time (s)	$\lambda$	$\delta$	$\ \cdot\ $	$\lambda \cdot \ \cdot\ $	C2E2		CORA		FLOW*	
2	1.96E+0	5.41	7.95E-4	4.43E-4	2.40E-3	1.00E+1	safe	4.82E+1	8.50E-3	2.51E+1	8.21E-3
3	3.51E+0	6.27	6.33E-4	7.42E-5	4.65E-4	4.20E+2	safe	1.42E+2	7.80E-3	1.10E+2	9.74E-3
4	7.75E+0	7.78	4.71E-4	2.17E-5	1.68E-4	—		6.23E+2	5.50E-3	9.40E+2	1.16E-2
5	2.46E+1	7.78	4.71E-4	3.31E-4	2.58E-3	—		5.39E+3	5.00E-3	—	—
6	8.42E+1	7.78	4.71E-4	8.97E-5	6.98E-4	—		—	—	—	—
7	4.74E+2	7.78	4.71E-4	4.22E-4	3.28E-3	—		—	—	—	—
8	—	—	—	—	—	—		—	—	—	—

tolerance  $\eta = 0.01\%$ . This corresponds to a practical situation when components or measurements parameters enjoy high accuracy. We considered networks of different size by varying the maximum depth  $N$  from 2 to 8. For each size, in order to simulate a quasi-symmetric scenario we built 30 distinct ODE models by sampling values for  $R_{i,k}$  and  $C_{i,k}$  uniformly at random within  $\eta$  percent from their nominal values. These repetitions were made in order to avoid fixing a single instance that might unfairly favor our algorithm. To each model we applied the  $\varepsilon$ -BDE reduction algorithm; choosing  $\varepsilon = 6.00E-4$ , it returned a quotient corresponding to a perfectly symmetrical case. The reduction times were below 0.5s in all cases. (Throughout this section, the runtimes reported were measured on a VirtualBox virtual machine running Ubuntu 64 bits over an actual 2.6 GHz Intel Core i5 machine with 4 GB of RAM.) Then, we computed the values of  $\delta$  and  $\lambda$  over a time horizon of 7 times units. This was chosen as a representative time point, for any  $N$ , of the transient state of the network (to account for the fact that, typically, circuits are analyzed in the time domain for transient analysis only).

The presence of uncertain parameters required us to transform the originally affine system (8) into a polynomial system of degree two (by substituting each  $1/(R_{i,k}C_{j,l})$  with a corresponding new state variable) with  $2^{N+1}$  states. This nonlinearity ruled out the application of standard over-approximation techniques for linear systems.

We present the results for the random model with the smallest value of  $\delta$  in Table 2. The runtimes (second column) refer to the computation cost of the  $\lambda$ - $\delta$  pair. In all cases,  $\delta$  turned out to be larger than the distance between the original model and its quotient  $\|\hat{\sigma}_0 - \hat{\sigma}_*\| = \|x^{\hat{\sigma}_0}(0) - x^{\hat{\sigma}_*}(0)\|$  (shown in column  $\|\cdot\|$ ). This demonstrates that the 0.01% tolerance can be formally explained by approximate differential equivalence, as confirmed by the small values of the amplifiers  $\lambda$ .

We compared against C2E2, CORA, and Flow\*. (We did not compare our technique with other approaches applicable to nonlinear dynamics such as VNODE-LP [32], Ariadne [6] or HySAT/iSAT [28] because those have been compared to Flow\* in the past [12].) While for CORA and Flow\* the comparison

can be made directly, C2E2 seeks to decide whether a given set is reachable. In order to perform as fair a comparison as possible we chose unreachable sets generously far away from the over-approximations computed with our bound, in order to ensure that C2E2 computes an over-approximation proving this. In all three cases, the initial uncertain set was fixed so as to correspond to the ball around the initial condition of the reference model  $\hat{\sigma}_*$  that included the original model  $\hat{\sigma}_0$ . This is the most favorable condition for the tools since it corresponds to the smallest uncertainty set which can provide a guaranteed error bound. Other settings of C2E2, CORA, and Flow\* were chosen such to ensure successful termination. In the case of Flow\* we used estimation remainder 0.10 and allowed for adaptive Taylor model degree between 2 and 6. In C2E2 we set the  $K$  value to 2000. We used the time step 0.10 for C2E2 since this ensured safety for the models that could be analyzed. For CORA and Flow\* we set time step equal to 0.01 as this led to tight enough bounds. For our approach, instead, we used time step 0.023 because this ensured tight approximations of  $\lambda_0$  and  $\lambda_1$  via Lemma 1. In all cases the time-out was set to 3 h.

The comparison results are also reported in Table 2. The over-approximations computed by C2E2 are not shown because they need not to be tight in order to verify the reachability problem. This is because C2E2 is refining an over-approximation only if this is necessary to decide safety. Thus, we report that C2E2 was able to verify a set to be safe, i.e., unreachable. For a network of depth  $N$ , the over-approximations for CORA and Flow\* are reported as the maximal diameter of the flowpipe underlying  $v_{N,1}$  across all time points. As such, it can be compared to the product  $\lambda \cdot \|\hat{\sigma}_0 - \hat{\sigma}_*\|$  given by our bound, which is also explicitly reported in the table for the sake of easy comparability (column  $\lambda \cdot \|\cdot\|$ ). Both CORA and Flow\* reported tight bounds, of the same orders as ours. These correspond to at most ca. 1% error on the observed variable, which cannot exceed the value of 2.0 (the source voltage applied to the network). C2E2 did not terminate within the time-out with models with  $N > 3$ , while Flow\* ran out of memory for  $N > 4$ ; CORA, instead, failed to compute the symbolic Jacobian matrices for  $N > 5$ . Our approach, instead, timed out for  $N = 8$ . However, we wish to point out that our algorithm naturally applies to parallelization. Indeed, its bottleneck is in the computation of the set of linear ODE systems discussed in Remark 2, which can be trivially solved independently from each other.

*Discussion.* In summary, the experimental results suggest that our bounding technique may complement the current state of the art in reachability analysis. Indeed, it has been shown to handle ODE systems of larger size, but it provides a  $\delta$  neighborhood that can explain perturbations up to ca 0.1% at best. This makes our approach beneficial for the automatic detection and abstraction from quasi-symmetries due to small uncertainties, e.g., measurement errors. On the other hand, algorithms such as those implemented in C2E2, CORA, and Flow\* can theoretically cover arbitrarily larger initial uncertainties, but at a computational cost that blocked their applicability to our larger benchmarks. In future work it is worth investigating a possible combination of these techniques.

## 6 Conclusion

Reasoning about quantitative properties approximately can represent an effective way of taming the complexity of real systems. Here we have considered ordinary differential equations (ODEs) with polynomial derivatives. We developed notions of equivalence as a relaxation of their exact counterparts, allowing the derivatives of related ODE variables to vary up to a desired tolerance. Our algorithmic approach can be useful to systematically discover quasi-symmetries in situations such as those presented in our case study. In future work, it would be also possible to integrate other bounding techniques, such as [34] which lacks an automatic synthesis of a reference model but can offer a tradeoff between tightness of the bound and computation cost in its derivation.

**Acknowledgement.** Luca Cardelli is partially funded by a Royal Society Research Professorship. Mirco Tribastone is supported by a DFG Mercator Fellowship (SPP 1593, DAPS2 Project). Max Tschaikowski is supported by a Lise Meitner Fellowship funded by the Austrian Science Fund (FWF) under grant number M 2393-N32 (COCO).

## References

1. Abate, A., Brim, L., Češka, M., Kwiatkowska, M.: Adaptive aggregation of Markov chains: quantitative analysis of chemical reaction networks. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 195–213. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-21690-4\\_12](https://doi.org/10.1007/978-3-319-21690-4_12)
2. Althoff, M.: Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In: HSCC, pp. 173–182 (2013)
3. Althoff, M.: An introduction to CORA 2015. In: Proceedings of the Workshop on Applied Verification for Continuous and Hybrid Systems (2015)
4. Arand, J.: In vivo control of CpG and non-CpG DNA methylation by DNA methyltransferases. *PLoS Genet.* **8**(6), e1002750 (2012)
5. Asarin, E., Dang, T., Girard, A.: Reachability analysis of nonlinear systems using conservative approximation. In: Maler, O., Pnueli, A. (eds.) HSCC 2003. LNCS, vol. 2623, pp. 20–35. Springer, Heidelberg (2003). [https://doi.org/10.1007/3-540-36580-X\\_5](https://doi.org/10.1007/3-540-36580-X_5)
6. Benvenuti, L., et al.: Reachability computation for hybrid systems with Ariadne. In: Proceedings of the 17th IFAC World Congress, vol. 41, no. 2, pp. 8960–8965 (2008)
7. Bogomolov, S., Frehse, G., Grosu, R., Ladan, H., Podelski, A., Wehrle, M.: A box-based distance between regions for guiding the reachability analysis of SpaceEx. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 479–494. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31424-7\\_35](https://doi.org/10.1007/978-3-642-31424-7_35)
8. Boreale, M.: Algebra, coalgebra, and minimization in polynomial differential equations. In: Esparza, J., Murawski, A.S. (eds.) FoSSaCS 2017. LNCS, vol. 10203, pp. 71–87. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54458-7\\_5](https://doi.org/10.1007/978-3-662-54458-7_5)
9. van Breugel, F., Worrell, J.: Towards quantitative verification of probabilistic transition systems. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 421–432. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-48224-5\\_35](https://doi.org/10.1007/3-540-48224-5_35)

10. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Maximal aggregation of polynomial dynamical systems. *Proc. Natl. Acad. Sci.* **114**(38), 10029–10034 (2017)
11. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Symbolic computation of differential equivalences. In: *POPL* (2016)
12. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow\*: an analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) *CAV 2013*. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39799-8\\_18](https://doi.org/10.1007/978-3-642-39799-8_18)
13. Danos, V., Laneve, C.: Formal molecular biology. *Theoret. Comput. Sci.* **325**(1), 69–110 (2004)
14. Donzé, A., Maler, O.: Systematic simulation using sensitivity analysis. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) *HSCC 2007*. LNCS, vol. 4416, pp. 174–189. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-71493-4\\_16](https://doi.org/10.1007/978-3-540-71493-4_16)
15. Duggirala, P.S., Mitra, S., Viswanathan, M.: Verification of annotated models from executions. In: *EMSOFT*, pp. 26:1–26:10. IEEE Press (2013)
16. Weinan, E., Li, T., Vanden-Eijnden, E.: Optimal partition and effective dynamics of complex networks. *PNAS* **105**(23), 7907–7912 (2008)
17. Fan, C., Qi, B., Mitra, S., Viswanathan, M., Duggirala, P.S.: Automatic reachability analysis for nonlinear hybrid models with C2E2. In: Chaudhuri, S., Farzan, A. (eds.) *CAV 2016*. LNCS, vol. 9779, pp. 531–538. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-41528-4\\_29](https://doi.org/10.1007/978-3-319-41528-4_29)
18. Girard, A., Pappas, G.: Approximate bisimulations for nonlinear dynamical systems. In: *IEEE Conference on Decision and Control and European Control Conference* (2005)
19. Iacobelli, G., Tribastone, M.: Lumpability of fluid models with heterogeneous agent types. In: *DSN* (2013)
20. Islam, M.A., et al.: Model-order reduction of ion channel dynamics using approximate bisimulation. *Theor. Comput. Sci.* **599**, 34–46 (2015)
21. Iwasa, Y., Levin, S.A., Andreasen, V.: Aggregation in model ecosystems II. Approximate aggregation. *Math. Med. Biol.* **6**(1), 1–23 (1989)
22. Kozlov, M., Tarasov, S., Khachiyan, L.: The polynomial solvability of convex quadratic programming. *USSR Comput. Math. Math. Phys.* **20**(5), 223–228 (1980)
23. Kuo, J.C.W., Wei, J.: Lumping analysis in monomolecular reaction systems. Analysis of approximately lumpable system. *Ind. Eng. Chem. Fund.* **8**(1), 124–133 (1969)
24. Lal, R., Prabhakar, P.: Bounded error flowpipe computation of parameterized linear systems. In: *EMSOFT*, pp. 237–246 (2015)
25. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *Inf. Comput.* **94**(1), 1–28 (1991)
26. Li, G., Rabitz, H.: A general analysis of approximate lumping in chemical kinetics. *Chem. Eng. Sci.* **45**(4), 977–1002 (1990)
27. Majumdar, R., Zamani, M.: Approximately bisimilar symbolic models for digital control systems. In: Madhusudan, P., Seshia, S.A. (eds.) *CAV 2012*. LNCS, vol. 7358, pp. 362–377. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-31424-7\\_28](https://doi.org/10.1007/978-3-642-31424-7_28)
28. Fränzle, M., Herde, C., Teige, T., Ratschan, S., Schubert, T.: Efficient solving of large non-linear arithmetic constraint systems with complex Boolean structure. *J. Satisfiability Boolean Model. Comput.* **1**, 209–236 (2007)
29. Pardalos, P.M., Vavasis, S.A.: Quadratic programming with one negative eigenvalue is NP-hard. *J. Global Optim.* **1**(1), 15–22 (1991)



30. Pedersen, M., Plotkin, G.D.: A language for biochemical systems: design and formal specification. In: Priami, C., Breitling, R., Gilbert, D., Heiner, M., Uhrmacher, A.M. (eds.) Transactions on Computational Systems Biology XII. LNCS, vol. 5945, pp. 77–145. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-11712-1\\_3](https://doi.org/10.1007/978-3-642-11712-1_3). IEEE/ACM
31. Rosenfeld, J., Friedman, E.G.: Design methodology for global resonant H-Tree clock distribution networks. *IEEE Trans. VLSI Syst.* **15**(2), 135–148 (2007)
32. Nedialkov, N.S.: Implementing a rigorous ODE solver through literate programming. In: Rauh, A., Auer, E. (eds.) Modeling, Design, and Simulation of Systems with Uncertainties. Mathematical Engineering, vol. 3, pp. 3–19. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-15956-5\\_1](https://doi.org/10.1007/978-3-642-15956-5_1)
33. Tschaikowski, M., Tribastone, M.: Tackling continuous state-space explosion in a Markovian process algebra. *Theor. Comput. Sci.* **517**, 1–33 (2014)
34. Tschaikowski, M., Tribastone, M.: Approximate reduction of heterogeneous nonlinear models with differential hulls. In: *IEEE TAC* (2016)



# Verifying Probabilistic Timed Automata Against Omega-Regular Dense-Time Properties

Hongfei Fu<sup>1</sup>, Yi Li<sup>2</sup>, and Jianlin Li<sup>3,4,5</sup>(✉)

<sup>1</sup> Shanghai Jiao Tong University, Shanghai, China  
fuhf@cs.sjtu.edu.cn

<sup>2</sup> Department of Informatics, School of Mathematical Sciences, Peking University,  
Beijing, China

<sup>3</sup> State Key Laboratory of Computer Science, Institute of Software,  
Chinese Academy of Sciences, Beijing, China

<sup>4</sup> University of Chinese Academy of Sciences, Beijing, China

<sup>5</sup> College of Computer Science and Technology,  
Nanjing University of Aeronautics and Astronautics, Nanjing, China  
ljlin@nuaa.edu.cn

**Abstract.** Probabilistic timed automata (PTAs) are timed automata (TAs) extended with discrete probability distributions. They serve as a mathematical model for a wide range of applications that involve both stochastic and timed behaviours. In this work, we consider the problem of model-checking linear *dense-time* properties over PTAs. In particular, we study linear dense-time properties that can be encoded by TAs with infinite acceptance criterion. First, we show that the problem of model-checking PTAs against deterministic-TA specifications can be solved through a product construction. Based on the product construction, we prove that the computational complexity of the problem with deterministic-TA specifications is EXPTIME-complete. Then we show that when relaxed to general (nondeterministic) TAs, the model-checking problem becomes undecidable. Our results substantially extend state of the art with both the dense-time feature and the nondeterminism in TAs.

## 1 Introduction

Stochastic timed systems are systems that exhibit both timed and stochastic behaviours. Such systems play a dominant role in many applications [1], hence addressing fundamental issues such as safety and performance over these systems are important. *Probabilistic timed automata* (PTAs) [2–4] serve as a good mathematical model for these systems. They extend the well-known model of timed automata [5] (for nonprobabilistic timed systems) with discrete probability distributions, and Markov Decision Processes (MDPs) [6] (for untimed probabilistic systems) with timing constraints.

Formal verification of PTAs has received much attention in recent years [2]. For branching-time model-checking of PTAs, the problem is reduced to computation of reachability probabilities over MDPs through well-known finite abstraction for timed automata (namely *regions* and *zones*) [3,4,7]. Advanced techniques for branching-time model checking of PTAs such as inverse method and symbolic method have been further explored in [8–11]. Extension with *cost* or *reward*, resulting in *priced* PTAs, has also been well investigated. Jurdzinski *et al.* [12] and Kwiatkowska *et al.* [13] proved that several notions of accumulated or discounted cost are computable over priced PTAs, while cost-bounded reachability probability over priced PTAs is shown to be undecidable by Berendsen *et al.* [14]. Most verification algorithms for PTAs have been implemented in the model checker PRISM [15]. Computational complexity of several verification problems for PTAs has been studied, for example, [12,16,17].

For linear-time model-checking, much less is known. As far as we know, the only relevant result is by Sproston [18] who proved that the problem of model-checking PTAs against linear *discrete-time* properties encoded by *untimed* deterministic omega-regular automata (e.g., Rabin automata) can be solved by a product construction. In his paper, Sproston first devised a production construction that produces a PTA out of the input PTA and the automaton; then he proved that the problem can be reduced to omega-regular verification of MDPs through maximal end components.

In this paper, we study the problem of model-checking linear *dense-time* properties over PTAs. Compared with discrete-time properties, dense-time properties take into account timing constraints, and therefore is more expressive and applicable to time-critical systems. Simultaneously, verification of dense-time properties is more challenging since it requires to involve timing constraints. The extra feature of timing constraints also brings more theoretical difficulty, e.g., timed automata [5] (TAs) are generally not determinizable, which is in contrast to untimed automata (such as Rabin or Muller automata).

We focus on linear dense-time properties that can be encoded by TAs. Due to the ability to model dense-time behaviours, TAs can be used to model real-time systems, while they can also act as language recognizers for timed omega-regular languages. Here we treat TAs as language recognizers for timed paths from a PTA, and study the problem of computing the minimum or maximum probability that a timed path from the PTA is accepted by the TA. The intuition is that a TA can recognize the set of “good” (or “bad”) timed paths emitting from a PTA, so the problem is to compute the probability that the PTA behaves in a good (or bad) manner. The relationship between TAs and linear temporal logic (e.g., Metric Temporal Logic [19]) is studied in [20,21].

*Our Contributions.* We distinguish between the subclass of *deterministic* TAs (DTAs) and general *nondeterministic* TAs. DTAs are the deterministic subclass of TAs. Although the class of DTAs is weaker than general timed automata, it can recognize a wide class of formal timed languages, and express interesting linear dense-time properties which cannot be expressed in branching-time logics (cf. [22]). We consider Rabin acceptance condition as the infinite

acceptance criterion for TAs. We first show that the problem of model-checking PTAs against DTA specifications with Rabin acceptance condition can be solved through a nontrivial product construction which tackles the integrated feature of timing constraints and randomness. From the product construction, we further prove that the problem is EXPTIME-complete. Then we show that the problem becomes undecidable when one considers general TAs. Our results substantially extend previous ones (e.g. [18]) with both the dense-time feature and the nondeterminism in TAs.

Due to lack of space, detailed proofs of several results and some experimental results are put in the full version [23].

## 2 Preliminaries

We denote by  $\mathbb{N}$ ,  $\mathbb{N}_0$ ,  $\mathbb{Z}$ , and  $\mathbb{R}$  the sets of all positive integers, non-negative integers, integers and real numbers, respectively. For any infinite word  $w = b_0b_1\dots$  over an alphabet  $\Sigma$ , we denote by  $\text{inf}(w)$  the set of symbols in  $\Sigma$  that occur infinitely often in  $w$ . A *clock* is a variable for a nonnegative real number. Below we fix a finite set  $\mathcal{X}$  of clocks.

*Clock Valuations.* A *clock valuation* is a function  $\nu : \mathcal{X} \rightarrow [0, \infty)$ . The set of clock valuations is denoted by  $\text{Val}(\mathcal{X})$ . Given a clock valuation  $\nu$ , a subset  $X \subseteq \mathcal{X}$  of clocks and a non-negative real number  $t$ , we let (i)  $\nu[X := 0]$  be the clock valuation such that  $\nu[X := 0](x) = 0$  for  $x \in X$  and  $\nu[X := 0](x) = \nu(x)$  otherwise, and (ii)  $\nu + t$  be the clock valuation such that  $(\nu + t)(x) = \nu(x) + t$  for all  $x \in \mathcal{X}$ . We denote by  $\mathbf{0}$  the clock valuation such that  $\mathbf{0}(x) = 0$  for  $x \in \mathcal{X}$ .

*Clock Constraints.* The set  $CC(\mathcal{X})$  of *clock constraints* over  $\mathcal{X}$  is generated by the following grammar:  $\phi := \mathbf{true} \mid x \leq d \mid c \leq x \mid x + c \leq y + d \mid \neg\phi \mid \phi \wedge \phi$  where  $x, y \in \mathcal{X}$  and  $c, d \in \mathbb{N}_0$ . We write  $\mathbf{false}$  for a short hand of  $\neg\mathbf{true}$ . The satisfaction relation  $\models$  between valuations  $\nu$  and clock constraints  $\phi$  is defined through substituting every  $x \in \mathcal{X}$  appearing in  $\phi$  by  $\nu(x)$  and standard semantics for logical connectives. For a given clock constraint  $\phi$ , we denote by  $\llbracket \phi \rrbracket$  the set of all clock valuations that satisfy  $\phi$ .

### 2.1 Probabilistic Timed Automata

A *discrete probability distribution* over a countable non-empty set  $U$  is a function  $q : U \rightarrow [0, 1]$  such that  $\sum_{z \in U} q(z) = 1$ . The *support* of  $q$  is defined as  $\text{supp}(q) := \{z \in U \mid q(z) > 0\}$ . We denote the set of discrete probability distributions over  $U$  by  $\mathcal{D}(U)$ .

**Definition 1 (Probabilistic Timed Automata [2]).** A probabilistic timed automaton (PTA)  $\mathcal{C}$  is a tuple

$$\mathcal{C} = (L, \ell^*, \mathcal{X}, \text{Act}, \text{inv}, \text{enab}, \text{prob}, AP, \mathcal{L}) \quad (1)$$

where:

- $L$  is a finite set of locations;
- $\ell^* \in L$  is the initial location;
- $\mathcal{X}$  is a finite set of clocks;
- $\text{Act}$  is a finite set of actions;
- $\text{inv} : L \rightarrow \text{CC}(\mathcal{X})$  is an invariant condition;
- $\text{enab} : L \times \text{Act} \rightarrow \text{CC}(\mathcal{X})$  is an enabling condition;
- $\text{prob} : L \times \text{Act} \rightarrow \mathcal{D}(2^{\mathcal{X}} \times L)$  is a probabilistic transition function;
- $AP$  is a finite set of atomic propositions;
- $\mathcal{L} : L \rightarrow 2^{AP}$  is a labelling function.

W.l.o.g, we consider that both  $\text{Act}$  and  $AP$  is disjoint from  $[0, \infty)$ . Below we fix a PTA  $\mathcal{C}$ . The semantics of PTAs is as follows.

*States and Transition Relation.* A state of  $\mathcal{C}$  is a pair  $(\ell, \nu)$  in  $L \times \text{Val}(\mathcal{X})$  such that  $\nu \models \text{inv}(\ell)$ . The set of all states is denoted by  $S_{\mathcal{C}}$ . The *transition relation*  $\rightarrow$  consists of all triples  $((\ell, \nu), a, (\ell', \nu'))$  satisfying the following conditions:

- $(\ell, \nu), (\ell', \nu')$  are states and  $a \in \text{Act} \cup [0, \infty)$ ;
- if  $a \in [0, \infty)$  then  $\nu + \tau \models \text{inv}(\ell)$  for all  $\tau \in [0, a]$  and  $(\ell', \nu') = (\ell, \nu + a)$ ;
- if  $a \in \text{Act}$  then  $\nu \models \text{enab}(\ell, a)$  and there exists a pair  $(X, \ell'') \in \text{supp}(\text{prob}(\ell, a))$  such that  $(\ell', \nu') = (\ell'', \nu[X := 0])$ .

By convention, we write  $s \xrightarrow{a} s'$  instead of  $(s, a, s') \in \rightarrow$ . We omit ‘ $\mathcal{C}$ ’ in ‘ $S_{\mathcal{C}}$ ’ if the underlying context is clear.

*Probability Transition Kernel.* The *probability transition kernel*  $\mathbf{P}$  is the function  $\mathbf{P} : S \times \text{Act} \times S \rightarrow [0, 1]$  such that

$$\mathbf{P}((\ell, \nu), a, (\ell', \nu')) = \begin{cases} 1 & \text{if } (\ell, \nu) \xrightarrow{a} (\ell', \nu') \text{ and } a \in [0, \infty) \\ \sum_{Y \in B} \text{prob}(\ell, a)(Y, \ell') & \text{if } (\ell, \nu) \xrightarrow{a} (\ell', \nu') \text{ and } a \in \text{Act} \\ 0 & \text{otherwise} \end{cases}$$

where  $B := \{X \subseteq \mathcal{X} \mid \nu' = \nu[X := 0]\}$ .

*Well-formedness.* We say that  $\mathcal{C}$  is *well-formed* if for every state  $(\ell, \nu)$  and action  $a \in \text{Act}$  such that  $\nu \models \text{enab}(\ell, a)$  and every  $(X, \ell') \in \text{supp}(\text{prob}(\ell, a))$ , one has that  $\nu[X := 0] \models \text{inv}(\ell')$ . The well-formedness is to ensure that when an action is enabled, the next state after taking this action will always be legal. In the following, we always assume that the underlying PTA is well-formed. Non-well-formed PTAs can be repaired into well-formed PTAs [9].

*Paths.* A *finite path*  $\rho$  (under  $\mathcal{C}$ ) is a finite sequence  $\langle s_0, a_0, s_1, \dots, a_{n-1}, s_n \rangle$  ( $n \geq 0$ ) in  $S \times ((\text{Act} \cup [0, \infty)) \times S)^*$  such that (i)  $s_0 = (\ell^*, \mathbf{0})$ , (ii)  $a_{2k} \in [0, \infty)$  (resp.  $a_{2k+1} \in \text{Act}$ ) for all integers  $0 \leq k \leq \frac{n}{2}$  (resp.  $0 \leq k \leq \frac{n-1}{2}$ ) and (iii) for all  $0 \leq k \leq n-1$ ,  $s_k \xrightarrow{a_k} s_{k+1}$ . The length  $|\rho|$  of  $\rho$  is defined by  $|\rho| := n$ . An *infinite path* (under  $\mathcal{C}$ ) is an infinite sequence  $\langle s_0, a_0, s_1, a_1, \dots \rangle$  in  $(S \times (\text{Act} \cup [0, \infty)))^\omega$

such that for all  $n \in \mathbb{N}_0$ , the prefix  $\langle s_0, a_0, \dots, a_{n-1}, s_n \rangle$  is a finite path. The set of finite (resp. infinite) paths under  $\mathcal{C}$  is denoted by  $Paths_{\mathcal{C}}^*$  (resp.  $Paths_{\mathcal{C}}^\omega$ ).

*Schedulers.* A (deterministic) scheduler is a function  $\sigma$  from the set of finite paths into  $Act \cup [0, \infty)$  such that for all finite paths  $\rho = s_0 a_0 \dots s_n$ , (i)  $\sigma(\rho) \in Act$  (resp.  $\sigma(\rho) \in [0, \infty)$ ) if  $n$  is odd (resp. even) and (ii) there exists a state  $s'$  such that  $s_n \xrightarrow{\sigma(\rho)} s'$ .

*Paths under Schedulers.* A finite path  $s_0 a_0 \dots s_n$  follows a scheduler  $\sigma$  if for all  $0 \leq m < n$ ,  $a_m = \sigma(s_0 a_0 \dots s_m)$ . An infinite path  $s_0 a_0 s_1 a_1 \dots$  follows  $\sigma$  if for all  $n \in \mathbb{N}_0$ ,  $a_n = \sigma(s_0 a_0 \dots s_n)$ . The set of finite (resp. infinite) paths following a scheduler  $\sigma$  is denoted by  $Paths_{\mathcal{C},\sigma}^*$  (resp.  $Paths_{\mathcal{C},\sigma}^\omega$ ). We note that the set  $Paths_{\mathcal{C},\sigma}^*$  is countably infinite from definition.

*Probability Spaces under Schedulers.* Let  $\sigma$  be any scheduler. The probability space w.r.t  $\sigma$  is defined as  $(\Omega^{\mathcal{C},\sigma}, \mathcal{F}^{\mathcal{C},\sigma}, \mathbb{P}^{\mathcal{C},\sigma})$  where (i)  $\Omega^{\mathcal{C},\sigma} := Paths_{\mathcal{C},\sigma}^\omega$ , (ii)  $\mathcal{F}^{\mathcal{C},\sigma}$  is the smallest sigma-algebra generated by all cylinder sets induced by finite paths for which a finite path  $\rho$  induces the cylinder set  $Cyl(\rho)$  of all infinite paths in  $Paths_{\mathcal{C},\sigma}^\omega$  with  $\rho$  being their (common) prefix, and (iii)  $\mathbb{P}^{\mathcal{C},\sigma}$  is the unique probability measure such that for all finite paths  $\rho = s_0 a_0 \dots a_{n-1} s_n$  in  $Paths_{\mathcal{C},\sigma}^*$ ,

$$\mathbb{P}^{\mathcal{C},\sigma}(Cyl(\rho)) = \prod_{k=0}^{n-1} \mathbf{P}(s_k, \sigma(s_0 a_0 \dots a_{k-1} s_k), s_{k+1}).$$

For details see [4].

*Zenoness and Time-Divergent Schedulers.* An infinite path  $\pi = s_0 a_0 s_1 a_1 \dots$  is *zeno* if  $\sum_{n=0}^\infty d_n < \infty$ , where  $d_n := a_n$  if  $a_n \in [0, \infty)$  and  $d_n := 0$  otherwise. Then a scheduler  $\sigma$  is *time divergent* if  $\mathbb{P}^{\mathcal{C},\sigma}(\{\pi \mid \pi \text{ is zeno}\}) = 0$ . In the following, we only consider time-divergent schedulers. The purpose is to eliminate non-realistic zeno behaviours (i.e., performing infinitely many actions within a finite amount of time).

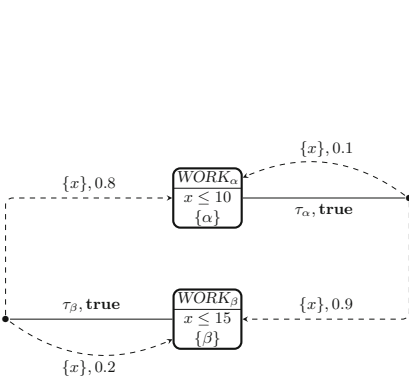


Fig. 1. A simple task-processing example

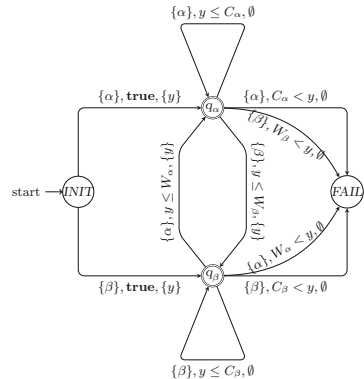


Fig. 2. A DTRA specification

In the following example, we illustrate a PTA which models a simple task-processing example.

*Example 1.* Consider the PTA depicted in Fig. 1.  $WORK_\alpha, WORK_\beta$  are locations and  $x$  is the only clock. Below each location first comes (vertically) its invariant condition and then the set of labels assigned to the location. For example,  $inv(WORK_\alpha) = x \leq 10$  and  $\mathcal{L}(WORK_\alpha) = \{\alpha\}$ . The two dots together with their corresponding solid line and dashed arrows refer to two actions  $\tau_\alpha, \tau_\beta$  with their enabling conditions and transition probabilities given by the probabilistic transition function. For example, the upper dot at the right of  $WORK_\alpha$  refers to the action  $\tau_\alpha$  for which  $enab(WORK_\alpha, \tau_\alpha) = \mathbf{true}$ ,  $prob(WORK_\alpha, \tau_\alpha)(\{x\}, WORK_\alpha) = 0.1$ , and  $prob(WORK_\alpha, \tau_\alpha)(\{x\}, WORK_\beta) = 0.9$ . The PTA models a faulty machine which processes two different kinds of jobs (i.e.,  $\alpha, \beta$ ) in an alternating fashion. If the machine fails to complete the current job, then it will repeat processing the job until it completes the job. For job  $\alpha$ , the machine always processes the job within 10 time units (cf. the invariant condition  $x \leq 10$ ), but may fail to complete the job with probability 0.1; Analogously, the machine always processes the job  $\beta$  within 15 time units (cf. the invariant condition  $x \leq 15$ ), but may fail to complete the job with probability 0.2. Note that we omit the initial location in this example.

## 2.2 Timed Automata

**Definition 2 (Timed Automata [22, 24, 25]).** A timed automaton (TA)  $\mathcal{A}$  is a tuple

$$\mathcal{A} = (Q, \Sigma, \mathcal{Y}, \Delta) \quad (2)$$

where

- $Q$  is a finite set of modes;
- $\Sigma$  is a finite alphabet of symbols disjoint from  $[0, \infty)$ ;
- $\mathcal{Y}$  is a finite set of clocks;
- $\Delta \subseteq Q \times \Sigma \times CC(\mathcal{Y}) \times 2^{\mathcal{Y}} \times Q$  is a finite set of rules.

$\mathcal{A}$  is a deterministic TA (DTA) if the following holds:

1. (determinism) for  $(q_i, b_i, \phi_i, X_i, q'_i) \in \Delta$  ( $i \in \{1, 2\}$ ), if  $(q_1, b_1) = (q_2, b_2)$  and  $\llbracket \phi_1 \rrbracket \cap \llbracket \phi_2 \rrbracket \neq \emptyset$  then  $(\phi_1, X_1, q'_1) = (\phi_2, X_2, q'_2)$ ;
2. (totality) for all  $(q, b) \in Q \times \Sigma$  and  $\nu \in Val(\mathcal{X})$ , there exists  $(q, b, \phi, X, q') \in \Delta$  such that  $\nu \models \phi$ .

Informally, A TA is deterministic if there is always exactly one rule applicable for the timed transition. We do not incorporate invariants in TAs as we use TAs as language acceptors.

Below we illustrate the semantics of TAs. We fix a TA  $\mathcal{A}$  in the form (2).

*Configurations and One-Step Transition Relation.* A configuration is a pair  $(q, \nu)$ , where  $q \in Q$  and  $\nu \in \text{Val}(\mathcal{Y})$ . The *one-step transition relation*

$$\Rightarrow \subseteq (Q \times \text{Val}(\mathcal{Y})) \times (\Sigma \cup [0, \infty)) \times (Q \times \text{Val}(\mathcal{Y}))$$

is defined by:  $((q, \nu), a, (q', \nu')) \in \Rightarrow$  iff either (i)  $a \in [0, \infty)$  and  $(q', \nu') = (q, \nu + a)$  or (ii)  $a \in \Sigma$  and there exists a rule  $(q, a, \phi, X, q') \in \Delta$  such that  $\nu \models \phi$  and  $\nu' = \nu[X := 0]$ . For the sake of convenience, we write  $(q, \nu) \xrightarrow{a} (q', \nu')$  instead of  $((q, \nu), a, (q', \nu')) \in \Rightarrow$ . Note that if  $\mathcal{A}$  is deterministic, then there is a unique  $(q', \nu')$  such that  $(q, \nu) \xrightarrow{a} (q', \nu')$  given any  $(q, \nu), a$ .

*Infinite Timed Words and Runs.* An *infinite timed word* is an infinite sequence  $w = \{a_n\}_{n \in \mathbb{N}_0}$  such that  $a_{2n} \in [0, \infty)$  and  $a_{2n+1} \in \Sigma$  for all  $n$ ; the infinite timed word  $w$  is *time-divergent* if  $\sum_{n \in \mathbb{N}_0} a_{2n} = \infty$ . A *run* of  $\mathcal{A}$  on an infinite timed word  $w = \{a_n\}_{n \in \mathbb{N}_0}$  with *initial configuration*  $(q, \nu)$ , is an infinite sequence  $\xi = \{(q_n, \nu_n, a_n)\}_{n \in \mathbb{N}_0}$  satisfying that  $(q_0, \nu_0) = (q, \nu)$  and  $(q_n, \nu_n) \xrightarrow{a_n} (q_{n+1}, \nu_{n+1})$  for all  $n \in \mathbb{N}_0$ ; the *trajectory*  $\text{traj}(\xi)$  of the run  $\xi$  is defined as an infinite word over  $Q$  such that  $\text{traj}(\xi) := q_0 q_1 \dots$ . Note that if  $\mathcal{A}$  is deterministic, then there is a unique run on every infinite timed word.

Below we illustrate the acceptance condition for TAs. We consider Rabin acceptance condition as the infinite acceptance condition.

**Definition 3 (Rabin Acceptance Condition [1]).** A TA with Rabin acceptance condition (TRA) is a tuple

$$A = (Q, \Sigma, \mathcal{Y}, \Delta, \mathcal{F}) \tag{3}$$

where  $(Q, \Sigma, \mathcal{Y}, \Delta)$  is a TA and  $\mathcal{F}$  is a finite set of pairs  $\mathcal{F} = \{(H_1, K_1), \dots, (H_n, K_n)\}$  representing a Rabin condition for which  $H_i$  and  $K_i$  are subsets of  $Q$  for all  $i \leq n$ .  $\mathcal{A}$  is a deterministic TRA (DTRA) if  $(Q, \Sigma, \mathcal{Y}, \Delta)$  is a DTA. A set  $Q' \subseteq Q$  is Rabin-accepting by  $\mathcal{F}$ , written as the predicate  $\text{ACC}(Q', \mathcal{F})$ , if there is  $1 \leq i \leq n$  such that  $Q' \cap H_i = \emptyset$  and  $Q' \cap K_i \neq \emptyset$ . An infinite timed word  $w$  is Rabin-accepted by  $\mathcal{A}$  with initial configuration  $(q, \nu)$  iff there exists a run  $\xi$  of  $(Q, \Sigma, \mathcal{Y}, \Delta)$  on  $w$  with  $(q, \nu)$  such that  $\text{inf}(\text{traj}(\xi))$  is Rabin-accepting by  $\mathcal{F}$ .

*Example 2.* Consider the DTRA depicted in Fig. 2. The alphabet of this DTRA is the powerset of atomic propositions in Fig. 1. In the figure, *INIT*,  $q_\alpha$ ,  $q_\beta$  and *FAIL* are modes with the Rabin condition  $\mathcal{F} = \{(\{\text{FAIL}\}, \{q_\alpha, q_\beta\})\}$ ,  $y$  is a clock and arrows between modes are rules.  $C_\gamma, W_\gamma$  ( $\gamma \in \{\alpha, \beta\}$ ) are undetermined integer constants. For example, there are four rules emitting from  $q_\alpha$ :

$$\begin{aligned} & (q_\alpha, \{\alpha\}, y \leq C_\alpha, \emptyset, q_\alpha), (q_\alpha, \{\beta\}, y \leq W_\beta, \{y\}, q_\beta), \\ & (q_\alpha, \{\alpha\}, C_\alpha < y, \emptyset, \text{FAIL}), (q_\alpha, \{\beta\}, W_\beta < y, \emptyset, \text{FAIL}). \end{aligned}$$

*INIT* is the initial mode to read the first symbol upon which transiting to either  $q_\alpha$  or  $q_\beta$ . *FAIL* is a deadlock mode from which all rules go to itself. Note that the rules of the DTRA does not satisfy the totality condition. However, we assume



that all missing rules lead to the mode *FAIL* and does not affect the Rabin acceptance condition. The mode  $q_\alpha$  does not reset the clock  $y$  until it reads  $\beta$ . Moreover,  $q_\alpha$  does not transit to *FAIL* only if the time spent within a maximal consecutive segment of  $\alpha$ 's (in an infinite timed word) is no greater than  $C_\alpha$  time units (cf. the rule  $(q_\alpha, \{\alpha\}, y \leq C_\alpha, \emptyset, q_\alpha)$ ) and the total time from the start of the segment until  $\beta$  is read (the time within a maximal consecutive segment of  $\alpha$ 's plus the time spent on the last  $\alpha$  in the segment) is no greater than  $W_\beta$  (cf. the rule  $(q_\alpha, \{\beta\}, y \leq W_\beta, \{y\}, q_\beta)$ ). The behaviour of the mode  $q_\beta$  can be argued similar to that of  $q_\alpha$  where the only difference is to flip  $\alpha$  and  $\beta$ . From the Rabin acceptance condition, the DTRA specifies a property on infinite timed words that the time spent within a maximal consecutive segment of  $\alpha$ 's (resp.  $\beta$ 's) and the total time until  $\beta$  (resp.  $\alpha$ ) is read always satisfy the conditions specified by  $q_\alpha$  (resp.  $q_\beta$ ).

### 3 Problem Statement

In this part, we define the PTA-TRA problem of model-checking PTAs against TA-specifications. The problem takes a PTA and a TRA as input, and computes the minimum and the maximum probability that infinite paths under the PTA are accepted by the TRA. Informally, the TRA encodes the linear dense-time property by judging whether an infinite path is accepted or not through its external behaviour, then the problem is to compute the probability that an infinite path is accepted by the TRA. In practice, the TRA can be used to capture all good (or bad) behaviours, so the problem can be treated as a task to evaluate to what extent the PTA behaves in a good (or bad) way.

Below we fix a well-formed PTA  $\mathcal{C}$  taking the form (1) and a TRA  $\mathcal{A}$  taking the form (3). W.l.o.g., we assume that  $\mathcal{X} \cap \mathcal{Y} = \emptyset$  and  $\Sigma = 2^{AP}$ . We first show how an infinite path in  $\text{Paths}_{\mathcal{C}}^\omega$  can be interpreted as an infinite timed word.

**Definition 4 (Infinite Paths as Infinite Timed Words).** *Given an infinite path  $\pi = (\ell_0, \nu_0)a_0(\ell_1, \nu_1)a_1(\ell_2, \nu_2)a_2 \dots$  under  $\mathcal{C}$ , the infinite timed word  $\mathcal{L}(\pi)$  is defined as  $\mathcal{L}(\pi) := a_0\mathcal{L}(\ell_2)a_2\mathcal{L}(\ell_4) \dots a_{2n}\mathcal{L}(\ell_{2n+2}) \dots$ . Recall that  $\nu_0 = \mathbf{0}$ ,  $a_{2n} \in [0, \infty)$  and  $a_{2n+1} \in \text{Act}$  for  $n \in \mathbb{N}_0$ .*

*Remark 1.* Informally, the interpretation in Definition 4 works by (i) dropping (a) the initial location  $\ell_0$ , (b) all clock valuations  $\nu_n$ 's, (c) all locations  $\ell_{2n+1}$ 's following a time-elapse, (d) all internal actions  $a_{2n+1}$ 's of  $\mathcal{C}$  and (ii) replacing every  $\ell_{2n}$  ( $n \geq 1$ ) by  $\mathcal{L}(\ell_{2n})$ . The interpretation captures only external behaviours including time-elapses and labels of locations upon state-change, and discards internal behaviours such as the concrete locations, clock valuations and actions. Although the interpretation ignores the initial location, we deal with it in our acceptance condition where the initial location is preprocessed by the TRA.

**Definition 5 (Path Acceptance).** *An infinite path  $\pi$  of  $\mathcal{C}$  is accepted by  $\mathcal{A}$  w.r.t initial configuration  $(q, \nu)$ , written as the single predicate  $\text{ACC}(\mathcal{A}, (q, \nu), \pi)$ , if there is a configuration  $(q', \nu')$  such that  $(q, \nu) \xrightarrow{\mathcal{L}(\ell^*)} (q', \nu')$  and the infinite word  $\mathcal{L}(\pi)$  is Rabin-accepted by  $\mathcal{A}$  with  $(q', \nu')$ .*

The initial location omitted in Definition 4 is preprocessed by specifying explicitly that the first label  $\mathcal{L}(\ell^*)$  is read by the initial configuration  $(q, \nu)$ . Below we define acceptance probabilities over infinite paths under  $\mathcal{C}$ .

**Definition 6 (Acceptance Probabilities).** *The probability that  $\mathcal{C}$  observes  $\mathcal{A}$  under scheduler  $\sigma$  and initial mode  $q \in Q$ , denoted by  $\mathfrak{p}_q^\sigma$ , is defined by:*

$$\mathfrak{p}_q^\sigma := \mathbb{P}^{\mathcal{C}, \sigma}(\text{AccPaths}_{\mathcal{C}, \sigma}^{\mathcal{A}, q})$$

where  $\text{AccPaths}_{\mathcal{C}, \sigma}^{\mathcal{A}, q}$  is the set of infinite paths under  $\mathcal{C}$  that are accepted by the TRA  $\mathcal{A}$  w.r.t  $(q, \mathbf{0})$  i.e.  $\text{AccPaths}_{\mathcal{C}, \sigma}^{\mathcal{A}, q} = \{\pi \in \text{Paths}_{\mathcal{C}, \sigma}^\omega \mid \mathbf{ACC}(\mathcal{A}, (q, \mathbf{0}), \pi)\}$ .

Since the set  $\text{Paths}_{\mathcal{C}, \sigma}^*$  is countably-infinite,  $\text{AccPaths}_{\mathcal{C}, \sigma}^{\mathcal{A}, q}$  is measurable since it can be represented as a countable intersection of certain countable unions of some cylinder sets (cf. [1, Remark 10.24] for details).

Now we introduce the PTA-TRA problem.

- **Input:** a well-formed PTA  $\mathcal{C}$ , a TRA  $\mathcal{A}$  and an initial mode  $q$  in  $\mathcal{A}$ ;
- **Output:**  $\inf_\sigma \mathfrak{p}_q^\sigma$  and  $\sup_\sigma \mathfrak{p}_q^\sigma$ , where  $\sigma$  ranges over all time-divergent schedulers for  $\mathcal{C}$ .

We refer to the problem as PTA-DTRA if  $\mathcal{A}$  is deterministic.

## 4 The PTA-DTRA Problem

In this section, we solve the PTA-DTRA problem through a product construction. Based on the product construction, we also settle the complexity of the problem. Below we fix a well-formed PTA  $\mathcal{C}$  in the form (1) and a DTRA  $\mathcal{A}$  in the form (3). W.l.o.g, we consider that  $\mathcal{X} \cap \mathcal{Y} = \emptyset$  and  $\Sigma = 2^{AP}$ .

**The Main Idea.** The core part of the product construction is a PTA which preserves the probability of the set of infinite paths accepted by  $\mathcal{A}$ . The intuition is to let  $\mathcal{A}$  reads external actions of  $\mathcal{C}$  while  $\mathcal{C}$  evolves along the time axis. The major difficulty is that when  $\mathcal{C}$  performs actions in  $\text{Act}$ , there is a probabilistic choice between the target locations. Then  $\mathcal{A}$  needs to know the labelling of the target location and the rule (in  $\Delta$ ) used for the transition. A naive solution is to integrate each single rule in  $\Delta$  into the enabling condition *enab* in  $\mathcal{C}$ . However, this simple solution does not work since a single rule fixes the labelling of a location in  $\mathcal{C}$ , while the probability distribution (given by *prob*) can jump to locations with different labels. We solve this difficulty by integrating into the enabling condition enough information on clock valuations under  $\mathcal{A}$  so that the rule used for the transition is clear.

**The Product Construction.** For each  $q \in Q$ , we let

$$\mathcal{T}_q := \{h : \Sigma \rightarrow CC(\mathcal{Y}) \mid \forall b \in \Sigma. (q, b, h(b), X, q') \in \Delta \text{ for some } X, q'\} .$$

The totality of  $\Delta$  ensures that  $\mathcal{T}_q$  is non-empty. Intuitively, every element of  $\mathcal{T}_q$  is a tuple of clock constraints  $\{\phi_b\}_{b \in \Sigma}$ , where each clock constraint  $\phi_b$  is chosen from the rules emitting from  $q$  and  $b$ . The *product PTA*  $\mathcal{C} \otimes \mathcal{A}_q$  (between  $\mathcal{C}$  and  $\mathcal{A}$  with initial mode  $q$ ) is defined as  $(L_\otimes, \ell_\otimes^*, \mathcal{X}_\otimes, Act_\otimes, inv_\otimes, enab_\otimes, prob_\otimes, Q, \mathcal{L}_\otimes)$  where:

- $L_\otimes := L \times Q$ ;
- $\ell_\otimes^* := (\ell^*, q^*)$  where  $q^*$  is the unique mode such that  $(q, \mathbf{0}) \xrightarrow{\mathcal{L}(\ell^*)} (q^*, \mathbf{0})$ ;
- $\mathcal{X}_\otimes := \mathcal{X} \cup \mathcal{Y}$ ;
- $Act_\otimes := Act \times \bigcup_q \mathcal{T}_q$ ;
- $inv_\otimes(\ell, q) := inv(\ell)$  for all  $(\ell, q) \in L_\otimes$ ;
- $enab_\otimes((\ell, q), (a, h)) := enab(\ell, a) \wedge \bigwedge_{b \in \Sigma} h(b)$  if  $h \in \mathcal{T}_q$ , and  $enab_\otimes((\ell, q), (a, h)) := \mathbf{false}$  otherwise, for all  $(\ell, q) \in L_\otimes, (a, h) \in Act_\otimes$ .
- $\mathcal{L}_\otimes(\ell, q) := \{q\}$  for all  $(\ell, q) \in L_\otimes$ ;
- $prob_\otimes$  is given by

$$prob_\otimes((\ell, q), (a, h))(Y, (\ell', q')) := \begin{cases} prob(\ell, a)(Y \cap \mathcal{X}, \ell') & \text{if } (q, \mathcal{L}(\ell'), h(\mathcal{L}(\ell')), Y \cap \mathcal{Y}, q') \\ & \text{is a (unique) rule in } \Delta \\ 0 & \text{otherwise} \end{cases}$$

for all  $(\ell, q), (\ell', q') \in L_\otimes, (a, h) \in Act_\otimes$  and  $Y \in \mathcal{X}_\otimes$ .

Besides standard constructions (e.g., the Cartesian product between  $L$  and  $Q$ ), the product construction also has Cartesian product between  $Act$  and  $\bigcup_q \mathcal{T}_q$ . For each extended action  $(a, h)$ , the enabling condition for this action is the conjunction between  $enab(\ell, a)$  and all clock constraints from  $h$ . This is to ensure that when the action  $(a, h)$  is taken, the clock valuation under  $\mathcal{A}$  satisfies every clock constraint in  $h$ . Then in the definition for  $prob_\otimes$ , upon the action  $(a, h)$ , the product PTA first perform probabilistic jump from  $\mathcal{C}$  with the target location  $\ell'$ , then chooses the unique rule  $(q, \mathcal{L}(\ell'), h(\mathcal{L}(\ell')), Y \cap \mathcal{Y}, q')$  from the emitting mode  $q$  and the label  $\mathcal{L}(\ell')$  for which the uniqueness comes from the determinism of  $\Delta$ , then perform the discrete transition from  $\mathcal{A}$ . Finally, we label each  $(\ell, q)$  by  $q$  to meet the Rabin acceptance condition.  $\square$

It is easy to see that the PTA  $\mathcal{C} \otimes \mathcal{A}_q$  is well-formed as  $\mathcal{C}$  is well-formed and  $\mathcal{A}$  does not introduce extra invariant conditions.

*Example 3.* The product PTA between the PTA in Example 1 and the DTRA in Example 2 is depicted in Fig. 3. In the figure,  $(WORK_\alpha, q_\alpha), (WORK_\beta, q_\beta)$  and  $(WORK_\alpha, FAIL), (WORK_\beta, FAIL)$  are product locations. We omit the initial location and unreachable locations in the product construction. From the construction of  $\mathcal{T}_q$ 's, the functions  $h_i$ 's are as follows (we omit redundant labels such as  $\emptyset$  and  $\{\alpha, \beta\}$  which never appear in the PTA):

- $h_0 = \{\{\alpha\} \mapsto y \leq C_\alpha, \{\beta\} \mapsto y \leq W_\beta\}$ ;
- $h_1 = \{\{\alpha\} \mapsto y \leq C_\alpha, \{\beta\} \mapsto W_\beta < y\}$ ;
- $h_2 = \{\{\alpha\} \mapsto C_\alpha < y, \{\beta\} \mapsto y \leq W_\beta\}$ ;



- for all  $0 \leq k < n$ , if  $a_k \in [0, \infty)$  then  $a'_k = a_k$  and  $(q_k, \mu_k) \xrightarrow{a_k} (q_{k+1}, \mu_{k+1})$ ;
- for all  $0 \leq k < n$ , if  $a_k \in \text{Act}$  then  $a'_k = (a_k, \xi_k)$  and  $(q_k, \mu_k) \xrightarrow{\mathcal{L}(\ell_{k+1})} (q_{k+1}, \mu_{k+1})$  where  $\xi_k$  is the unique function such that for each symbol  $b \in \Sigma$ ,  $\xi_k(b)$  is the unique clock constraint appearing in a rule emitting from  $q_k$  and with symbol  $b$  such that  $\mu_k \models \xi_k(b)$ .

Likewise, for an infinite path  $\pi = (\ell_0, \nu_0)a_0(\ell_1, \nu_1)a_1 \dots$  under  $\mathcal{C}$ , we define  $\mathcal{T}(\pi)$  to be the unique infinite path

$$\mathcal{T}(\pi) := ((\ell_0, q_0), \nu_0 \cup \mu_0)a'_0((\ell_1, q_1), \nu_1 \cup \mu_1)a'_1 \dots \quad (5)$$

under  $\mathcal{C} \otimes \mathcal{A}_q$  such that the three conditions in (†) hold for all  $k \in \mathbb{N}_0$  instead of all  $0 \leq k < n$ . From the determinism and totality of  $\mathcal{A}$ , it is straightforward to prove the following result.

**Lemma 1.** *The function  $\mathcal{T}$  is a bijection. Moreover, for any infinite path  $\pi$  under  $\mathcal{C}$ ,  $\pi$  is non-zeno iff  $\mathcal{T}(\pi)$  is non-zeno.*

Below we also show the correspondence on schedulers before and after the product construction.

**Transformation  $\theta$  for Schedulers from  $\mathcal{C}$  into  $\mathcal{C} \otimes \mathcal{A}_q$ .** We define the function  $\theta$  from the set of schedulers under  $\mathcal{C}$  into the set of schedulers under  $\mathcal{C} \otimes \mathcal{A}_q$  as follows: for any scheduler  $\sigma$  for  $\mathcal{C}$ ,  $\theta(\sigma)$  (for  $\mathcal{C} \otimes \mathcal{A}_q$ ) is defined such that for any finite path  $\rho$  under  $\mathcal{C}$  where  $\rho = (\ell_0, \nu_0)a_0 \dots a_{n-1}(\ell_n, \nu_n)$  and  $\mathcal{T}(\rho)$  given as in (4),

$$\theta(\sigma)(\mathcal{T}(\rho)) := \begin{cases} \sigma(\rho) & \text{if } n \text{ is even} \\ (\sigma(\rho), \lambda(\rho)) & \text{if } n \text{ is odd} \end{cases}$$

where  $\lambda(\rho)$  is the unique function such that for each symbol  $b \in \Sigma$ ,  $\lambda(\rho)(b)$  is the clock constraint in the unique rule emitting from  $q_n$  and with symbol  $b$  such that  $\mu_n \models \lambda(\rho)(b)$ . Note that the well-definedness of  $\theta$  follows from Lemma 1.

From Lemma 1, the product construction, the determinism and totality of  $\Delta$ , one can prove directly the following lemma.

**Lemma 2.** *The function  $\theta$  is a bijection.*

Now we prove the relationship between infinite paths accepted by a DTRA before product construction and infinite paths satisfying certain Rabin condition.

We introduce more notations. First, we lift the function  $\mathcal{T}$  to all subsets of paths in the standard fashion: for all subsets  $A \subseteq \text{Paths}_{\mathcal{C}}^* \cup \text{Paths}_{\mathcal{C}}^\omega$ ,  $\mathcal{T}(A) := \{\mathcal{T}(\omega) \mid \omega \in A\}$ . Then for an infinite path  $\pi$  under  $\mathcal{C} \otimes \mathcal{A}_q$  in the form (5), we define the *trace* of  $\pi$  as an infinite word over  $Q$  by  $\text{trace}(\pi) := q_0q_1 \dots$ . Finally, for any scheduler  $\sigma$  for  $\mathcal{C} \otimes \mathcal{A}_q$ , we define the set  $R\text{Paths}_\sigma$  by

$$R\text{Paths}_\sigma := \left\{ \pi \in \text{Paths}_{\mathcal{C} \otimes \mathcal{A}_q}^\omega \mid \mathbf{ACC}(\text{inf}(\text{trace}(\pi)), \mathcal{F}) \right\}.$$

Intuitively,  $R\text{Paths}_\sigma$  is the set of infinite paths under  $\mathcal{C} \otimes \mathcal{A}_q$  that meet the Rabin condition  $\mathcal{F}$  from  $\mathcal{A}$ . The following proposition clarifies the role of  $R\text{Paths}_\sigma$ .

**Proposition 1.** *For any scheduler  $\sigma$  for  $\mathcal{C}$  and any initial mode  $q$  for  $\mathcal{A}$ , we have  $\mathcal{T} \left( \text{AccPaths}_{\mathcal{C},\sigma}^{\mathcal{A},q} \right) = \text{RPaths}_{\theta(\sigma)}$ .*

Finally, we demonstrate the relationship between acceptance probabilities before product construction and Rabin(-accepting) probabilities after product construction. We also clarify the probability of zenoness before and after the product construction. The proof follows standard argument from measure theory.

**Proposition 2.** *For any scheduler  $\sigma$  for  $\mathcal{C}$  and mode  $q$ , the followings hold:*

- $\mathfrak{p}_q^\sigma = \mathbb{P}^{\mathcal{C},\sigma} \left( \text{AccPaths}_{\mathcal{C},\sigma}^{\mathcal{A},q} \right) = \mathbb{P}^{\mathcal{C} \otimes \mathcal{A}_q, \theta(\sigma)} \left( \text{RPaths}_{\theta(\sigma)} \right)$ ;
- $\mathbb{P}^{\mathcal{C},\sigma} \left( \{ \pi \mid \pi \text{ is zero} \} \right) = \mathbb{P}^{\mathcal{C} \otimes \mathcal{A}_q, \theta(\sigma)} \left( \{ \pi' \mid \pi' \text{ is zero} \} \right)$ .

A side result from Proposition 2 says that  $\theta$  preserves time-divergence for schedulers before and after product construction. From Proposition 2 and Lemma 2, one immediately obtains the following result which transforms the PTA-DTRA problem into Rabin(-accepting) probabilities under the product PTA.

**Corollary 1.** *For any initial mode  $q$ ,  $\text{opt}_\sigma \mathfrak{p}_q^\sigma = \text{opt}_{\sigma'} \mathbb{P}^{\mathcal{C} \otimes \mathcal{A}_q, \sigma'} \left( \text{RPaths}_{\sigma'} \right)$  where  $\text{opt}$  refers to either  $\inf$  (infimum) or  $\sup$  (supremum),  $\sigma$  (resp.  $\sigma'$ ) range over all time-divergent schedulers for  $\mathcal{C}$  (resp.  $\mathcal{C} \otimes \mathcal{A}_q$ ).*

**Solving Rabin Probabilities.** We follow the approach in [18] to solve Rabin probabilities over PTAs. Below we briefly describe the approach. The approach can be divided into two steps. The first step is to ensure time-divergence. This is achieved by (i) making a copy for every location in the PTA, (ii) enforcing a transition from every location to its copy to happen after 1 time-unit elapses, (iii) enforcing a transition from every copy location back to the original one immediately with no time-delay, and (iv) putting a special label *tick* in every copy. Then time-divergence is guaranteed by adding the label *tick* to the Rabin condition. The second step is to transform the problem into limit Rabin properties over MDPs [1, Theorem 10.127]. This step constructs an MDP  $\text{Reg}[\mathcal{C} \otimes \mathcal{A}_q]$  from the PTA  $\mathcal{C} \otimes \mathcal{A}_q$  through a *region-graph* construction so that the problem is reduced to solving limit Rabin properties over  $\text{Reg}[\mathcal{C} \otimes \mathcal{A}_q]$ . *Regions* are finitely-many equivalence classes of clock valuations that serve as a finite abstraction which capture exactly reachability behaviours over timed transitions (cf. [5]). Then standard methods based on *maximal end components* (MECs) are applied to  $\text{Reg}[\mathcal{C} \otimes \mathcal{A}_q]$ . In detail, the algorithm computes the reachability probability to MECs that satisfy the Rabin acceptance condition. In order to guarantee time-divergence, the algorithm only picks up MECs with at least one location that has a *tick* label. Based on this approach, our result leads to an algorithm for solving the problem PTA-DTRA.

Note that in  $\mathcal{C} \otimes \mathcal{A}_q$ , although the size of  $\text{Act}_\otimes$  may be exponential due to possible exponential blow-up from  $\mathcal{T}_q$ , one easily sees that  $|L_\otimes|$  is  $|L| \cdot |Q|$  and  $|\mathcal{X}_\otimes| = |\mathcal{X}| + |\mathcal{Y}|$ . Hence, the size of  $\text{Reg}[\mathcal{C} \otimes \mathcal{A}_q]$  is still exponential in the sizes of  $\mathcal{C}$  and  $\mathcal{A}$ . It follows that  $\text{opt}_\sigma \mathfrak{p}_q^\sigma$  can be calculated in exponential time from the MEC-based algorithm illustrated in [1, Theorem 10.127], as is demonstrated by the following proposition.

**Proposition 3.** *The problem PTA-DTRA is in EXPTIME in the size of the input PTA and DTRA.*

It is proved in [16] that the reachability-probability problem for arbitrary PTAs is EXPTIME-complete. Since Rabin acceptance condition subsumes reachability, one obtains that the problem PTA-DTRA is EXPTIME-hard. Thus we obtain the main result of this section which settles the computational complexity of the problem PTA-DTRA.

**Theorem 1.** *The PTA-DTRA problem is EXPTIME-complete.*

*Remark 2.* The main novelty for our product construction is that by adopting extended actions (i.e.  $\mathcal{T}_q$ ) and integrating them into enabling condition and probabilistic transition function, the product PTA can know which rule to use from the DTA upon any symbol to be read. This solves the problem that probabilistic jumps can lead to different locations, causing the usage of different rules from the DTA. Moreover, our product construction ensures EXPTIME-completeness of the problem.

## 5 The PTA-TRA Problem

In this section, we study the PTA-TRA problem where the input timed automaton needs not to be deterministic. In contrast to the deterministic case (which is shown to be decidable and EXPTIME-complete in the previous section), we show that the problem is undecidable.

**The Main Idea.** The main idea for the undecidability result is to reduce the universality problem of timed automata to the PTA-TRA problem. The universality problem over timed automata is well-known to be undecidable, as follows.

**Lemma 3** ([5, Theorem 5.2]). *Given a timed automaton over an alphabet  $\Sigma$  and an initial mode, the problem of deciding whether it accepts all time-divergent timed words w.r.t Büchi acceptance condition over  $\Sigma$  is undecidable.*

Although Lemma 3 is on Büchi acceptance condition, it holds also for Rabin acceptance condition since Rabin acceptance condition extends Büchi acceptance condition. Actually the two acceptance conditions are equivalent over timed automata (cf. [5, Theorem 3.20]). We also remark that Lemma 3 was originally for multiple initial modes, which can be mimicked by a single initial mode through aggregating all rules emitting from some initial mode as rules emitting from one initial mode.

Now we prove the undecidability result as follows. The proof idea is that we construct a PTA that can generate every time-divergent timed words with probability 1 by some time-divergent scheduler. Then the TRA accepts all time-divergent timed words iff the minimal probability that the PTA observes the TRA equals 1.

**Theorem 2.** *Given a PTA  $\mathcal{C}$  and a TRA  $\mathcal{A}$ , the problem to decide whether the minimal probability that  $\mathcal{C}$  observes  $\mathcal{A}$  (under a given initial mode) is equal to 1 is undecidable.*

*Proof (Proof Sketch).* Let  $\mathcal{A} = (Q, \Sigma, \mathcal{Y}, \Delta, \mathcal{F})$  be any TRA where the alphabet  $\Sigma = \{b_1, b_2, \dots, b_k\}$  and the initial mode is  $q_{start}$ . W.l.o.g, we consider that  $\Sigma \subseteq 2^{AP}$  for some finite set  $AP$ . This assumption is not restrictive since what  $b_i$ 's concretely are is irrelevant, while the only thing that matters is that  $\Sigma$  has  $k$  different symbols. We first construct the TRA  $\mathcal{A}' = (Q', \Sigma', \mathcal{Y}, \Delta', \mathcal{F})$  where  $Q' = Q \cup \{q_{init}\}$  for which  $q_{init}$  is a fresh mode,  $\Sigma' = \Sigma \cup \{b_0\}$  for which  $b_0$  is a fresh symbol and  $\Delta' = \Delta \cup \{\langle q_{init}, b_0, \mathbf{true}, \mathcal{Y}, q_{start} \rangle\}$ . Then we construct the PTA:

- $L := \Sigma', \ell^* := b_0, \mathcal{X} := \emptyset$  and  $Act := \Sigma$ ;
- $inv(b_i) := \mathbf{true}$  for  $b_i \in L$ ;
- $enab(b_i, b_j) := \mathbf{true}$  for  $b_i \in L$  and  $b_j \in Act$ ;
- $prob(b_i, b_j)$  is the Dirac distribution at  $(\emptyset, b_j)$  (i.e.,  $prob(b_i, b_j)(\emptyset, b_j) = 1$  and  $prob(b_i, b_j)(X, b) = 0$  whenever  $(X, b) \neq (\emptyset, b_j)$ ), for  $b_i \in L$  and  $b_j \in Act$ ;
- $\mathcal{L}(b_i) := b_i$  for  $b_i \in L$ .

Note that we allow no clocks in the construction since clocks are irrelevant for our result. Since we omit clocks, we also treat states (of  $\mathcal{C}'$ ) as single locations. One can prove that  $\mathcal{A}$  accepts all time-divergent timed words over  $\Sigma$  with initial mode  $q_{start}$  iff the minimal probability that  $\mathcal{C}'$  observes  $\mathcal{A}'$  with initial mode  $q_{init}$  equals 1.  $\square$

*Remark 3* Theorem 2 shows that the problem to qualitatively decide the minimal probability is undecidable. On the other hand, the decidability of the problem to decide maximum acceptance probabilities is left open.

## 6 Conclusion

In this paper, we studied the problem of model-checking PTAs against timed-automata specifications. We considered Rabin acceptance condition as the acceptance criterion. We first solved the problem with deterministic-timed-automata specifications through a product construction and proved that its computational complexity is EXPTIME-complete. Then we proved that the problem with general timed-automata specifications is undecidable through a reduction from the universality problem of timed automata.

A future direction is zone-based algorithms for Rabin acceptance condition. Another direction is to investigate timed-automata specifications with cost or reward. Besides, it is also interesting to explore model-checking PTAs against Metric Temporal Logic [19].



## 7 Related Works

Model-checking TAs or MDPs against omega-regular (dense-time) properties is well-studied (cf. [1, 20, 26], etc.). PTAs extend both TAs and MDPs with either probability or timing constraints, hence require new techniques for verification problems. On one hand, our technique extends techniques for MDPs (e.g. [26]) with timing constraints. On the other hand, our technique is incomparable to techniques for TAs since linear-time model checking of TAs focus mostly on proving decidability of temporal logic formulas (e.g. Metric Temporal logic [19–21]), while we prove that model-checking PTAs against TA-specifications is undecidable.

Model-checking probabilistic timed models against linear dense-time properties are mostly considered for continuous-time Markov processes (CTMPs). First, Donatelli *et al.* [22] proved an expressibility result that the class of linear dense-time properties encoded by DTAs is not subsumed by branching-time properties. They also demonstrated an efficient algorithm for verifying continuous-time Markov chains [22] against one-clock DTAs. Then various results on verifying CTMPs are obtained for specifications through DTAs and general timed automata (cf. e.g. [22, 24, 25, 27–29]). The fundamental difference between CTMPs and PTAs is that the former assign probability distributions to time elapses, while the latter treat time-elapses as pure nondeterminism. As a consequence, the techniques for CTMPs cannot be applied to PTAs.

For PTAs, the only relevant result is by Sproston [18] who developed an approach for verifying PTAs against deterministic discrete-time omega-regular automata by a similar product construction. Our results extend his result in two ways. First, our product construction has the extra ability to tackle timing constraints from the DTA. The extension is nontrivial since it needs to resolve the integration between randomness (from the PTA) and timing constraints (from the DTA), and still ensures the EXPTIME-completeness of the problem, matching the computational complexity in the discrete-time case [18]. Second, we have proved an undecidability result in the case of general nondeterministic timed automata, thus extending [18] with nondeterminism.

**Acknowledgements.** This work has been supported by the National Natural Science Foundation of China (Grants 61761136011, 61532019, 61472473, 61772038, 61272160). We also thank anonymous reviewers for detailed comments.

## References

1. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
2. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. *Formal Methods Syst. Des.* **43**(2), 164–190 (2013)
3. Beauquier, D.: On probabilistic timed automata. *Theor. Comput. Sci.* **292**(1), 65–84 (2003)
4. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. *Theor. Comput. Sci.* **282**(1), 101–150 (2002)

5. Alur, R., Dill, D.L.: A theory of timed automata. *Theor. Comput. Sci.* **126**(2), 183–235 (1994)
6. Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York (1994)
7. Jensen, H.E.: Model checking probabilistic real time systems. In: 7th Nordic Workshop on Programming Theory, Chalmers University of Technology, pp. 247–261 (1996)
8. André, É., Fribourg, L., Sproston, J.: An extension of the inverse method to probabilistic timed automata. *Formal Methods Syst. Des.* **42**(2), 119–145 (2013)
9. Kwiatkowska, M.Z., Norman, G., Sproston, J., Wang, F.: Symbolic model checking for probabilistic timed automata. *Inf. Comput.* **205**(7), 1027–1077 (2007)
10. Kwiatkowska, M.Z., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: Ouaknine, J., Vaandrager, F.W. (eds.) FORMATS 2009. LNCS, vol. 5813, pp. 212–227. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-04368-0\\_17](https://doi.org/10.1007/978-3-642-04368-0_17)
11. Jovanović, A., Kwiatkowska, M., Norman, G.: Symbolic minimum expected time controller synthesis for probabilistic timed automata. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 140–155. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-22975-1\\_10](https://doi.org/10.1007/978-3-319-22975-1_10)
12. Jurdziński, M., Kwiatkowska, M., Norman, G., Trivedi, A.: Concavely-priced probabilistic timed automata. In: Bravetti, M., Zavattaro, G. (eds.) CONCUR 2009. LNCS, vol. 5710, pp. 415–430. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-04081-8\\_28](https://doi.org/10.1007/978-3-642-04081-8_28)
13. Kwiatkowska, M.Z., Norman, G., Parker, D., Sproston, J.: Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods Syst. Des.* **29**(1), 33–78 (2006)
14. Berendsen, J., Chen, T., Jansen, D.N.: Undecidability of cost-bounded reachability in priced probabilistic timed automata. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 128–137. Springer, Heidelberg (2009). [https://doi.org/10.1007/978-3-642-02017-9\\_16](https://doi.org/10.1007/978-3-642-02017-9_16)
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
16. Laroussinie, F., Sproston, J.: State explosion in almost-sure probabilistic reachability. *Inf. Process. Lett.* **102**(6), 236–241 (2007)
17. Jurdzinski, M., Sproston, J., Laroussinie, F.: Model checking probabilistic timed automata with one or two clocks. *LMCS* **4**(3), 1–28 (2008)
18. Sproston, J.: Discrete-time verification and control for probabilistic rectangular hybrid automata. In: QEST, pp. 79–88 (2011)
19. Koymans, R.: Specifying real-time properties with metric temporal logic. *Real Time Syst.* **2**(4), 255–299 (1990)
20. Ouaknine, J., Worrell, J.: On the decidability of metric temporal logic. In: LICS, pp. 188–197 (2005)
21. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. *J. ACM* **43**(1), 116–146 (1996)
22. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with  $\text{CSL}^{\wedge}\{TA\}$ . *IEEE Trans. Software Eng.* **35**(2), 224–240 (2009)
23. Fu, H., Li, Y., Li, J.: Verifying probabilistic timed automata against omega-regular dense-time properties. *CoRR* abs/1712.00275 (2017)

24. Chen, T., Han, T., Katoen, J., Mereacre, A.: Model checking of continuous-time Markov chains against timed automata specifications. *Log. Methods Comput. Sci.* **7**(1), 1–34 (2011)
25. Fu, H.: Approximating acceptance probabilities of CTMC-paths on multi-clock deterministic timed automata. In: HSCC, pp. 323–332 (2013)
26. Vardi, M.Y.: Probabilistic linear-time model checking: an overview of the automata-theoretic approach. In: Katoen, J.-P. (ed.) ARTS 1999. LNCS, vol. 1601, pp. 265–276. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48778-6\\_16](https://doi.org/10.1007/3-540-48778-6_16)
27. Brázdil, T., Krcál, J., Kretínský, J., Kucera, A., Reháč, V.: Measuring performance of continuous-time stochastic processes using timed automata. In: HSCC, pp. 33–42 (2011)
28. Barbot, B., Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: Efficient CTMC model checking of linear real-time objectives. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 128–142. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-19835-9\\_12](https://doi.org/10.1007/978-3-642-19835-9_12)
29. Bortolussi, L., Lanciani, R.: Fluid model checking of timed properties. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 172–188. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-22975-1\\_12](https://doi.org/10.1007/978-3-319-22975-1_12)



# Incremental Verification of Parametric and Reconfigurable Markov Chains

Paul Gainer<sup>(✉)</sup>, Ernst Moritz Hahn<sup>(✉)</sup>, and Sven Schewe<sup>(✉)</sup>

University of Liverpool, Liverpool, UK  
{P.Gainer,E.M.Hahn,Sven.Schewe}@liverpool.ac.uk

**Abstract.** The analysis of parametrised systems is a growing field in verification, but the analysis of parametrised probabilistic systems is still in its infancy. This is partly because it is much harder: while there are beautiful cut-off results for non-stochastic systems that allow to focus only on small instances, there is little hope that such approaches extend to the quantitative analysis of probabilistic systems, as the probabilities depend on the size of a system. The unicorn would be an automatic transformation of a parametrised system into a formula, which allows to plot, say, the likelihood to reach a goal or the expected costs to do so, against the parameters of a system. While such analysis exists for narrow classes of systems, such as waiting queues, we aim both lower—stepwise exploring the parameter space—and higher—considering general systems.

The novelty is to heavily exploit the similarity between instances of parametrised systems. When the parameter grows, the system for the smaller parameter is, broadly speaking, present in the larger system. We use this observation to guide the elegant state-elimination method for parametric Markov chains in such a way, that the model transformations will start with those parts of the system that are stable under increasing the parameter. We argue that this can lead to a very cheap iterative way to analyse parametric systems, show how this approach extends to reconfigurable systems, and demonstrate on two benchmarks that this approach scales.

## 1 Introduction

Probabilistic systems are everywhere, and their analysis can be quite challenging. Challenges, however, come in many flavours. They range from theoretical questions, such as decidability and complexity, through algorithms design and tool development, to the application of parametric systems. This paper is motivated by the latter, but melds the different flavours together.

We take our motivation from the first author's work on biologically inspired synchronisation protocols [8, 9]. This application leaning work faced the problem of exploring a parameter space for a family of network coordination protocols, where interacting nodes achieve consensus on their local clocks by imitating the behaviour of fireflies [22]. Global clock synchronisation emerges from local interactions between the nodes, whose behaviour is that of coupled limit-cycle

oscillators. The method used was the same that we have seen applied by several practitioners from engineering and biology: adjust the parameters, produce a model, and use a tool like ePMC/IscasMC [12], PRISM [20], or Storm [6] to analyse it.

In the case of the synchronisation protocols, the parameters investigated were typical of those considered when evaluating the emergence of synchronisation in a network of connected nodes: the number of nodes forming the network, the granularity of the model (discrete length of an oscillation cycle), the strength of coupling between the oscillators, the likelihood of interactions between nodes being inhibited by some external factor, for instance message loss in a communication medium, and the length of the refractory period, an initial period in the oscillation cycle of a node where interactions with other nodes are ignored.

The reason to explore the parameter space can be manifold. Depending on the application, one might simply want to obtain a feeling of how the parameters impact on the behaviour. Another motivation is to see how the model behaves, compare it with observations, and adjust it when it fails to comply. Regardless of the reason to adjust the parameter, the changes often lead to very similar models.

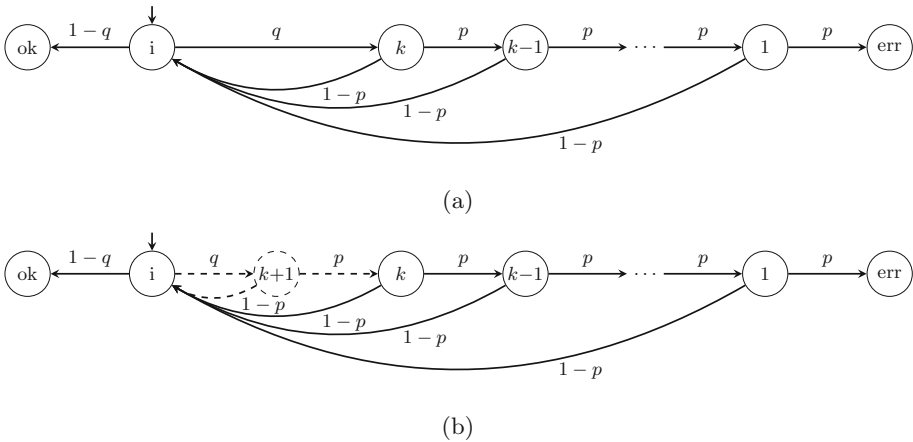
Now, if we want to analyse hundreds of similar models, then it becomes paramount to re-use as much of the analysis as possible. With this in mind, we have selected model checking techniques for safety and reachability properties of Markov chains that build on repeated state elimination [11] as the backbone of our verification technique. State elimination is a technique that successively changes the model by removing states. It works like the transformation from finite automata to regular expressions: a state is removed, and the new structure has all successors of this state as (potentially new) successors of the predecessors of this state, with the respectively adjusted probabilities (and, if applicable, expected rewards).

If these models are changed in shape and size when playing with the parameters, then these changes tend to be smooth: small changes of the parameters lead to small changes of the model. Moreover, the areas that change—and, sequentially, the areas that do *not* change—are usually quite easy to predict, be it by an automated analysis of the model or by the knowledge of the expert playing with her model, who would know full well which parts do or do not change when increasing a parameter. These insights inform the order in which the states are eliminated.

When, say, the increase of a parameter allows for re-using all elimination steps but the last two or three, then repeating the analysis is quite cheap. Luckily, this is typically the case in structured models, e.g. those who take a chain-, ring-, or tree-like shape that can be inductively defined. As a running example of a structured model we consider the Zeroconf protocol [3] for the autonomous configuration of multiple hosts in a network with unique IP addresses (Fig. 1). A host that joins the network selects an address uniformly at random from  $a$  available addresses. If the network consists of  $h$  hosts, then the probability that the selected address is already in use is  $q = \frac{h}{a}$ .

The protocol then checks  $n$  times if the selected address is already in use by sending a request to the network. If the address is fresh (which happens with a probability of  $1 - q$ ), none of these tests will fail, and the address will be classed as new. If the address is already in use (which happens with a probability of  $q$ ), then each test is faulty: collisions go undetected with some likelihood  $p$  due to message loss and time-outs. When a collision is detected (which happens with a likelihood of  $1 - p$  in each attempt, provided that a collision exists), then the host restarts the process. If a collision has gone undetected after  $n$  attempts, then the host will incorrectly assume that its address is unique.

While the family of Zeroconf protocols is also parameterised in the transition probabilities, we are mostly interested in their parametrisation in the structure of the model. Figures 1a and b show the models for  $n = k$  and  $n = k + 1$ , respectively, successive checks after each selection of an IP. These two models are quite similar: they structurally differ only in the introduction of a single state, and the transitions that come with this additional state. If we are interested in calculating the function that represents the probability of reaching the state *err* in both models, where this function is given in terms of individual rational functions that label the transitions, then the structural similarities allow us to re-use the intermediate terms obtained from the evaluation for  $n = k$  when evaluating for  $n = k + 1$ .



**Fig. 1.** The zeroconf protocol for  $n = k$  (a) and  $n = k + 1$  (b).

The structure of the paper is as follows. We begin by comparing our work to similar approaches in Sect. 2. In Sect. 3, we introduce the novel algorithms for the analysis of reconfigured models. We then evaluate our approach on two different types of parametric models which are discussed in Sect. 4. Section 5 concludes the paper and outlines future work.

## 2 Related Work

Our work builds on previous results in the area of parametric Markov model checking and incremental runtime verification of stochastic systems.

Daws [4] considered Markov chains, which are parametric in the transition probabilities, but not in their graph structure. He introduced an algorithm to calculate the function that represents the probability of reaching a set of target states for all well-defined evaluations for a parametric Markov chain. For this, he interprets the Markov chain under consideration as a finite automaton, in which transitions are labelled with symbols that correspond to rational numbers or variables. He then uses state elimination [13] to obtain a regular expression for the language of the automaton. Evaluating these regular expressions into rational functions yields the probability of reaching the target states. One limiting factor of this approach is that the complete regular expression has to be stored in memory.

Hahn et al. introduced [11] and implemented [10] a simplification and refinement of Daws' algorithm. Instead of using regular expressions, they store rational functions directly. This has the advantage that possible simplifications of these functions, such as cancellation of common factors, can be applied on the fly. This allows memory to be saved. It also provides a more concise representation of the values computed to the user. They have also extended the scope of the approach from reachability, to additionally handle accumulated reward properties. Several works from RWTH Aachen have followed up on solution methods for parametric Markov chains [5, 14, 23]. This type of parametric model checking has been used in [2] to build a model-repair framework for stochastic systems and in [15–17] to reason about the robustness of robot controllers against sensor errors.

Our paper borrows some ideas from the work of Kwiatkowska et al. [21]. Their work considers MDPs that are labelled with parametric transition probabilities. The authors do not aim to compute a closed-form function that represents properties of a model, but rather at accelerating the computation of results for individual instantiations of parameter values. Rather than state elimination, they use value iteration and other methods to evaluate the model for certain parameter values. In doing so, they can for instance, re-use computations for different instantiations of parameters that only depend on the graph structure of the model that remains unchanged for different instantiations.

We also take inspiration from Forejt et al. [7], where the role of parameters is different. Forejt et al. describe a policy iteration-based technique to evaluate parametric MDPs. While they also considered parameters in [7] that can influence the model structure, they would exploit similarities to inform the search for the policy when moving from one parameter value to the next. The repeatedly called model checking of Markov chains, on the other hand, is not parametric. Our approach is therefore completely orthogonal, as we focus on the analysis of Markov chains. In more detail, Forejt et al. [7] would use an incremental approach to find a good starting point for a policy iteration approach for MDPs. The intuition there is that an optimal policy is likely to be good—if not optimal—on the shared part of an MDP that grows with a parameter. This approach has

the potential to find the policy in less steps, as less noise disturbs the search in smaller MDPs, but its main promise is to provide an excellent oracle for a starting policy. Moreover, in the lucky instances where the policy is stable, it can also happen that there is a part of the Markov chain, obtained by using a policy that builds on a smaller parameter, that is outside of the cone of influence of the changes to the model. In this case, not only the policy, but also its evaluation is stable under the parameter change.

### 3 Algorithms

We first describe the state elimination method of Hahn [11] for parametric Markov Chains (PMCs), and then introduce an algorithm that substantially reduces the cost of recomputation of the parametric reachability probability for a reconfigured PMC. First we give some general definitions. Given a function  $f$  we denote the domain of  $f$  by  $\text{Dom}(f)$ . We use the notation  $f \oplus f' = f \upharpoonright_{\text{Dom}(f) \setminus \text{Dom}(f')} \cup f'$  to denote the overriding union of  $f$  and  $f'$ . Let  $V = \{v_1, \dots, v_n\}$  denote a set of variables over  $\mathbb{R}$ . A *polynomial*  $g$  over  $V$  is a sum of monomials

$$g(v_1, \dots, v_n) = \sum_{i_1, \dots, i_n} a_{i_1, \dots, i_n} v_1^{i_1} \dots v_n^{i_n},$$

where each  $i_j \in \mathbb{N}$  and each  $a_{i_1, \dots, i_n} \in \mathbb{R}$ . A *rational function*  $f$  over a set of variables  $V$  is a fraction  $f(v_1, \dots, v_n) = \frac{f_1(v_1, \dots, v_n)}{f_2(v_1, \dots, v_n)}$  of two polynomials  $f_1, f_2$  over  $V$ . We denote the set of rational functions from  $V$  to  $\mathbb{R}$  by  $\mathcal{F}_V$ .

**Definition 1.** A parametric Markov chain (PMC) is a tuple  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$ , where  $\mathcal{S}$  is a finite set of states,  $s_0$  is the initial state,  $V = \{v_1, \dots, v_n\}$  is a finite set of parameters, and  $\mathbf{P}$  is the probability matrix  $\mathbf{P} : \mathcal{S} \times \mathcal{S} \rightarrow \mathcal{F}_V$ .

A path  $\omega$  of a PMC  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$  is a non-empty finite, or infinite, sequence  $s_0, s_1, s_2, \dots$  where  $s_i \in \mathcal{S}$  and  $\mathbf{P}(s_i, s_{i+1}) > 0$  for  $i \geq 0$ . We denote the  $i^{\text{th}}$  state of  $\omega$  by  $\omega[i]$ , the set of all paths starting at state  $s$  by  $\text{Paths}(s)$ , and the set of all finite paths starting in  $s$  by  $\text{Paths}_f(s)$ . For a finite path  $\omega_f \in \text{Paths}_f(s)$  the cylinder set of  $\omega_f$  is the set of all infinite paths in  $\text{Paths}(s)$  that share the prefix  $\omega_f$ . The probability of taking a finite path  $s_0, s_1, \dots, s_n \in \text{Paths}_f(s_0)$  is given by  $\prod_{i=1}^n \mathbf{P}(s_{i-1}, s_i)$ . This measure over finite paths can be extended to a probability measure  $\text{Pr}_s$  over the set of infinite paths  $\text{Paths}(s)$ , where the smallest  $\sigma$ -algebra over  $\text{Paths}(s)$  is the smallest set containing all cylinder sets for paths in  $\text{Paths}_f(s)$ . For a detailed description of the construction of the probability measure we refer the reader to [18].

**Definition 2.** Given a PMC  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$ , the underlying graph of  $\mathcal{D}$  is given by  $\mathcal{G}_{\mathcal{D}} = (\mathcal{S}, E)$  where  $E = \{(s, s') \mid \mathbf{P}(s, s') > 0\}$ .

Given a state  $s$ , we denote the set of all immediate predecessors and successors of  $s$  in the underlying graph of  $\mathcal{D}$  by  $\text{pre}_{\mathcal{D}}(s)$  and  $\text{post}_{\mathcal{D}}(s)$ , respectively, and we define the *neighbourhood* of  $s$  as  $\text{Neigh}(s) = s \cup \text{pre}_{\mathcal{D}}(s) \cup \text{post}_{\mathcal{D}}(s)$ . We write  $\text{reach}_{\mathcal{D}}(s, s')$  if  $s'$  is reachable from  $s$  in the underlying graph of  $\mathcal{D}$ .



**Algorithm 1.** State Elimination

---

```

1: procedure STATEELIMINATION( $\mathcal{D}, s_e$ )
2:   requires: A PMC  $\mathcal{D}$  and  $s_e$ , a state to eliminate in  $\mathcal{D}$ .
3:   for all  $(s_1, s_2) \in \text{pre}_{\mathcal{D}}(s_e) \times \text{post}_{\mathcal{D}}(s_e)$  do
4:      $\mathbf{P}(s_1, s_2) \leftarrow \mathbf{P}(s_1, s_2) + \mathbf{P}(s_1, s_e) \frac{1}{1 - \mathbf{P}(s_e, s_e)} \mathbf{P}(s_e, s_2)$ 
5:   end for
6:   Eliminate( $\mathcal{D}, s_e$ ) // remove  $s_e$  and incident transitions from  $\mathcal{D}$ 
7:   return  $\mathcal{D}$ 
8: end procedure

```

---

**3.1 State Elimination**

The algorithm of Hahn [11] proceeds as follows, where the input is a PMC  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$  and a set of target states  $T \subset \mathcal{S}$ . Initially, preprocessing is applied and without loss of generality all outgoing transitions from states in  $T$  are removed and a new state  $s_t$  is introduced such that  $\mathbf{P}(t, s_t) = 1$  for all  $t \in T$ . All states  $s$ , where  $s$  is unreachable from the initial state or  $T$  is unreachable from  $s$ , are then removed along with all incident transitions. A state  $s_e \in \mathcal{S} \setminus \{s_0, s_t\}$  is then chosen for elimination and Algorithm 1 is applied. Firstly, for every pair  $(s_1, s_2) \in \text{pre}_{\mathcal{D}}(s_e) \times \text{post}_{\mathcal{D}}(s_e)$ , the existing probability  $\mathbf{P}(s_1, s_2)$  is incremented by the probability of reaching  $s_2$  from  $s_1$  via  $s_e$ . The state and any incident transitions are then eliminated from  $\mathcal{D}$ . This procedure is repeated until only  $s_0$  and  $s_t$  remain, and the probability of reaching  $T$  from  $s_0$  is then given by  $\mathbf{P}(s_0, s_t)$ .

**3.2 Reconfiguration**

Recall that we are interested in the re-use of information when recalculating reachability for a reconfigured PMC. We can do this by choosing the order in which we eliminate states in the original PMC. The general idea is that, if the set of states where structural changes might occur is known a priori, then we can apply state elimination to all other states first. We say that states where structural changes might occur are *volatile* states.

**Definition 3.** A volatile parametric Markov chain (VPMC) is a tuple  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \text{Vol})$  where  $(\mathcal{S}, s_0, \mathbf{P}, V)$  is a PMC and  $\text{Vol} \subseteq \mathcal{S}$  is a set of volatile states for  $\mathcal{D}$ .

Given a VPMC  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \text{Vol})$ , we can define an *elimination ordering* for  $\mathcal{D}$  as a bijection  $\prec_{\mathcal{D}}: \mathcal{S} \rightarrow \{1, \dots, |\mathcal{S}|\}$  that defines an ordering for the elimination of states in  $\mathcal{S}$ , such that  $\prec_{\mathcal{D}}(s) < \prec_{\mathcal{D}}(s')$  holds for all  $s \in \mathcal{S} \setminus \text{Vol}$ ,  $s' \in \text{Vol}$ , where  $\prec_{\mathcal{D}}(s) < \prec_{\mathcal{D}}(s')$  indicates that  $s$  is eliminated before  $s'$ . Observe that a volatile state in  $\mathcal{D}$  is only eliminated after all non-volatile states.

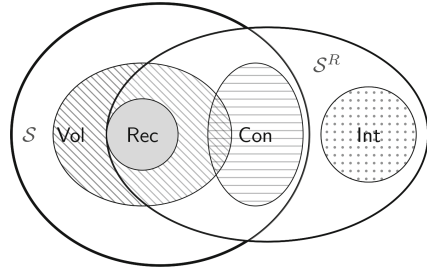
**Definition 4.** A reconfiguration for a VPMC  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \text{Vol})$  is a PMC  $\mathcal{D}^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, V)$ , where  $\mathcal{S}^R$  is a set of states with  $\mathcal{S}^R \cap \mathcal{S} \neq \{s_0\}$ ,  $s_0$  and

$V$  are the initial state and the finite set of parameters for  $\mathcal{D}$ . The reconfigured probability matrix  $\mathbf{P}^R$  is a total function  $\mathbf{P}^R : \mathcal{S}^R \times \mathcal{S}^R \rightarrow \mathcal{F}_V$  such that, for all  $s, s' \in \mathcal{S}^R$  where  $\mathbf{P}(s, s')$  is defined,  $\mathbf{P}(s, s') \neq \mathbf{P}^R(s, s')$  implies  $s, s' \in \text{Vol}$ .

Given a VPMC  $\mathcal{D}$  and a reconfiguration  $\mathcal{D}^R$  for  $\mathcal{D}$  we say that a state  $s$  in  $\mathcal{S}$  is *consistent* in  $\mathcal{D}^R$  if  $s$  is also in  $\mathcal{S}^R$ , and the set of all predecessors and successors of  $s$  remains unchanged in  $\mathcal{D}^R$  (that is  $\text{pre}_{\mathcal{D}}(s) = \text{pre}_{\mathcal{D}^R}(s)$ ,  $\text{post}_{\mathcal{D}}(s) = \text{post}_{\mathcal{D}^R}(s)$ ,  $\mathbf{P}(s_1, s) = \mathbf{P}^R(s_1, s)$  for every  $s_1 \in \text{pre}_{\mathcal{D}}(s)$ , and  $\mathbf{P}(s, s_2) = \mathbf{P}^R(s, s_2)$  for every  $s_2 \in \text{post}_{\mathcal{D}}(s)$ ). We say that a state  $s$  in  $\mathcal{S}$  is *reconfigured* in  $\mathcal{D}^R$  if  $s$  is also in  $\mathcal{S}^R$  and  $s$  is not consistent. Finally, we say that a state  $s$  in  $\mathcal{S}^R$  is *introduced* in  $\mathcal{D}^R$  if  $s$  is neither consistent nor reconfigured. We denote the sets of all consistent, reconfigured, and introduced states by  $\text{Con}(\mathcal{D}, \mathcal{D}^R)$ ,  $\text{Rec}(\mathcal{D}, \mathcal{D}^R)$ , and  $\text{Int}(\mathcal{D}, \mathcal{D}^R)$ , respectively. Figure 2 shows the consistent, reconfigured, and introduced states for  $\mathcal{D}$  and  $\mathcal{D}^R$ .

Algorithm 2 computes the parametric reachability probability of some target state in a VPMC  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \text{Vol})$  for a given elimination ordering for  $\mathcal{D}$ . Observe that we compute the reachability probability with respect to a single target state. The reachability of a set of target states can be computed by first removing all outgoing transitions from existing target states, and then introducing a new target state to which a transition is taken from any existing target state with probability 1. We introduce a new initial state to the model, from which a transition is taken to the original initial state with probability 1. The algorithm computes a partial probability matrix  $\mathbf{P}'$ , initialised as a zero matrix, that stores the probability of reaching  $s_2$  from  $s_1$  via any eliminated non-volatile state, where  $s_1, s_2$  are either volatile states, the initial state, or the target state. It also computes an *elimination map*  $m_{\mathcal{D}}^{\text{Vol}}$ , a function mapping tuples of the form  $(s_e, s_1, s_2)$ , where  $s_e$  is an eliminated volatile state and  $s_1, s_2$  are either volatile states, the initial state, or the target state, to the value computed during state elimination for the probability of reaching  $s_2$  from  $s_1$  via  $s_e$ . We are only interested in transitions between volatile states, the initial state, or the target state, since all non-volatile states in any reconfiguration of  $\mathcal{D}$  will be eliminated first. Computed values for transitions to or from these states therefore serve no purpose once they have been eliminated.

Given a reconfiguration  $\mathcal{D}^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, V)$  for  $\mathcal{D}$ , an elimination ordering for  $\mathcal{D}$ , and the partial probability matrix and mapping computed using Algorithm 2, Algorithm 3 computes the parametric reachability probability for  $\mathcal{D}^R$  as follows. Firstly the set of all non-volatile states of  $\mathcal{D}$  and incident transitions are removed from  $\mathcal{D}^R$ , though state elimination itself does not occur. A set of *infected* states is then initialised to be the set of all states that are reconfigured



**Fig. 2.** Venn diagram showing the consistent, reconfigured, and introduced states for a VPMC  $\mathcal{D}$  and reconfiguration  $\mathcal{D}^R$ .

**Algorithm 2.** Parametric Reachability Probability for VPMC's

---

```

1: procedure PARAMETRICREACHABILITY( $\mathcal{D}$ ,  $\prec_{\mathcal{D}}$ ,  $s_t$ )
2:   requires: a target state  $s_t \in \mathcal{S}$ , and for all  $s \in \mathcal{S}$  it holds  $\text{reach}_{\mathcal{D}}(s_0, s)$  and
    $\text{reach}_{\mathcal{D}}(s, s_t)$ .
3:    $E \leftarrow \mathcal{S} \setminus \{s_0, s_t\}$  // states to be eliminated from  $\mathcal{D}$ 
4:    $\mathbf{P}' \leftarrow 0_{|\mathcal{S}|, |\mathcal{S}|}$  // partial probability matrix
5:    $m_{\mathcal{D}}^{\text{Vol}} \leftarrow \emptyset$  // elimination map
6:   while  $E \neq \emptyset$  do
7:      $s_e \leftarrow \arg \min \prec_{\mathcal{D}} \upharpoonright_E$ 
8:     for all  $(s_1, s_2) \in \text{pre}_{\mathcal{D}}(s_e) \times \text{post}_{\mathcal{D}}(s_e)$  do
9:        $p = \mathbf{P}(s_1, s_e) \frac{1}{1 - \mathbf{P}(s_e, s_e)} \mathbf{P}(s_e, s_2)$ 
10:      if  $s_1 \in \text{Vol} \cup \{s_0, s_t\}$  and  $s_2 \in \text{Vol} \cup \{s_0, s_t\}$  then
11:        if  $s_e \notin \text{Vol}$  then
12:           $\mathbf{P}'(s_1, s_2) \leftarrow \mathbf{P}'(s_1, s_2) + p$ 
13:        else
14:           $m_{\mathcal{D}}^{\text{Vol}} \leftarrow m_{\mathcal{D}}^{\text{Vol}} \oplus \{(s_e, s_1, s_2) \mapsto p\}$ 
15:        end if
16:      end if
17:       $\mathbf{P}(s_1, s_2) \leftarrow \mathbf{P}(s_1, s_2) + p$ 
18:    end for
19:    Eliminate( $\mathcal{D}$ ,  $s_e$ ) // remove  $s_e$  and incident transitions from  $\mathcal{D}$ 
20:     $E \leftarrow E \setminus \{s_e\}$ 
21:  end while
22:  return  $(\mathbf{P}(s_0, s_t), \mathbf{P}', m_{\mathcal{D}}^{\text{Vol}})$ 
23: end procedure

```

---

in  $\mathcal{D}^R$ . Then, for every other remaining state that is not introduced in  $\mathcal{D}^R$ , if that state or its neighbours are not infected we treat this state as a non-volatile state. That is, we update  $\mathbf{P}'$  with the corresponding values in  $m_{\mathcal{D}}^{\text{Vol}}$  and remove the state and its incident transitions without performing state elimination. If the state, or one of its neighbours, is infected then the probability matrix is updated such that all transitions to and from that state are augmented with the corresponding values in  $\mathbf{P}'$ . These entries are then removed from the mapping. Subsequently, state elimination (Algorithm 1) is applied, and the infected area is expanded to include the immediate neighbourhood of the eliminated state. Finally, state elimination is applied to the set of all remaining introduced states in  $\mathcal{D}^R$ .

*Example 1.* Consider again the Zeroconf models from Figs. 1a and b. Let  $Z_k = (\mathcal{S}, s_0, \mathbf{P}, V, \text{Vol})$  be a VPMC for  $n = k$ , such that  $\mathcal{S} = \{1, \dots, k\} \cup \{s_0, \text{i}, \text{err}\}$ ,  $V = \{p, q\}$ , and  $\text{Vol} = \{\text{i}, k\}$ . We are interested in the parametric reachability probability of the state  $\text{err}$ . Note that preprocessing removes the state  $\text{ok}$  from  $Z_k$  since  $\text{reach}_{Z_k}(\text{ok}, \text{err})$  does not hold. Now define  $\prec_{Z_k} = \{1 \mapsto 1, 2 \mapsto 2, \dots, k \mapsto k, \text{i} \mapsto k+1\}$  to be an elimination ordering for  $Z_k$ . State elimination then proceeds according to  $\prec_{Z_k}$ , and after the first  $k-1$  states have been eliminated we have

**Algorithm 3.** Parametric Reachability Probability for reconfigured VMPC

---

```

1: procedure RECONFIGUREDPARAMETRICREACHABILITY( $\mathcal{D}$ ,  $\mathcal{D}^R$ ,  $\prec_{\mathcal{D}}$ ,  $\mathbf{P}'$ ,  $m_{\mathcal{D}}^{\text{Vol}}$ ,  $s_t$ )
2:   requires: absorbing target state  $s_t$  such that  $s_t \in \mathcal{S}$  and  $s_t \in \mathcal{S}^R$ , for all  $s \in \mathcal{S}$  it
   holds  $\text{reach}_{\mathcal{D}^R}(s_0, s)$  and  $\text{reach}_{\mathcal{D}^R}(s, s_t)$ , and for all  $s' \in \mathcal{S}^R$  it holds  $\text{reach}_{\mathcal{D}^R}(s_0, s')$ 
   and  $\text{reach}_{\mathcal{D}^R}(s', s_t)$ .
3:    $M \leftarrow (\text{Vol} \cap \mathcal{S}^R) \cup \{s_0, s_t\}$ 
4:    $\text{Elim} \leftarrow \text{Con}(\mathcal{D}, \mathcal{D}^R) \setminus M$ 
5:    $\text{Eliminate}(\mathcal{D}^R, \text{Elim})$  // remove all  $s_e \in \text{Elim}$  and incident transitions from  $\mathcal{D}$ 
6:    $\text{Elim} \leftarrow \text{Vol} \cap \mathcal{S}^R$ 
7:    $\text{Infected} \leftarrow \text{Rec}(\mathcal{D}, \mathcal{D}^R)$ 
8:    $\mathbf{P}^R(s_0, s_t) = \mathbf{P}'(s_0, s_t)$ 
9:   while  $\text{Elim} \neq \emptyset$  do
10:      $s_e \leftarrow \arg \min \prec_{\mathcal{D}} \upharpoonright_{\text{Elim}}$ 
11:     if  $\text{Infected} \cap \text{Neigh}(s_e) = \emptyset$  then
12:       for all  $(s'_e, s_1, s_2) \in \text{Dom}(m_{\mathcal{D}}^{\text{Vol}} \upharpoonright_{\{s_e\} \times M^2})$  do
13:          $\mathbf{P}'(s_1, s_2) \leftarrow \mathbf{P}'(s_1, s_2) + m_{\mathcal{D}}^{\text{Vol}}(s'_e, s_1, s_2)$ 
14:       end for
15:        $\text{Eliminate}(\mathcal{D}^R, s_e)$  // remove  $s_e$  and incident transitions from  $\mathcal{D}^R$ 
16:     else
17:       for all  $\{(s_1, s_2) \in \mathcal{S}^R \times \mathcal{S}^R \mid s_1 = s_e \text{ or } s_2 = s_e\}$  do
18:          $\mathbf{P}^R(s_1, s_2) \leftarrow \mathbf{P}^R(s_1, s_2) + \mathbf{P}'(s_1, s_2)$ 
19:          $\mathbf{P}'(s_1, s_2) \leftarrow 0$ 
20:       end for
21:        $\mathcal{D}^R \leftarrow \text{StateElimination}(\mathcal{D}^R, s_e)$ 
22:        $\text{Infected} \leftarrow \text{Infected} \cup \text{Neigh}(s_e)$ 
23:     end if
24:      $\text{Elim} \leftarrow \text{Elim} \setminus \{s_e\}$ 
25:   end while
26:   for all  $s_e \in \text{Int}(\mathcal{D}, \mathcal{D}^R)$  do
27:      $\mathcal{D}^R \leftarrow \text{StateElimination}(\mathcal{D}^R, s_e)$ 
28:   end for
29:   return  $\mathbf{P}^R(s_0, s_t)$ 
30: end procedure

```

---

$$\mathbf{P}'(k, \text{err}) = p^k, \quad \mathbf{P}'(k, i) = \sum_{j=1}^{k-1} (p^j - p^{j+1}).$$

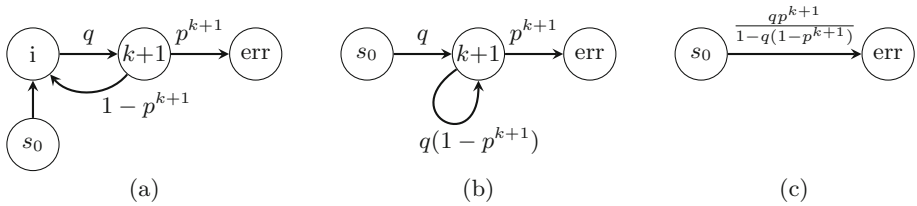
Eliminating the remaining volatile states  $k$  and  $i$  then yields

$$m_{Z_k}^{\text{Vol}} = \{(k, i, \text{err}) \mapsto qp^k, (k, i, i) \mapsto q(1 - p^k), (i, s_0, \text{err}) \mapsto \frac{qp^k}{1 - q(1 - p^k)}\}.$$

Now let  $Z_{k+1} = (\mathcal{S}^R, s_0, \mathbf{P}^R, V)$  be a reconfiguration for  $Z_k$  such that  $\mathcal{S}^R = \mathcal{S} \cup \{k+1\}$ . We then have  $\text{Con}(Z_k, Z_{k+1}) = \{1 \dots k-1\} \cup \{s_0, \text{err}\}$ ,  $\text{Rec}(Z_k, Z_{k+1}) = \{k, i\}$ , and  $\text{Int}(Z_k, Z_{k+1}) = \{k+1\}$ . First, all states  $1, \dots, k-1$  and their incident transitions are simply eliminated from  $Z_{k+1}$ , and the infected set is initialised to be  $\{k, i\}$ . Since  $k$  is already infected we update the probability matrix as follows,

$$\begin{aligned}
 \mathbf{P}^R(k, \text{err}) &\leftarrow \mathbf{P}^R(k, \text{err}) + \mathbf{P}'(k, \text{err}) \\
 &\leftarrow 0 + p^k = p^k, \\
 \mathbf{P}^R(k, i) &\leftarrow \mathbf{P}^R(k, i) + \mathbf{P}'(k, i) \\
 &\leftarrow (1-p) + \sum_{j=1}^{k-1} (p^j - p^{j+1}) = \sum_{j=0}^{k-1} (p^j - p^{j+1}) = 1 - p^k.
 \end{aligned}$$

State elimination is then applied to state  $k$  and the corresponding entries in  $\mathbf{P}'$  are set to zero. The state of the model after this step is shown in Fig. 3a. State  $i$  is also infected, but this time there are no corresponding non-zero values in  $\mathbf{P}'$ . State elimination is then applied to state  $i$  resulting in the model shown in Fig. 3b. Finally, state elimination is applied to the single introduced state  $k+1$ , resulting in the model shown in Fig. 3c, and the algorithm terminates.



**Fig. 3.**  $Z_{k+1}$  after the elimination of states  $k$  (a),  $i$  (b), and  $k+1$  (c).

### 3.3 Correctness

The correctness of the approach follows as an easy corollary from the correctness of Hahn’s general state elimination approach [11]. We outline the simple inductive argument, starting with the first parameter under consideration—which serves as the induction basis—and then look at incrementing the parameter value—which serves as the induction step.

For the induction basis, the first parameter considered, there is really nothing to show: we would merely choose a particular order in which states are eliminated, and the correctness of Hahn’s state-elimination approach does not depend on the order in which states are eliminated.

For the induction step, consider that we have an order for one parameter value, and that we have an execution of the state elimination along this given order  $<_o$ . Our approach then builds a new order for the next parameter value. The new order  $<_n$  is quite closely linked to the old order  $<_o$ , but for correctness, a very weak property suffices.

To prepare our argument, let us consider a set  $E$  of states with the following properties: the neighbourhood of  $E$  is the same in the Markov chains for the old and new parameter; the restriction of  $<_o$  and  $<_n$  to  $E$  define the same order; and  $E$  is the set of smallest states w.r.t.  $<_o$  and  $<_n$  ( $s \in E$  and ( $s' <_o s \vee s' <_n s$ ))

implies  $s' \in E$ ). In this case, the initial sequence of the first  $|E|$  reductions for the new Markov chain (along  $<_n$ ) are the same as the first  $|E|$  state eliminations along the old Markov chain (along  $<_o$ ). Consequently, these elimination steps can be re-used, rather than re-done.

In Algorithm 3 we require less: we still require that the neighbourhood of  $E$  is the same in the Markov chains for the old and new parameter and the restriction of  $<_o$  and  $<_n$  to  $E$  define the same order, but relax the third requirement to  $s \in E$  and  $(s' <_o s \vee s' <_n s)$  implies that  $s' \in E$  or  $s'$  is no neighbour of  $s$ . The result is the same: for the states in  $E$ , the  $|E|$  state eliminations for the new Markov chain (along  $<_n$ ) are the same as  $|E|$  state eliminations along the old Markov chain (along  $<_o$ ). Consequently, these elimination steps can be re-used.

### 3.4 Extension to Parametric Markov Reward Models

We now describe how we can extend the algorithms to PMCs annotated with rewards.

**Definition 5.** A *Parametric Markov Reward Model (PMRM)* is a tuple  $\mathcal{R} = (\mathcal{D}, r)$  where  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$  is a PMC and  $r : \mathcal{S} \rightarrow \mathcal{F}_V$  is the reward function.

The reward function labels states in  $\mathcal{R}$  with a rational function over  $V$  that corresponds to the reward that is gained if that state is visited. Given a PMRM  $\mathcal{R} = (\mathcal{D}, r)$  with  $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$ , we are interested in the *parametric expected accumulated reward* [19] until some target state  $s_t \in \mathcal{S}$  is reached. This is defined as the expectation of the random variable  $X^{\mathcal{R}} : \text{Paths}(s_0) \rightarrow \mathbb{R} \cup \{\infty\}$  over the infinite paths of  $\mathcal{R}$ . Given the set  $\omega_{s_t} = \{i \mid w[i] = s_t\}$  we define

$$X^{\mathcal{R}}(\omega) = \begin{cases} \infty & \text{if } \omega_{s_t} = \emptyset \\ \sum_{i=0}^{k-1} r(\omega[i]) & \text{otherwise, where } k = \min \omega_{s_t}, \end{cases}$$

and define the expectation of  $X^{\mathcal{R}}$  with respect to  $\text{Pr}_{s_0}$  as

$$E[X^{\mathcal{R}}] = \sum_{\omega \in \text{Paths}(s_0)} X^{\mathcal{R}}(\omega) \text{Pr}_{s_0}(\omega).$$

We extend our notion of volatility to PMRMs as follows. We say that a state is volatile if structural changes might occur in that state *or* if the reward labelling that state might change. Because of space limitations we omit the full definitions for volatile PMRMs, but the constructions are straightforward. Algorithms 1 to 3 are extended to incorporate rewards. For Algorithm 1, in addition to updating the probability matrix for the elimination of some state  $s_e$ , we also update the reward function as follows,

$$r(s_1) \leftarrow r(s_1) + \mathbf{P}(s_1, s_e) \frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)} r(s_e).$$

The updated value for  $r(s_1)$  reflects the reward that would be accumulated if a transition would be taken from  $s_1$  to  $s_e$ , where the expected number of self

transitions would be  $\frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)}$ . Algorithm 2 then constructs additional mappings to record these computed expected reward values, which are then used for reconfiguration in Algorithm 3.

## 4 Case Studies

We provide a prototypical implementation<sup>1</sup> of the technique and define the metric that we will use for the evaluation of different models to be the total number of arithmetic operations performed for the elimination of all states in a model. Our implementation serves only to illustrate the potential of the method, and we will integrate the technique into the probabilistic model checker ePMC [12].

Due to space limitations we restrict our analysis to two classes of models. Firstly we consider the family of Zeroconf protocols described in Sect. 1, and secondly we consider a family of models used for the analysis of biologically inspired firefly synchronisation protocols—the class of protocols that inspired this work.

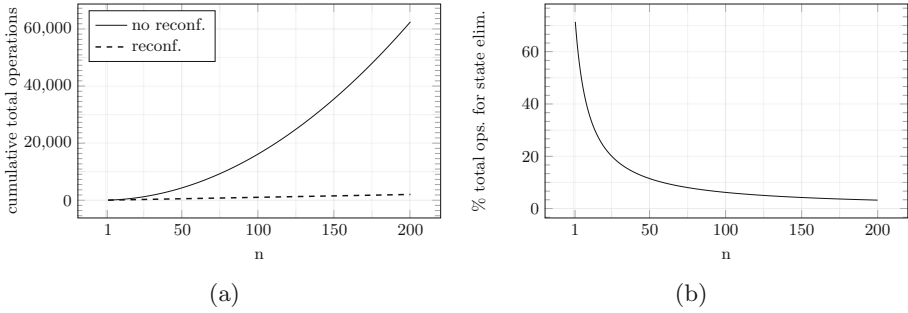
### 4.1 Zeroconf

We are interested in the reachability of the error state for the family of Zeroconf models, parameterised in the number  $n$  of attempts, after which the protocol will (potentially incorrectly) assume that it has selected a unique address. The initial model for  $n = 1$  is defined, its volatile region is determined as in Example 1, and Algorithm 2 is applied. In each incremental step we increment  $n$  and apply Algorithm 3 to the model. Volatile states can be identified in each step.

Figures 4a and b show the total number of performed arithmetic operations accumulated during the incremental analysis of the models and the ratio of the number of arithmetic operations performed for regular state elimination, respectively. This ratio shows the small share of the number of iterations required when the values are calculated for a range of parameters in our approach (repeated applications of Algorithm 3), when compared to the naïve approach to recalculate all values from scratch (applying Algorithm 2).

Figure 4a shows that the total number of operations is quadratic in the parameter when regular state elimination (applying Algorithm 2) is repeatedly applied from scratch. This is a consequence of the number of operations for *each* parameter being linear in the parameter value when naïvely applying Algorithm 2. This is in stark contrast to the number of operations needed when the parameter is stepwise incremented using Algorithm 3, stepwise capitalising on the analysis of the respective predecessor model. Here the update cost is constant: since the extent of structural change at each step is constant. This leads to dramatic savings (quadratic vs. linear) when exploring the parameter space, as illustrated by Fig. 4b.

<sup>1</sup> <https://github.com/PaulGainer/PMC>.



**Fig. 4.** Cumulative total of arithmetic operations performed for iterative analysis of Zeroconf for  $n = 1 \dots 200$  (a), and the ratio of total operations for reconfiguration to total operations for regular state elimination, given as a percentage (b).

## 4.2 Oscillator Synchronisation

We now consider the models developed in [8,9] to analyse protocols for the clock synchronisation of nodes in a network. In these protocols, consensus on clock values emerges from interactions between the nodes. The underlying mathematical model is that of coupled oscillators. This family of models is parametric in the number  $N$  of nodes that form the network; the granularity  $T$  of the discretisation of the oscillation cycle; the length  $R$  of the refractory period, during which nodes ignores interactions with their neighbours; the strength  $\epsilon$  of the coupling between the oscillators; and finally the likelihood  $\mu$  of any individual interaction between two nodes not occurring due to some external factor.

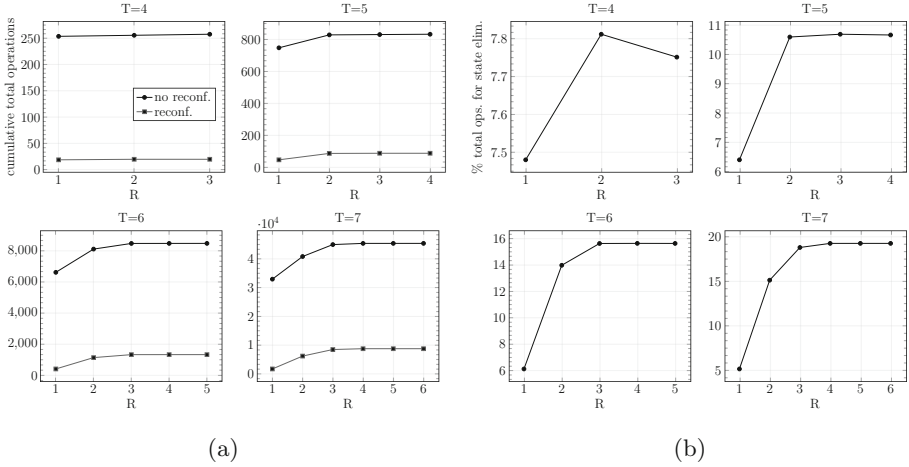
Each state of the model corresponds to some global configuration for the network—a vector encoding the size of node clusters that share the same progress through their oscillation cycle. The target states of interest are those in which all nodes share the same progression through their cycle and are therefore *synchronised*.

Changing the parameters  $N$  and  $T$  redefines the encoding of a global network state. This results in drastic changes to the structure of the model and therefore makes it hard to identify volatile states. Our prototypical implementation only considers low-level models defined explicitly as a set of states and a transition matrix, which trivialises the identification of volatile areas. Future implementation into ePMC, however, will allow volatile states to be clearly identified by analysing the guards present in high-level model description languages [1]. This works in particular for the parameters  $N$  and  $T$  we have studied.

Changing the parameter  $\epsilon$  results in such severe changes in the structure of the model that we do not see how the synergistic effects we have observed can be ported to analysing its parameter space, while changing  $\mu$  does not change the structure of the underlying graph and hence is not interesting for what we want to show.

In this paper, we therefore focus on the incremental analysis for the parameter  $R$ . We arbitrarily fix  $N$  to be 5 and  $\epsilon$  to be 0.1, and repeat the incremental





**Fig. 5.** Cumulative total of arithmetic operations performed for iterative analysis of synchronisation models with respect to  $R$  (a), and the ratio of totals for reconfiguration to totals for regular state elimination given as a percentage (b).

analysis for four different values for  $T$ . The parameter  $R$  varies from 1 to  $T$  (for each of the different values of  $T$  we have considered).

Figures 5a and b show the total number of performed arithmetic operations and the ratio of the totals for regular state elimination to the totals for reconfiguration given as a percentage, respectively. The effectiveness of the approach lessens as  $T$  increases, a result of the rounding of real values to discrete integer values that occurs when generating the transitions for the initial model [8]. Higher values of  $T$  result in an increase in the number of possible successor states for global states of the network, which in turn leads to an increase in the number of transitions in the model. Similarly, incrementing  $R$  results in reduced effectiveness as fewer interactions between nodes are ignored, and again more transitions are introduced to the model.

Overall it is clear that, while still substantial, the gains here are not as pronounced as those seen for the analysis of the Zeroconf protocol. This is to be expected, since the structural changes induced by changing the parameter  $R$  are not constant for each iteration—the higher the value of  $R$  the greater the extent of the structural changes incurred.

## 5 Conclusion and Future Work

It is clear—and, in hindsight, unsurprising—that our approach works well for structured Markov chains, such as chain-, ring-, or tree-like structures. Our experiments have lent evidence to this by showing that where the cost of model-checking an individual model grows linearly with a parameter, model checking up to a parameter becomes linear in the maximal parameter considered, whereas

the overall costs grow quadratically if all models are considered individually. Thus, we expect significant gain wherever changes can be localised and isolated. Moreover, we expect this to be the norm rather than the exception. After all, chains, rings, and trees are common structures in models.

It is quite striking that *very* specialised structures have enjoyed a lot of attention, and so have *absolutely* general ones. The standard example for very specialised structures is waiting queues. Fixed length waiting queues, for example, have closed form solutions. Thus, when the system analyst creates a structure, which is so standard that it has a known closed form solution *and*—and this is a big ‘and’—realises that this is the case and looks up the closed form solution, then this analysis is the unicorn. However, if the structure is slightly different, if she fails to see that the problem has a closed form solution, or if she does not want to invest the time to research the closed form solution, then she would currently have to fall back to the naïve solution. Here our technique is a nice sweet spot between these extremes: the speed is close to evaluating closed form solutions, but applying our method does not put any burden on the system analyst who creates the parametrised model.

The limitations of our model are that it loses much of its advantage when a change in a parameter induces severe structural changes in the model. For the synchronisation protocol, some parameters severely change the structure. This is because most of the nodes are connected by an edge, and for such dense graphs, structural changes can have a huge cone of influence. In the worst case, e.g. a fully connected graph, a cubic overhead is incurred [10].

The next step of our work will be to tap the full potential of our approach by integrating it into the probabilistic model checker ePMC [12]. Here the symbolic description of the system will expose the volatile areas and—more importantly—the non-volatile areas that appear to be stable under successive increments of the parameter values. We also expect to obtain synergies by combining our method with the approach of [7], extending our approach to models with non-determinism, such as interactive Markov chains and Markov decision processes.

**Acknowledgements.** This work was supported by the Sir Joseph Rotblat Alumni Scholarship at Liverpool, EPSRC grants EP/M027287/1 and EP/N007565/1, and by the Marie Skłodowska Curie Fellowship *Parametrised Verification and Control*.

## References

1. Alur, R., Henzinger, T.A.: Reactive modules. *Formal Methods Syst. Des.* **15**(1), 7–48 (1999)
2. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) *TACAS 2011*. LNCS, vol. 6605, pp. 326–340. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-19835-9\\_30](https://doi.org/10.1007/978-3-642-19835-9_30)
3. Bohnenkamp, H., van der Stok, P., Hermanns, H., Vaandrager, F.: Cost-optimization of the IPv4 zeroconf protocol, pp. 531–540. *IEEE Computer Society Press* (2003)

4. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: Liu, Z., Araki, K. (eds.) ICTAC 2004. LNCS, vol. 3407, pp. 280–294. Springer, Heidelberg (2005). [https://doi.org/10.1007/978-3-540-31862-0\\_21](https://doi.org/10.1007/978-3-540-31862-0_21)
5. Dehnert, C., et al.: PROPhESY: a probabilistic parameter synthesis tool. In: CAV, pp. 214–231 (2015)
6. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A STORM is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčák, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_31](https://doi.org/10.1007/978-3-319-63390-9_31)
7. Forejt, V., Kwiatkowska, M., Parker, D., Qu, H., Ujma, M.: Incremental runtime verification of probabilistic systems. In: Qadeer, S., Tasiran, S. (eds.) RV 2012. LNCS, vol. 7687, pp. 314–319. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-35632-2\\_30](https://doi.org/10.1007/978-3-642-35632-2_30)
8. Gainer, P., Linker, S., Dixon, C., Hustadt, U., Fisher, M.: Investigating parametric influence on discrete synchronisation protocols using quantitative model checking. In: Bertrand, N., Bortolussi, L. (eds.) QEST 2017. LNCS, vol. 10503, pp. 224–239. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66335-7\\_14](https://doi.org/10.1007/978-3-319-66335-7_14)
9. Gainer, P., Linker, S., Dixon, C., Hustadt, U., Fisher, M.: The power of synchronisation: formal analysis of power consumption in networks of pulse-coupled oscillators. arXiv preprint [arXiv:1709.04385](https://arxiv.org/abs/1709.04385) (2017)
10. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: a model checker for parametric markov models. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 660–664. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14295-6\\_56](https://doi.org/10.1007/978-3-642-14295-6_56)
11. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric markov models. STTT **13**(1), 3–19 (2011)
12. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: ISCASMC: a web-based probabilistic model checker. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 312–317. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-06410-9\\_22](https://doi.org/10.1007/978-3-319-06410-9_22)
13. Hopcroft, J.E.: Introduction to Automata Theory, Languages, and Computation. Pearson Education India (2008)
14. Jansen, N., et al.: Accelerating parametric probabilistic verification. In: QEST, pp. 404–420 (2014)
15. Johnson, B., Kress-Gazit, H.: Probabilistic analysis of correctness of high-level robot behavior with sensor error. In: Robotics: Science and Systems (2011)
16. Johnson, B., Kress-Gazit, H.: Probabilistic guarantees for high-level robot behavior in the presence of sensor error. Auton. Robots **33**(3), 309–321 (2012)
17. Johnson, B.L.: Synthesis, analysis, and revision of correct-by-construction controllers for robots with sensing and actuation errors. Ph.D. thesis, Cornell University (2015)
18. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov chains: with a chapter of Markov random fields by David Griffeath, vol. 40. Springer, New York (2012). <https://doi.org/10.1007/978-1-4684-9455-6>
19. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-72522-0\\_6](https://doi.org/10.1007/978-3-540-72522-0_6)
20. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)

21. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for Markov decision processes. In: International Conference on Dependable Systems & Networks, pp. 359–370. IEEE (2011)
22. Mirollo, R.E., Strogatz, S.H.: Synchronization of pulse-coupled biological oscillators. *SIAM J. Appl. Math.* **50**(6), 1645–1662 (1990)
23. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.: Parameter synthesis for Markov models: faster than ever. In: ATVA, pp. 50–67 (2016)



# Lumping the Approximate Master Equation for Multistate Processes on Complex Networks

Gerrit Großmann<sup>1</sup>(✉), Charalampos Kyriakopoulos<sup>1</sup>, Luca Bortolussi<sup>2</sup>,  
and Verena Wolf<sup>1</sup>

<sup>1</sup> Computer Science Department, Saarland University, Saarbrücken, Germany  
gerrit.grossmann@uni-saarland.de

<sup>2</sup> Department of Mathematics and Geosciences, University of Trieste, Trieste, Italy

**Abstract.** Complex networks play an important role in human society and in nature. Stochastic multistate processes provide a powerful framework to model a variety of emerging phenomena such as the dynamics of an epidemic or the spreading of information on complex networks. In recent years, mean-field type approximations gained widespread attention as a tool to analyze and understand complex network dynamics. They reduce the model's complexity by assuming that all nodes with a similar local structure behave identically. Among these methods the approximate master equation (AME) provides the most accurate description of complex networks' dynamics by considering the whole neighborhood of a node. The size of a typical network though renders the numerical solution of multistate AME infeasible. Here, we propose an efficient approach for the numerical solution of the AME that exploits similarities between the differential equations of structurally similar groups of nodes. We cluster a large number of similar equations together and solve only a single *lumped* equation per cluster. Our method allows the application of the AME to real-world networks, while preserving its accuracy in computing estimates of global network properties, such as the fraction of nodes in a state at a given time.

**Keywords:** Complex networks · Multistate processes · AME  
Model reduction · Lumping

## 1 Introduction

Various emerging phenomena of social, biological, technical, or economic nature can be modeled as stochastic multistate processes on complex networks [1, 3, 24, 26]. Such networks typically consist of millions or even billions of nodes [1, 3], each one being in one of a finite number of states. The state of a node can potentially change over time as a result of interaction with one of its neighboring nodes. The interactions among neighbors are specified by rules and occur independently at random time points, governed by the exponential distribution. Hence, the

underlying process is a discrete-state space Markovian process in continuous time (CTMC). Its state space consists of all labeled graphs representing all possible configurations of the complex network. For instance, in the susceptible-infective (SI) model, which describes the spread of a simple epidemic process, each node can either be susceptible or infected; infected nodes propagate the infection to their susceptible neighbors [5, 19].

Monte-Carlo simulations can be carried out only for small networks [11, 19], as they become very expensive for large networks, due to the large number of simulation runs which are necessary to draw reliable conclusions about the network's dynamics.

An alternative and viable approach is based on mean-field approximations, in which nodes sharing a similar local structure are assumed to behave identically and can be described by a single equation, capturing their mean behavior [3, 4, 10, 12, 18]. The heterogeneous (also called degree-based) mean-field (DBMF) approach proposes a system of ordinary differential equations (ODEs) with one equation approximating the nodes of degree  $k$  which are in a certain state [9, 18, 25]. The approximate master equation (AME) provides a far more accurate approximation of the network's dynamics, considering explicitly the complete neighborhood of a node in a certain state [14, 16, 17]. However, the corresponding number of differential equations that have to be solved is of the order  $\mathcal{O}(k_{\max}^{|\mathcal{S}|})$ , where  $k_{\max}$  is the network's largest degree and  $|\mathcal{S}|$  the number of possible states. A coarser approximation called pair approximation (PA) can be derived from AME by imposing the multinomial assumption for the number of neighbors in a state [16, 17]. Nevertheless, solving PA instead of AME is faster but for many networks not accurate enough [17].

Lumping is a popular model reduction technique for Markov-chains and systems of ODEs [6–8, 21, 28]. It has also been applied to the underlying model of epidemic contact processes [19, 27] and has recently been shown to be extremely effective for the DBMF equation as well as for the PA approach [20]. In this work, we generalize the approach of [20] providing a lumping scheme for the AME, leveraging the observation that nodes with a large degree having a similar neighborhood structure have also typically very similar behaviors. We show that it is possible to massively reduce the number of equations of the AME while preserving the accuracy of global statistical properties of the network. Our contributions, in particular, are the following: (i) we provide a fully automated aggregation scheme for the multistate AME; (ii) we introduce a heuristic to find a reasonable trade-off between number of equations and accuracy; (iii) we evaluate our method on different models from literature and compare our results with the original AME and Monte-Carlo simulation; (iv) we provide an open-source tool<sup>1</sup> written in Python, which takes as input a model specification, generates and solves the lumped (or original) AME.

The remainder of this paper is organized as follows: In Sect. 2 we describe multistate Markovian processes in networks and formally introduce the AME. In Sect. 3 we derive lumped equations for a given clustering scheme and in Sect. 4

<sup>1</sup> <https://github.com/gerritgr/LumPyQest>.

we propose and evaluate a clustering algorithm for grouping similar equations together. Case studies are presented in Sect. 5. We draw final conclusions and identify open research problems in Sect. 6.

## 2 The Multistate Approximate Master Equation

In this section, we first define contact processes and introduce our notation and terminology for the multistate AME.

### 2.1 Multistate Markovian Processes

We describe a contact process in a network  $(\mathcal{G}, \mathcal{S}, R, L)$  by a finite undirected graph  $\mathcal{G} = (V, E)$ , a finite set of states  $\mathcal{S}$ , a set of rules  $R$ , and an initial state for each agent (node) of the graph  $L : V \rightarrow \mathcal{S}$ . We use  $s, s', s''$  and  $s_1, s_2, \dots$  to denote elements of  $\mathcal{S}$ . At each time point  $t \geq 0$ , each node  $v \in V$  is in a state  $s \in \mathcal{S}$ . The rules  $R$  define how neighboring nodes influence the probability of state transitions. A rule consists of a *consumed* state, a *produced* state, and a transition rate, which depends on the neighborhood of the node. We use integer vectors to model a node's neighborhood. For a given set of states  $\mathcal{S}$  and maximal degree  $k_{\max}$ , the set of all potential neighborhood vectors is  $\mathcal{M} = \{\mathbf{m} \in \mathbb{Z}_{\geq 0}^{|\mathcal{S}|} \mid \sum_{s \in \mathcal{S}} \mathbf{m}[s] \leq k_{\max}\}$ , where we write  $\mathbf{m}[s]$  to refer to the number of neighbors in state  $s$ .

A rule  $r \in R$  is a triplet  $r = (s, f, s')$  with  $s, s' \in \mathcal{S}, s \neq s'$  and rate function  $f : \mathcal{M} \rightarrow \mathbb{R}_{\geq 0}$  corresponding to the exponential distribution. A rule  $r$  (also denoted as  $s \xrightarrow{f} s'$ ) can be applied at every node in state  $s$ , and, when applied, it transforms this node into state  $s'$ . Note that this general formulation of a rule containing the rate function can express all types of rules that are described in [14, 16, 17] such as spontaneous changes of a node's state (independent rules) or changes due to the state of a neighbor (contact rules). The delay until a certain rule is applied is exponentially distributed with rate  $f(\mathbf{m})$ , with rules competing in a race condition where the one with the shortest delay is executed. This results in an underlying stochastic model described by a CTMC.

In the following, we indicate with  $R^{s+} = \{(s', f, s) \in R, s' \in \mathcal{S}\}$  all the rules that change the state of a node into  $s$ , and with  $R^{s-} = \{(s, f, s') \in R, s' \in \mathcal{S}\}$  all rules that change an  $s$ -node into a different state.

**Example.** In the SIS model, a susceptible node can become infected by one of its neighbors. An infected node becomes susceptible again, independently of its neighbors. Hence, the infection rule is  $S \xrightarrow{\lambda_1 \cdot \mathbf{m}[I]} I$  and the recovery rule is  $I \xrightarrow{\lambda_2} S$ , where  $\mathbf{m}[I]$  denotes the number of infected neighbors and  $\lambda_1, \lambda_2 \in \mathbb{R}_{\geq 0}$  are rule-specific rate constants.

## 2.2 Multistate AME

Here, we briefly present the multistate AME, similarly to [14,20]. The AME assumes that all nodes in a certain state and with the same neighborhood structure are indistinguishable. We define  $\mathcal{M}_k = \{\mathbf{m} \in \mathcal{M} \mid \sum_{s \in \mathcal{S}} \mathbf{m}[s] = k\}$  to be the subset of neighborhood vectors referring to nodes of degree  $k$ . In addition, for  $s_1, s_2 \in \mathcal{S}$  and  $\mathbf{m} \in \mathcal{M}$ , we use  $\mathbf{m}^{\{s_1^+, s_2^-\}}$  to denote a neighborhood vector where all entries are equal to those of  $\mathbf{m}$ , apart from the  $s_1$ -th entry, which is equal to  $\mathbf{m}[s_1] + 1$ , and the  $s_2$ -th entry, which is equal to  $\mathbf{m}[s_2] - 1$ .

Let  $x_{s,\mathbf{m}}(t)$  be the fraction of network nodes that are in state  $s$  and have a neighborhood  $\mathbf{m}$  at time  $t$ , and assume the initial state  $x_{s,\mathbf{m}}(0)$  is known. Formally, the AME approximates the time evolution of  $x_{s,\mathbf{m}}$  with the following set of deterministic ODEs<sup>2</sup>:

$$\begin{aligned} \frac{\partial x_{s,\mathbf{m}}}{\partial t} = & \sum_{(s',f,s) \in R^{s^+}} f(\mathbf{m})x_{s',\mathbf{m}} - \sum_{(s,f,s') \in R^{s^-}} f(\mathbf{m})x_{s,\mathbf{m}} \\ & + \sum_{\substack{(s_1,s_2) \in \mathcal{S}^2 \\ s_1 \neq s_2}} \beta^{ss_1 \rightarrow ss_2} x_{s,\mathbf{m}^{\{s_1^+, s_2^-\}}} \mathbf{m}^{\{s_1^+, s_2^-\}}[s_1] \\ & - \sum_{\substack{(s_1,s_2) \in \mathcal{S}^2 \\ s_1 \neq s_2}} \beta^{ss_1 \rightarrow ss_2} x_{s,\mathbf{m}} \mathbf{m}[s_1], \end{aligned} \quad (1)$$

where, the term  $\beta^{ss_1 \rightarrow ss_2}$  is the average rate at which an  $(s, s_1)$ -edge changes into an  $(s, s_2)$ -edge, if  $s, s_1, s_2 \in \mathcal{S}$  with  $s_1 \neq s_2$ .

The first term in the right hand side models the inflow into  $(s, \mathbf{m})$  nodes from  $(s', \mathbf{m})$  nodes, while the second term models the outflow from  $(s, \mathbf{m})$  due to the application of a rule. The other two terms describe indirect effects on a  $(s, \mathbf{m})$  node due to changes in its neighboring nodes, again considering inflow and outflow (cf. Fig. 1). In particular, a node in the neighborhood  $\mathbf{m}$  of  $(s, \mathbf{m})$ , say in state  $s_1$ , changes to state  $s_2$  by the firing of a rule.

To compute  $\beta^{ss_1 \rightarrow ss_2}$  we need to define the subset of rules which consume a  $s_1$ -node and produce an  $s_2$ -node:  $R^{s_1 \rightarrow s_2} = \{(s_1, f, s_2) \in R \mid f : \mathcal{M} \rightarrow \mathbb{R}_{\geq 0}\}$ . Then

$$\beta^{ss_1 \rightarrow ss_2} = \frac{\sum_{\mathbf{m} \in \mathcal{M}} \sum_{(s_1, f, s_2) \in R^{s_1 \rightarrow s_2}} f(\mathbf{m})x_{s_1, \mathbf{m}} \mathbf{m}[s]}{\sum_{\mathbf{m} \in \mathcal{M}} x_{s_1, \mathbf{m}} \mathbf{m}[s]}, \quad (2)$$

where in the denominator we normalize dividing by the fraction of  $(s, s_1)$  edges. The total number of equations of AME is determined by the number of states  $|\mathcal{S}|$  and the maximal degree  $k_{\max}$ , and equals:

$$\binom{k_{\max} + |\mathcal{S}|}{|\mathcal{S}| - 1} (k_{\max} + 1). \quad (3)$$

<sup>2</sup> We omit  $t$  for the ease of notation.



The binomial arises from the number of ways in which, for a fixed degree  $k$ , one can distribute  $k$  neighbors into  $|\mathcal{S}|$  different states, see [20] for the proof.

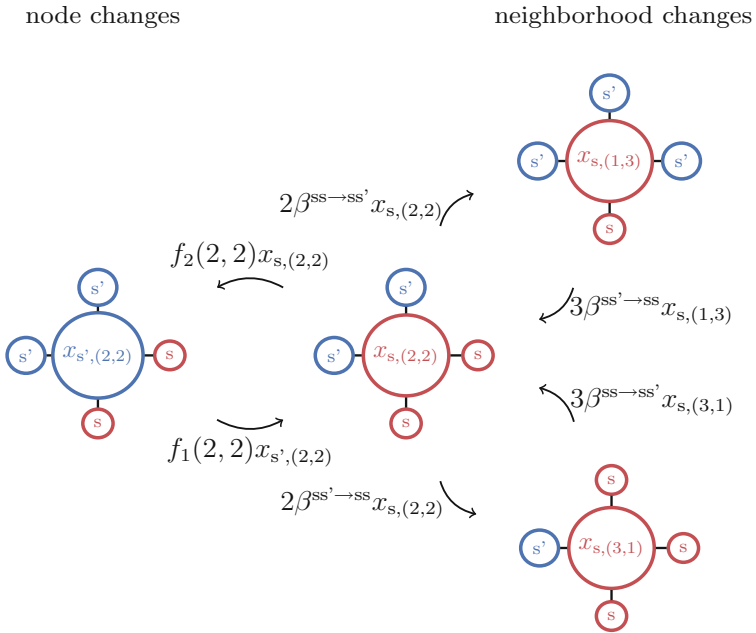
As  $x_{s,\mathbf{m}}$  are fractions of network nodes, the following identity holds for all  $t$ :

$$\sum_{s,\mathbf{m} \in \mathcal{S} \times \mathcal{M}} x_{s,\mathbf{m}}(t) = 1 \quad (4)$$

Moreover, we use  $x_s$  to denote the *global fraction* of nodes in a fixed state  $s$ , which we get by summing over all possible neighborhood vectors

$$x_s(t) = \sum_{\mathbf{m} \in \mathcal{M}} x_{s,\mathbf{m}}(t), \quad (5)$$

again with  $\sum_{s \in \mathcal{S}} x_s(t) = 1$ . Intuitively,  $x_s$  is the probability that a randomly chosen node from the network is in state  $s$ . This is the value of primary interest in many applications, e.g. [3, 24, 26]. Finally, the degree distribution  $P(k)$  gives the probability that a randomly chosen node is of degree  $k$  ( $0 \leq k \leq k_{\max}$ ). If we sum up all  $x_{s,\mathbf{m}}$  which belong to a specific  $k$  (i.e.  $\mathbf{m} \in \mathcal{M}_k$ ), as the network



**Fig. 1.** Illustration of how the AME governs the fraction of  $x_{s,(2,2)}$  in a two-state model with rules  $(s', f_1, s)$ ,  $(s, f_2, s')$ . The inflow and outflow between  $x_{s,(2,2)}$  and  $x_{s',(2,2)}$  is induced by the direct change of a node's state from  $s$  to  $s'$  or vice versa. The inflow and outflow between  $x_{s,(2,2)}$  and  $x_{s,(3,1)}$ ,  $x_{s,(1,3)}$  is attributed to the change of state of a node's neighbor.

structure is assumed to be static, we will necessarily obtain the corresponding degree probability. Hence, for each  $t \geq 0$ , we have

$$\sum_{s, \mathbf{m} \in \mathcal{S} \times \mathcal{M}_k} x_{s, \mathbf{m}}(t) = P(k). \quad (6)$$

### 3 Lumping

The key idea of this paper is to group together equations of the AME which have a similar structure and to solve only a single *lumped* equation per group. This lumped equation will capture the evolution of the sum of the AME variables in each group.

Therefore, we divide the set  $\{x_{s, \mathbf{m}} \mid s \in \mathcal{S}, \mathbf{m} \in \mathcal{M}\}$  into groups or *clusters*, constructing our clustering such that two equations  $x_{s, \mathbf{m}}, x_{s', \mathbf{m}'}$  can only end up in the same group if  $s = s'$  and  $\mathbf{m}$  is ‘sufficiently’ similar to  $\mathbf{m}'$ . This ensures that the fractions within a cluster as well as their time derivatives are similar, provided the change in the rate as a function of  $\mathbf{m}$  is relatively small when  $\mathbf{m}$  is large.

Next we consider a clustering  $\mathcal{C}$  defined as a partition over  $\mathcal{M}$ , i.e.,  $\mathcal{C} \subset 2^{\mathcal{M}}$  and  $\bigcup_{C \in \mathcal{C}} C = \mathcal{M}$  and all clusters  $C$  are disjoint and non-empty. Before we discuss in detail the construction of  $\mathcal{C}$  in Sect. 4, we derive the lumped equations for a given clustering  $\mathcal{C}$ .

First, recall that we want to approximate the global fractions for each state (cf. Eq. (5)), which can be split into sums over the clusters

$$x_s(t) = \sum_{C \in \mathcal{C}} \sum_{\mathbf{m} \in C} x_{s, \mathbf{m}}(t). \quad (7)$$

Our goal is now to construct a smaller equation system, where the variables  $z_{s, C}$  approximate the sum over all  $x_{s, \mathbf{m}}$  with  $\mathbf{m} \in C$

$$z_{s, C}(t) \approx \sum_{\mathbf{m} \in C} x_{s, \mathbf{m}}(t). \quad (8)$$

Henceforth, we can approximate the global fractions as

$$x_s(t) \approx \sum_{C \in \mathcal{C}} z_{s, C}(t). \quad (9)$$

The number of equations is then given by  $|\mathcal{S}| \cdot |\mathcal{C}|$ . As one might expect, there is a trade-off between the accuracy of  $z_{s, C}(t)$  and the computational cost, proportional to the number of clusters.

#### 3.1 Lumping the Initial State and the Time Derivative

As the initial values of  $x_{s, \mathbf{m}}$  are given, we define the initial lumped values

$$z_{s, C}(0) = \sum_{\mathbf{m} \in C} x_{s, \mathbf{m}}(0). \quad (10)$$

To achieve the criterion in Eq. (8) for the fractions computed at  $t > 0$ , we seek for time derivatives which fulfill

$$\frac{\partial z_{s,C}}{\partial t} \approx \sum_{\mathbf{m} \in C} \frac{\partial x_{s,\mathbf{m}}}{\partial t}. \quad (11)$$

Note that an exact lumping is in general not possible as  $\frac{\partial z_{s,C}}{\partial t}$  is a function of the individual  $x_{s,\mathbf{m}}(t)$ . In order to close the equations for  $z_{s,C}$ , we need to express  $x_{s,\mathbf{m}}$  as an approximate function of  $z_{s,C}$ . The naive idea is to assume that the true fractions  $x_{s,\mathbf{m}}$  are similar for all  $\mathbf{m}$  that belong to the same cluster, i.e., if  $\mathbf{m}, \mathbf{m}' \in C$  then  $x_{s,\mathbf{m}} \approx x_{s,\mathbf{m}'}$ , leading to an approximation of  $x_{s,\mathbf{m}}$  as  $z_{s,C}/|C|$ .

This is however problematic, as it neglects the fact that neighbors of nodes of different degree have different size. In fact, even if for two degrees  $k_1 < k_2$  in the same cluster we have  $P(k_1) = P(k_2)$  (while typically  $P(k_1) > P(k_2)$ ), the number of possible different neighbors  $\mathbf{m}_2$  of a  $k_2$ -node is larger than the number of different neighbors  $\mathbf{m}_1$  of a  $k_1$ -node,  $|\mathcal{M}_{k_1}| < |\mathcal{M}_{k_2}|$ , hence typically  $x_{s,\mathbf{m}_2} < x_{s,\mathbf{m}_1}$ , as the mass of  $P(k_2)$  has to be split among more variables. In order to correct for this asymmetry between degrees in each cluster, we introduce the following assumption:

**Assumption:** All fractions  $x_{s,\mathbf{m}}$  inside a cluster  $C$  that refer to the same degree contribute equally to the sum  $z_{s,C}$ . Equations of different degree contribute proportionally to their degree probability  $P(k)$  and inversely proportionally to the neighborhood size for that degree.

Based on the above assumption, we define a degree dependent scaling-factor  $w_{C,k} \in \mathbb{R}_{\geq 0}$ , which only depends on the corresponding cluster  $C$  and degree  $k$ . According to the above assumption  $w_{C,k} \propto \frac{P(k)}{|\mathcal{M}_k|}$ . To ensure that the weights of one cluster sum up to one, we define

$$w_{C,k} = \frac{P(k)}{|\mathcal{M}_k|} \cdot \left( \sum_{\mathbf{m} \in C} \frac{P(k_{\mathbf{m}})}{|\mathcal{M}_{k_{\mathbf{m}}}|} \right)^{-1}, \quad (12)$$

where  $k_{\mathbf{m}} = \sum_{s \in \mathcal{S}} \mathbf{m}[s]$  is the degree of a neighborhood  $\mathbf{m}$ . We compute approximations of  $x_{s,\mathbf{m}}$  based on  $z_{s,C}$  as

$$x_{s,\mathbf{m}} \approx z_{s,C} \cdot w_{C,k_{\mathbf{m}}}. \quad (13)$$

### 3.2 Building the Lumped Equations

To define a differential equation for the lumped fraction  $z_{s,C}$ , we consider again Eq. (11) and replace  $\frac{\partial x_{s,\mathbf{m}}}{\partial t}$  by the r.h.s. of Eq. (1). Then we substitute every occurrence of  $x_{s,\mathbf{m}}$  by its corresponding lumped variable multiplied with the scaling factor, i.e.,  $z_{s,C} \cdot w_{C,k_{\mathbf{m}}}$ , where  $\mathbf{m} \in C$ . Since  $\mathbf{m} \in C$  does generally not imply that  $\mathbf{m}^{\{s_1^+, s_2^-\}} \in C$ , the substitution of  $x_{s,\mathbf{m}^{\{s_1^+, s_2^-\}}}$  is somewhat more complicated. Let  $C(\mathbf{m})$  denote the cluster  $\mathbf{m}$  belongs to. If  $\mathbf{m}$  lies ‘‘at the border’’

of a cluster then  $C(\mathbf{m}^{\{s_1^+, s_2^-\}})$  might be different than  $C(\mathbf{m})$ . The lumped AME takes then the following form<sup>3</sup>:

$$\begin{aligned} \frac{\partial z_{s,C}}{\partial t} &= \sum_{(s',f,s) \in R^{s^+}} z_{s',C} \left( \sum_{\mathbf{m} \in C} w_{C,k_{\mathbf{m}}} f(\mathbf{m}) \right) \\ &- \sum_{(s,f,s') \in R^{s^-}} z_{s,C} \left( \sum_{\mathbf{m} \in C} w_{C,k_{\mathbf{m}}} f(\mathbf{m}) \right) \\ &+ \sum_{\substack{(s_1,s_2) \in \mathcal{S}^2 \\ s_1 \neq s_2}} \beta_{\mathcal{L}}^{ss_1 \rightarrow ss_2} \left( \sum_{\mathbf{m} \in C} w_{C(\mathbf{m}^{\{s_1^+, s_2^-\}}, k_{\mathbf{m}})} z_{s,C(\mathbf{m}^{\{s_1^+, s_2^-\}})} \mathbf{m}^{\{s_1^+, s_2^-\}}[s_1] \right) \\ &- \sum_{\substack{(s_1,s_2) \in \mathcal{S}^2 \\ s_1 \neq s_2}} \beta_{\mathcal{L}}^{ss_1 \rightarrow ss_2} z_{s,C} \left( \sum_{\mathbf{m} \in C} w_{C,k_{\mathbf{m}}} \mathbf{m}[s_1] \right), \end{aligned} \tag{14}$$

where

$$\beta_{\mathcal{L}}^{ss_1 \rightarrow ss_2} = \frac{\sum_{C \in \mathcal{C}} z_{s_1,C} \sum_{(s_1,f,s_2) \in R^{s_1 \rightarrow s_2}} \sum_{\mathbf{m} \in C} f(\mathbf{m}) w_{C,k_{\mathbf{m}}} \mathbf{m}[s]}{\sum_{C \in \mathcal{C}} z_{s_1,C} \sum_{\mathbf{m} \in C} w_{C,k_{\mathbf{m}}} \mathbf{m}[s]}. \tag{15}$$

To gain a significant speedup compared to the original equation system, it is necessary that the lumped equations can be efficiently evaluated. In particular, we want the number of terms in the lumped equation system to be proportional to the number of fractions  $z_{s,C}$  and not to the number of  $x_{s,\mathbf{m}}$ . This is possible for Eq. (14), because each time we have a sum over  $\mathbf{m} \in C$ , for instance  $\sum_{\mathbf{m} \in C} w_{C,k_{\mathbf{m}}} f(\mathbf{m})$ , we can precompute this value during the generation of the equations and do not have to evaluate it at every step of the ODE solver. The sum

$$\sum_{\mathbf{m} \in C} z_{s,C(\mathbf{m}^{\{s_1^+, s_2^-\}})} \mathbf{m}^{\{s_1^+, s_2^-\}}[s_1]$$

can be evaluated efficiently since we only have to consider lumped variables that correspond to clusters  $C(\mathbf{m}^{\{s_1^+, s_2^-\}})$  that are close to  $C(\mathbf{m})$ , i.e., that can be reached from a state in  $C(\mathbf{m})$  by the application of a rule. The number of such neighboring clusters is typically small, due to our definition of clusters, see Sect. 4.

*Remark 1.* For large  $k_{\max}$ , the number of neighbor vectors in  $\mathcal{M}$ , i.e. the size of the AME, becomes prohibitively large. For instance, for a maximum degree of the order of 10 thousands, quite common in real networks, the size of  $\mathcal{M}$  becomes of the order of  $10^{12}$ . Even summing a number of elements of this order while generating equations becomes very costly. To overcome this limit, the solution is to approximate terms involving summations in Eq. (14). Consider for instance  $\sum_{\mathbf{m} \in C} w_{C,k_{\mathbf{m}}} f(\mathbf{m})$ . Instead of evaluating  $f$  at every  $\mathbf{m} \in C$  and averaging it

<sup>3</sup> We omit once more  $t$  for easiness.

w.r.t.  $w_{C,k_m}$ , we can only evaluate  $f$  at the mean neighborhood vector  $\langle \mathbf{m} \rangle_C$ , where each coordinate is defined as  $\langle \mathbf{m} \rangle_C[s] = \sum_{\mathbf{m} \in C} w_{C,k_m} \mathbf{m}[s]$ . We can then approximate  $\sum_{\mathbf{m} \in C} w_{C,k_m} f(\mathbf{m}) \approx f(\langle \mathbf{m} \rangle_C)$ . Note that can compute  $\langle \mathbf{m} \rangle_C$  without iterating over all  $\mathbf{m} \in C$ . Likewise, we can efficiently approximate the sum in the third term, concerning the inflow induced by changes in the neighborhood, by exploiting the nice, triangle-based, geometrical structure of the clustering.

## 4 Partitioning of the Neighborhood Set

In this section we describe an algorithm to partition  $\mathcal{M}$ , and construct the clustering  $\mathcal{C}$ . Our algorithm builds partitions with a varying granularity to control the trade-off between accuracy and execution speed. We consider three main criteria: the similarity of different equations, their impact on the global error, and how fast is the evaluation of the lumped equations. Furthermore, as the size of  $\mathcal{M}$  can be extremely large, we cannot rely on typical hierarchical clustering algorithms having a cubic runtime in the number of elements to be clustered. Our solution is to decouple each  $\mathbf{m}$  into two components: its degree  $k_m$  (encoding its length) and its projection to the unit simplex (encoding its direction). We cluster these two components independently.

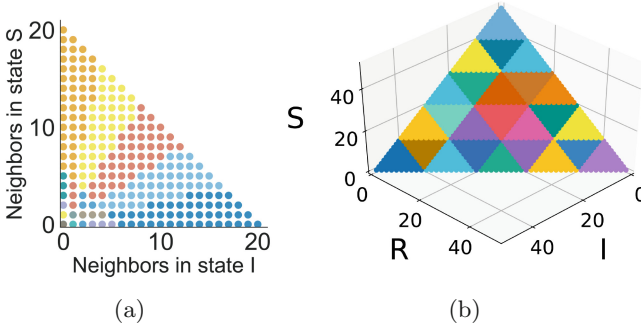
### 4.1 Hierarchical Clustering for Degrees

Since our clustering is degree-dependent, we first partition the set of degrees  $\{0, \dots, k_{\max}\}$ . Let  $\mathcal{K} \subset 2^{\{0, \dots, k_{\max}\}}$  be a degree partitioning, i.e., the disjoint union of all  $K \in \mathcal{K}$  is the set of degrees. The goal of the degree clustering is to merge together consecutive degrees with small probability while putting degrees with high probability mainly in separate clusters. This is particularly relevant for the power-law distribution, which is predominantly found in real world networks [1, 2] as it allows us to cluster a large number of high degrees with low total probability all together without losing much information.

We use an iterative procedure inspired by bottom-up hierarchical clustering to determine  $\mathcal{K}$ . We start by assigning to each degree an individual cluster and iteratively join the two consecutive clusters that increase the cost function  $\mathcal{L}$  by the least amount. The cost function  $\mathcal{L}$  punishes disparity in the spread of probability mass over clusters, leading to clusters that have approximately the same total probability mass. It is defined as

$$\mathcal{L}(\mathcal{K}) = \sum_{K \in \mathcal{K}} \left( \sum_{k \in K} P(k) \right)^2. \quad (16)$$

Note that  $\mathcal{L}(\mathcal{K})$  is minimal when all  $\sum_{k \in K} P(k)$  have equal values. The algorithm needs  $\mathcal{O}(k_{\max}^2)$  comparisons to determine the degree cluster of each element. At the end of this procedure, each  $\mathbf{m} \in \mathcal{M}$  has a corresponding degree-cluster  $K$  with  $k_m \in K$ .



**Fig. 2.** Left: Clustering of  $\mathcal{M}$  for a 2-state model with  $k_{\max} = 20$  and  $|\mathcal{K}| = |\mathcal{P}| = 7$ . Right: Proportionality cluster of a 3-state ( $\mathcal{S} = \{S, I, R\}$ ) model with  $k_{\max} = 50$  and  $|\mathcal{P}| = 5$ . Only the plane  $\mathcal{M}_{50}$  is shown.

### 4.2 Proportionality Clustering

Independently of  $\mathcal{K}$ , we partition  $\mathcal{M}$  along the different components of vectors  $\mathbf{m} \in \mathcal{M}$ . First, observe that if we normalize  $\mathbf{m}$  by dividing each dimension by  $k_{\mathbf{m}}$ , we can embed each  $\mathcal{M}_k$  into the unit simplex in  $\mathbb{R}^{|\mathcal{S}|}$ . The idea is then to partition the unit simplex, and apply the same partition to all  $\mathcal{M}_k$ . More specifically, we construct such partition coordinate-wise. As each element of the normalized  $\mathbf{m}$  takes values in  $[0, 1]$ , we split the unit interval in  $p + 1$  subintervals  $\mathcal{P} = \{[0, \frac{1}{p}), [\frac{1}{p}, \frac{2}{p}), \dots, [\frac{p-1}{p}, 1]\}$ . Then, two normalized neighbor vectors are in the same proportionality cluster if and only if their coordinates all belong to the subinterval  $P \in \mathcal{P}$ , possibly different for each coordinate.

### 4.3 Joint Clusters

Finally, we construct  $\mathcal{C}$  such that two points  $\mathbf{m}, \mathbf{m}'$  are in the same cluster if and only if they are in the same degree-cluster (i.e.,  $\exists K \in \mathcal{K} : k_{\mathbf{m}}, k_{\mathbf{m}'} \in K$ ) and in the same proportionality cluster, (i.e., for each dimension  $s \in \mathcal{S}$ , there exists a  $P \in \mathcal{P}$ , such that  $\frac{\mathbf{m}[s]}{k_{\mathbf{m}}}, \frac{\mathbf{m}'[s]}{k_{\mathbf{m}'}} \in P$ ).

The effect of combining degree and proportionality clusters, for a model with two different states, is shown in Fig. 2a, where the proportionality clustering gives equally sized triangles that are cut at different degrees by the degree clustering. If we fix a degree  $k$ , each cluster has only two neighbors (one in each direction). In the 3d-case, the proportionality clustering creates tetrahedra, which correspond to triangles if we fix a degree  $k$  (cf. Fig. 2b).

The above clustering admits some advantageous properties: (1) If we fix a degree, all clusters have approximately the same size and spatial shape; (2) The number of ‘direct spatial neighboring clusters’ of each cluster is always small, which simplifies the identification of clusters in the ‘border’ cases and eases the generation and evaluation of the lumped AME. Hence, the clusters can be efficiently computed even if  $\mathcal{M}$  is very large. Next, we discuss how to choose the size of the clusterings  $\mathcal{K}$  and  $\mathcal{P}$ .

#### 4.4 Stopping Heuristic

To find an adequate number of clusters, we solve the lumped AME of the model multiple times while increasing the number of clusters. We stop when the difference between different lumped solutions converges. The underlying assumption is that the approximations become more accurate with an increasing number of clusters and that the respective difference between consecutive lumped solutions becomes evidently smaller when the error starts to level off. Our goal is to stop when the increase in the number of clusters does not bring an appreciable increase in accuracy.

Let  $\mathbf{z}'(t)$ ,  $\mathbf{z}''(t)$  be two solution vectors, i.e., containing the fractions of nodes in each state at time  $t$ , of the lumped AME that correspond to two different clusterings  $\mathcal{C}'$  and  $\mathcal{C}''$ . We define the difference between two such solutions  $\mathbf{z}'$ ,  $\mathbf{z}''$  as their maximal Euclidean distance over time.

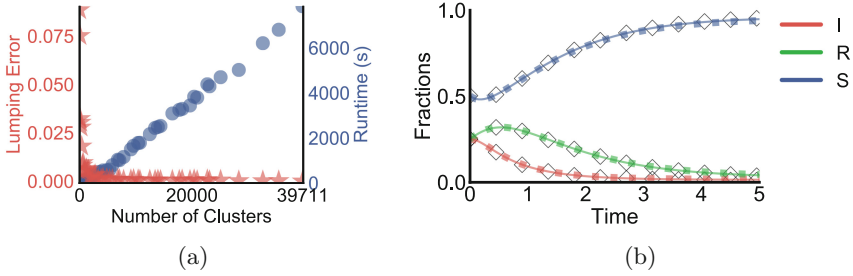
$$\epsilon(\mathbf{z}', \mathbf{z}'') = \max_{0 \leq t \leq H} \sqrt{\sum_{s \in \mathcal{S}} \left( z'_s(t) - z''_s(t) \right)^2}. \quad (17)$$

For the initial clustering we choose  $|\mathcal{K}| = |\mathcal{P}| = c_0$ . In each step, we increase the number of clusters by multiplying the previous  $c_i$  with a fixed constant, thus  $c_{i+1} = \lceil r c_i \rceil$  ( $r > 1$ ). We find this to be a more robust approach than increasing  $c_i$  by only a fixed amount in each step. We stop when the difference between two consecutive solutions are smaller than  $\epsilon_{\text{stop}} > 0$ . We consistently observe in all our case studies that  $\epsilon(\mathbf{z}', \mathbf{z}'')$  is a very good indicator on the behavior of the *real* error (cf. Fig. 6b). For our experiments we set empirically  $c_0 = 10$ ,  $r = 1.3$ , and  $\epsilon_{\text{stop}} = 0.01$ .

## 5 Case Studies

We demonstrate our approach on three different processes, namely the well-known SIR model, a rumor spreading model, and a SIS model with competing pathogens [13, 22]. We test how the number of clusters, the accuracy, and the runtime of our lumping method relate. In addition, we compare the dynamics of the original and lumped AME with the outcome of Monte-Carlo simulations on a synthetic network of  $10^5$  nodes [15, 23]. We performed our experiments on an Ubuntu machine with 8 GB of RAM and quad-core AMD Athlon II X4 620 processor. The code is written in Python 3.5 using SciPy's *vode*<sup>4</sup> ODE solver. The lumping error we provide is the difference between lumped solutions (corresponding to different granularities) and the outcome of the original AME. That is, for the original solution  $\mathbf{x}$  and a lumped solution  $\mathbf{z}$ , we define the lumping errors of  $\mathbf{z}$  as  $\epsilon(\mathbf{x}, \mathbf{z})$ . To generate the error curves, we start with  $|\mathcal{P}| = |\mathcal{K}| = 5$  and increase both quantities by one in each step. Note that we test our approach on models with comparably small  $k_{\text{max}}$ . In general, this undermines the effectiveness of our lumping approach; however using a larger  $k_{\text{max}}$  would have hindered the generation of the complete error curve.

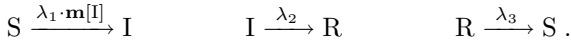
<sup>4</sup> <https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.integrate.ode.html>.



**Fig. 3.** SIR model. (a): We investigate how the lumping error (star) and the runtime of the ODE solver (circle) change with increasing number of clusters. For each cluster three ODEs are solved, one for each state. (b): Fractions of S,I,R nodes over time, as predicted by the original AME (solid line), by the lumped AME (dashed line), and based on Monte-Carlo simulations (diamonds).

### 5.1 SIR

First, we examine the well-known SIR model, where infected nodes (I) go through a recovery state (R) before they become susceptible (S) again:

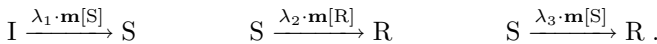


We choose  $(\lambda_1, \lambda_2, \lambda_3) = (3.0, 2.0, 1.0)$  and assume a network structure with  $k_{\max} = 60$  and a truncated power-law degree distribution with  $\gamma = 2.5$ . The initial distribution is  $(x_I(0), x_R(0), x_S(0)) = (0.25, 0.25, 0.5)$ .

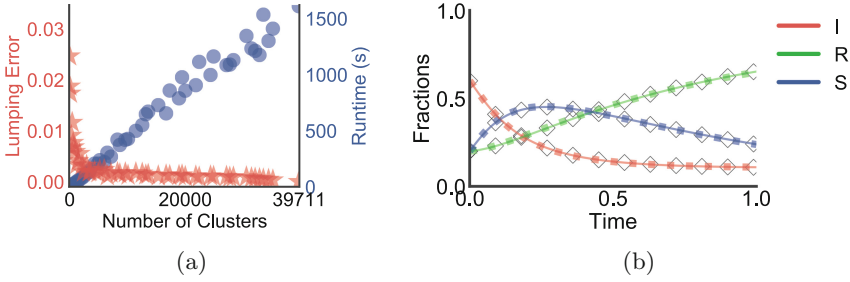
In this model the lumping is extremely accurate. In particular, we see that the lumping error of our method becomes quickly very small (Fig. 3a) and that we only need a few hundred ODEs to get a reasonable approximation of the original AME. The lumped solution  $\mathbf{z}$  we get from the stopping heuristic, consisting of less than 5% of the original equations, is almost indistinguishable from the original AME solution  $\mathbf{x}$  and the Monte-Carlo simulation (Fig. 3b). The lumping error is  $\epsilon(\mathbf{x}, \mathbf{z}) = 0.0015$ . The lumped solution used here 1791 clusters with a runtime of 235 s while solving the original AME we needed 39711 clusters and 7848 s.

### 5.2 Rumor Spreading

In the rumor spreading model [13], agents are either ignorants (I) who do not know about the rumor, spreaders (S) who spread the rumor, or stiflers (R) who know about the rumor, but are not interested in spreading it. Ignorants learn about the rumor from spreaders and spreaders lose interest in the rumor when they meet stiflers or other spreaders. Thus, the rules of the model are the following:

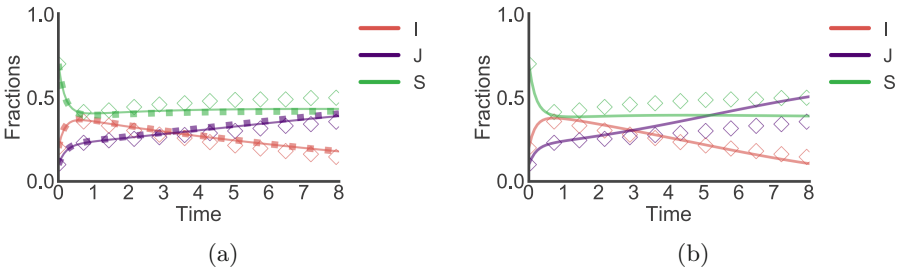






**Fig. 4.** Rumor spreading model. (a): Lumping error and runtime of the ODE solver w.r.t. the number of clusters. (b): Fractions of nodes in each state over time given by the original AME (solid line), by the lumped AME (dashed line), and based on Monte–Carlo simulations (diamonds).

We assume  $(\lambda_1, \lambda_2, \lambda_3) = (6.0, 0.5, 0.5)$  with  $k_{\max} = 60$  and  $\gamma = 3.0$ . The initial distribution is set to  $(x_I(0), x_R(0), x_S(0)) = (\frac{3}{5}, \frac{1}{5}, \frac{1}{5})$ . Again, we find that Monte–Carlo simulations, original AME, and lumped AME are in excellent agreement (Fig. 4b). The error curve, however, converges slower to zero than in the SIR model but it gets fast enough close to it (Fig. 4a). The lumped solution corresponds to 1032 clusters with a lumping error of 0.0059 and a runtime of 35 s compared to 39711 clusters of the original AME solution the runtime of which was 1606 s.



**Fig. 5.** Competing pathogens dynamics. (a): Fractions of nodes in I, J, S: original AME (solid line); lumped AME (dashed line); Monte–Carlo simulations (diamonds). (b): Comparison of pair approximation with Monte–Carlo simulation (diamonds).

### 5.3 Competing Pathogens

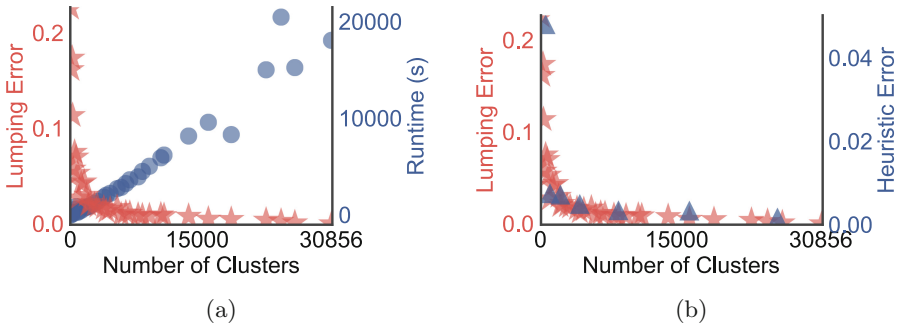
We, finally, examine an epidemic model with two competing pathogens [22]. The pathogens are denoted by I and J and the susceptible state by S:



We assume that both pathogens have the same infection rate and differ only in their respective recovery rates. Specifically, we set  $(\lambda_1, \lambda_2, \lambda_3, \lambda_4) = (5.0, 5.0, 1.5, 1.0)$  and assume network parameters of  $k_{\max} = 55$  and  $\gamma = 2.5$ . The initial distribution is  $(x_I(0), x_J(0), x_S(0)) = (0.2, 0.1, 0.7)$ .

This model is the most challenging case study for our approach. AME solution and naturally lumped AME are not in perfect alignment with Monte Carlo simulations (Fig. 5a) and, compared to the previous cases, our lumping approach needs a larger number of clusters to get a reasonably good approximation of the AME (Fig. 6a). The computational gain is, however, large as well. The lumped solution that comes with an approximation error of 0.02 corresponds to 2135 clusters and a runtime of 961 s compared to 30856 clusters and 17974 s of the original AME solution. Pair approximation approach (Fig. 5b), although being faster than the lumped AME (40 s runtime), would have here resulted to a much larger approximation error than our method (cf. Fig. 5b).

At last, the slow convergence of the error curve makes the competing pathogen model a good test case for our stopping heuristic. The heuristic evaluates the model for three different clusterings (509, 986, 2135 clusters). It stops as the difference between the two last clusterings is smaller than  $\epsilon_{\text{stop}}$ , showing its effectiveness also for challenging models. In Fig. 6b we show the alignment between the true lumping error and the surrogate error used by the heuristic.



**Fig. 6.** Competing pathogens lumping. (a): Lumping error and runtime of the ODE solver w.r.t. the number of clusters. (b): Lumping error compared to the error used by the heuristic.

## 6 Conclusions and Future Work

In this paper, we present a novel model-reduction technique to overcome the large computational burden of the multistate AME and make it tractable for real world problems. We show that it is possible to describe complex global behavior of dynamical processes using only an extremely small fraction of the original equations. Our approach exploits the high similarity among the original equations as well as the comparably small impact of equations belonging to the

tail of the power-law degree distribution. In addition, we propose an approach for finding a reasonable trade-off between accuracy and runtime of our method. Our approach is particularly useful in situations where several evaluations of the AME are necessary such as for the estimation of parameters or for model selection.

For future work, we plan to develop a method for on-the-fly clustering, which joins equations and breaks them apart during integration. This would allow the clustering to take into account the concrete (local) dynamics and to analyze adaptive networks with a variable degree distribution.

**Acknowledgments.** This research was been partially funded by the German Research Council (DFG) as part of the Collaborative Research Center “Methods and Tools for Understanding and Controlling Privacy”. We thank James P. Gleeson for his comments regarding the performance of AME on specific models and Michael Backenköhler for his comments on the manuscript.

## References

1. Barabási, A.-L.: Network Science. Cambridge University Press, Cambridge (2016)
2. Barabási, A.-L., Albert, R.: Emergence of scaling in random networks. *Science* **286**(5439), 509–512 (1999)
3. Barrat, A., Barthélemy, M., Vespignani, A.: Dynamical Processes on Complex Networks. Cambridge University Press, Cambridge (2008)
4. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective system behaviour: a tutorial. *Perform. Eval.* **70**(5), 317–349 (2013)
5. Brauer, F.: Mathematical epidemiology: past, present, and future. *Infect. Dis. Model.* **2**(2), 113–127 (2017)
6. Buchholz, P.: Exact and ordinary lumpability in finite markov chains. *J. Appl. Probab.* **31**(1), 59–75 (1994)
7. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: ERODE: a tool for the evaluation and reduction of ordinary differential equations. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, pp. 310–328. Springer, Heidelberg (2017). [https://doi.org/10.1007/978-3-662-54580-5\\_19](https://doi.org/10.1007/978-3-662-54580-5_19)
8. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Syntactic markovian bisimulation for chemical reaction networks. In: Aceto, L., Bacci, G., Bacci, G., Ingólfssdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools. LNCS, vol. 10460, pp. 466–483. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63121-9\\_23](https://doi.org/10.1007/978-3-319-63121-9_23)
9. Castellano, C., Pastor-Satorras, R.: Thresholds for epidemic spreading in networks. *Phys. Rev. Lett.* **105**(21), 218701 (2010)
10. Cator, E., Van Mieghem, P.: Second-order mean-field susceptible-infected-susceptible epidemic threshold. *Phys. Rev. E* **85**(5), 056111 (2012)
11. Cota, W., Ferreira, S.C.: Optimized gillespie algorithms for the simulation of markovian epidemic processes on large and heterogeneous networks. *Comput. Phys. Commun.* **219**, 303–312 (2017)
12. Demirel, G., Vazquez, F., Böhme, G.A., Gross, T.: Moment-closure approximations for discrete adaptive networks. *Physica D* **267**, 68–80 (2014)

13. Fedewa, N., Krause, E., Sisson, A.: Spread of a rumor. In: Society for Industrial and Applied Mathematics. Central Michigan University, vol. 25 (2013)
14. Fennell, P.G.: Stochastic processes on complex networks: techniques and explorations. Ph.D. thesis, University of Limerick (2015)
15. Fosdick, B.K., Larremore, D.B., Nishimura, J., Ugander, J.: Configuring random graph models with fixed degree sequences. arXiv preprint [arXiv:1608.00607](https://arxiv.org/abs/1608.00607) (2016)
16. Gleeson, J.P.: High-accuracy approximation of binary-state dynamics on networks. *Phys. Rev. Lett.* **107**(6), 068701 (2011)
17. Gleeson, J.P.: Binary-state dynamics on complex networks: pair approximation and beyond. *Phys. Rev. X*, **3**(2), 021004 (2013)
18. Gleeson, J.P., Melnik, S., Ward, J.A., Porter, M.A., Mucha, P.J.: Accuracy of mean-field theory for dynamics on real-world networks. *Phys. Rev. E* **85**(2), 26106 (2012)
19. Kiss, I.Z., Miller, J.C., Simon, P.L.: Mathematics of epidemics on networks: from exact to approximate models. In: Forthcoming in Springer TAM series. Springer, Heidelberg (2016). <https://doi.org/10.1007/978-3-319-50806-1>
20. Kyriakopoulos, C., Grossmann, G., Wolf, V., Bortolussi, L.: Lumping of degree-based mean-field and pair-approximation equations for multistate contact processes. *Phys. Rev. E* **97**(1), 012301 (2018)
21. Li, G., Rabitz, H.: A general analysis of approximate lumping in chemical kinetics. *Chem. Eng. Sci.* **45**(4), 977–1002 (1990)
22. Masuda, N., Konno, N.: Multi-state epidemic processes on complex networks. *J. Theor. Biol.* **243**(1), 64–75 (2006)
23. Newman, M.E.J.: The structure and function of complex networks. *SIAM Rev.* **45**(2), 167–256 (2003)
24. Pastor-Satorras, R., Castellano, C., Van Mieghem, P., Vespignani, A.: Epidemic processes in complex networks. *Rev. Mod. Phys.* **87**(3), 925 (2015)
25. Pastor-Satorras, R., Vespignani, A.: Epidemic spreading in scale-free networks. *Phys. Rev. Lett.* **86**(14), 3200 (2001)
26. Porter, M., Gleeson, J.: *Dynamical Systems on Networks: A Tutorial*, vol. 4. Springer, Switzerland (2016). <https://doi.org/10.1007/978-3-319-26641-1>
27. Simon, P.L., Taylor, M., Kiss, I.Z.: Exact epidemic models on graphs using graph-automorphism driven lumping. *J. Math. Biol.* **62**(4), 479–508 (2011)
28. Wei, J., Kuo, J.C.W.: Lumping analysis in monomolecular reaction systems. analysis of the exactly lumpable system. *Ind. Eng. Chem. Fundam.* **8**(1), 114–123 (1969)



# Analytical Solution for Long Battery Lifetime Prediction in Nonadaptive Systems

Dmitry Ivanov<sup>1,2</sup>(✉), Kim G. Larsen<sup>2</sup>, Sibylle Schupp<sup>1</sup>, and Jiří Srba<sup>2</sup>

<sup>1</sup> Hamburg University of Technology, Hamburg, Germany  
{d.ivanov,schupp}@tuhh.de

<sup>2</sup> Department of Computer Science, Aalborg University, Aalborg, Denmark  
{kg1,srba}@cs.aau.dk

**Abstract.** Uppaal SMC is a state-of-the-art tool for modelling and statistical analysis of hybrid systems, allowing the user to directly model the expected battery consumption in battery-operated devices. The tool employs a numerical approach for solving differential equations describing the continuous evolution of a hybrid system, however, the addition of a battery model significantly slows down the simulation and decreases the precision of the analysis. Moreover, Uppaal SMC is not optimized for obtaining simulations with durations of realistic battery lifetimes. We propose an analytical approach to address the performance and precision issues of battery modelling, and a trace extrapolation technique for extending the prediction horizon of Uppaal SMC. Our approach shows a performance gain of up to 80% on two industrial wireless sensor protocol models, while improving the precision with up to 55%. As a proof of concept, we develop a tool prototype where we apply our extrapolation technique for predicting battery lifetimes and show that the expected battery lifetime for several months of device operation can be computed within a reasonable computation time.

## 1 Introduction

Battery lifetime is one of the main concerns in the usability evaluation of modern portable devices. Modelling of battery consumption can help to evaluate the expected power consumption prior to the actual construction and deployment of the system. Uppaal SMC [4] is a state-of-the-art tool that allows one to model and evaluate a behaviour of (stochastic) hybrid systems via a statistical model checking approach. Among others, it can be applied to modelling of battery-operated devices as abstract hybrid automata. We take a closer look at the possibilities of battery modelling in Uppaal SMC, focusing primarily on estimating the battery lifetime for nonadaptive systems, i.e., systems whose behaviour does not depend on the actual state of the battery.

A straightforward approach to modelling energy consumption is to annotate an existing Uppaal SMC model with the information in what locations energy is

consumed and by what rate, and to extend the model with a hybrid automaton representing the battery. However, this method has several practical drawbacks and limitations. First, hybrid modelling in Uppaal SMC is performed numerically with the Euler method [21] and this causes a large performance overhead and precision loss, in particular when we want to reason about long simulation runs lasting for months. Second, Uppaal SMC is not optimized for generating long lasting simulations that are necessary for estimating typical battery lifetimes. In applications like adhoc wireless networks where sensors and other network components are battery-operated, an average battery life is expected to last for several months, whereas simulating just a few hours of such a network operation in Uppaal SMC takes unaffordable computation time and memory.

We propose an analytical approach for computing the current battery charge. As we restrict ourselves to nonadaptive systems, we can first simulate the system's behaviour (without the battery model) with Uppaal SMC and then annotate the produced simulation trace with battery consumption, instead of adding a hybrid automaton of the battery to the Uppaal SMC model. We also replace the numerical method used in the tool by computing a precise analytical solution to the battery model equations based on the kinetic battery model (KiBaM) [17]. We demonstrate on two industrial-strength case studies that our approach considerably speeds up the simulations and improves the precision of battery prediction. By analytically solving the equations for polynomials of arbitrary degree, we enable an accurate modelling of the electrical current as an input to the battery model. Our experiments show that this is achievable within acceptable time and memory usage and significantly faster than a direct modelling in Uppaal SMC. Moreover, we propose a trace extension technique for the traces generated by Uppaal SMC in order to estimate battery lifetimes for several months of device operation. We apply this technique to two case studies that model two different wireless network protocols and show that our approach allows us to obtain simulation results within an acceptable time horizon, while at the same time extending the prediction precision.

*Related Work.* The research in battery modelling and lifetime optimization has several directions. Some of the existing battery models were extended to capture environmental parameters and usage patterns more precisely. Jongerden et al. [15] observed that KiBaM parameters change over time for rechargeable batteries. Their experiments show that the modelling error grows with the amount of the recharge cycles of the battery. Rodrigues et al. [20] proposed a temperature-dependent KiBaM (T-KiBaM). They applied the Arrhenius law to the parameter  $k$  of the KiBaM making it dependent on the temperature. Thus, T-KiBaM represents the intensity of the chemical processes inside the battery induced by the temperature. Various approaches exist for applying battery models to battery lifetime maximization problems. Model checking is one of such approaches. Bisgaard et al. [1] used Uppaal CORA for finding an optimal schedule for a satellite. However, they used a naive battery model instead of KiBaM because the model does not match the priced timed automata (PTA) formalism used by Uppaal CORA. Jongerden et al. [13] proposed a discretized version of KiBaM

for finding an optimal schedule with Uppaal CORA. The discretized model can be expressed as a PTA and thus passed to Uppaal CORA. They experimentally compared the accuracy of the discretized version to the original model. The experiments showed a deviation up to 1%. However, the comparison was done for parameter sets corresponding to batteries with very small capacities (92 mAh and 183 mAh). Moreover, the experiments were done for scenarios with very short battery lifetimes (less than 100 min). Hence, it is an open question whether this experimental setup is relevant for evaluating the modelling accuracy for much longer battery lifetimes.

Wognsen et al. [23] introduced a wear score function based on dynamic evolution of state-of-charge. Using the recent branch UPPAAL Stratego the score function was used to optimize the life time of a battery powered nano-satellite of the company GOMSpace. David et al. [5] applied Uppaal SMC to measure the power consumption of the LMAC protocol. No battery model was considered in this work. These two approaches are helpful for battery life prolongation, although they do not predict the lifetime. Boker et al. [2] introduced battery transition systems based on a discretized KiBaM. They proved that for a certain class of these models model checking is decidable but restricted to the discretized model. We consider the undiscretized version of KiBaM and use the analytical solution to it, which promises to be more precise. Heni et al. [11] model a mobile network as an on/off Markov decision chain and use a naive battery model. Panigrahi et al. [18] estimate the battery life of mobile embedded systems with a stochastic process (a discrete time Markov chain by Chiasserini and Rao [3]). In our work we complement Uppaal SMC models with the battery model, thus enabling the battery life prediction for more complicated systems. Xue et al. [24] introduce an energy saving mechanism for the IEEE 802.16e standard for mobile stations. They analyze the current performance by using a Markov chain model of the protocol but do not consider the KiBaM battery model. They evaluate their proposal by simulation in the discrete event simulation tool NS2 while we use a statistical model checker instead. Energy-aware software engineering [7] can model energy on different levels of abstraction: on the instruction set or the LLVM IR. It uses Horn clauses coupled with energy information. If the average values are used, the bounds on the energy consumption can be unsafe. Static energy profiling [10] is a variant of static analysis for energy consumption that gives probability distributions instead of static bounds. Unfortunately, these two approaches are not applicable for network protocol modeling when the source code is unavailable, especially on the design phase. We are not aware of any work on integrating an analytical solution to the existing modelling approaches and most of the methods above either cannot deal with long lifetime battery modelling or consider naive battery models. In our work, we allow to compute the precise analytical solution of KiBaM battery model for several months of system execution and we can provide precise estimates on battery lifetimes.

## 2 Battery Model

The simplest way to model a battery is to assume that its capacity is constant (independent from the consumption pattern) and its charge decreases proportionally with the drawn current. However, the main drawback of this approach is that it ignores nonlinear battery effects. In reality, the delivered battery capacity, or the real amount of energy the battery provides, depends on the actual profile of the current, which is also known as the *rate capacity effect*. Different battery models were developed taking this into account and they can be classified as electrochemical [6], electrical-circuit [9], stochastic [3], and analytical models [17, 19]. We choose to work with the analytical model KiBaM [17] as it is a relatively simple model and for most practical scenarios there is only a small precision gain when considering more advanced models like the diffusion model [14, 15].

### 2.1 Kinetic Battery Model

The Kinetic Battery Model (KiBaM) [17] captures nonlinear rate capacity effects by representing the battery as two communicating vessels. The charge is split into two parts: the available charge  $a$  and the bound charge  $b$ . The charge is modelled as a liquid stored in two interconnected vessels of width  $c$  and  $1 - c$  that represent the available and bound charges, respectively. The liquid levels correspond to the charges and are written as  $h_a$  and  $h_b$ . Figure 1 illustrates the model.

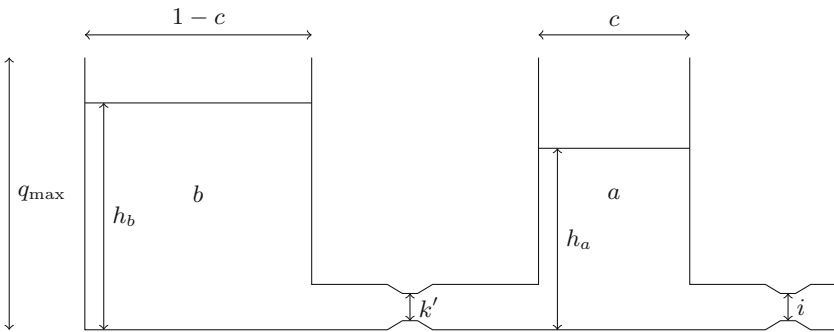


Fig. 1. KiBaM battery model

The available charge is taken on current  $i$  directly, and the bound charge supplies the available charge whenever  $h_a$  is smaller than  $h_b$ . The flow rate between the available and bound charges is proportional to the difference between their heights and the parameter  $k'$ . When the available charge is zero, the battery is



considered as discharged even though there may still be some bound charge left. Equation 1 formalizes the relationship between available and bound charge [17].

$$\begin{cases} \dot{a} = -i + k'(h_b - h_a) \\ \dot{b} = -k'(h_b - h_a) \end{cases} \quad (1)$$

As we can see from the model abstraction, the battery life depends on the energy consumption pattern. If energy is drawn rapidly then the available charge can become empty, having a significant amount of the bound charge left. Conversely, a slower energy consumption allows the available charge to restore and it reduces the amount of the bound charge left unused. Considering that  $h_a = \frac{a}{c}$  and  $h_b = \frac{b}{1-c}$ , and substituting  $k'$  with  $kc(1-c)$ , where  $0 \leq c \leq 1$  is the available charge fraction, we obtain Eq. 2.

$$\begin{cases} \dot{a} = -i + kcb - k(1-c)a \\ \dot{b} = -kcb + k(1-c)a \end{cases} \quad (2)$$

We can solve this equation for a constant current  $i$  by using Laplace transforms [17] and obtain

$$\begin{cases} a = a_0 e^{-kt} + \frac{(q_0 kc - i)(1 - e^{-kt}) - ic(kt - 1 + e^{-kt})}{k} \\ b = b_0 e^{-kt} + q_0(1-c)(1 - e^{-kt}) - \frac{i(1-c)(kt - 1 + e^{-kt})}{k} \end{cases} \quad (3)$$

where  $a_0$ ,  $b_0$ , and  $q_0$  are the available, bound, and full charge at the zero-time.

For evaluation purposes we also consider a naive battery model. It represents the battery as a single charge vessel from where the current is drawn. The naive model corresponds to KiBaM with  $c = 1$  and where  $k'$  becomes insignificant. A battery life computed for the naive model is a theoretical upper bound for the KiBaM battery life. The difference between them shows how much the battery life can be increased by changing the energy consumption pattern. For setting up the KiBaM parameters, we use the data sheet for the battery TADIRAN SL-750 [22]. For the parameter estimation we compute the discharge rate and lifetime pairs for model fitting as explained by Manwell et al. [17]. We use the obtained parameters  $c \approx 0.06$ ,  $k \approx 0.46 \text{ h}^{-1}$ ,  $q_0 \approx 1.17 \text{ Ah}$  in all our experiments.

## 2.2 Modeling KiBaM in Uppaal SMC

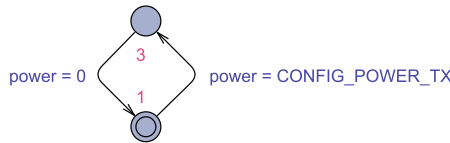
Uppaal SMC [4], the frontend of our work, is a statistical extension of the Uppaal model checker. It combats state space explosion by focusing on stochastic properties of the model. Uppaal models that are used for classical model checking can be adjusted to meet the requirements of Uppaal SMC, namely, input determinism, independent progress, and absence of priority between channels or processes. When all three requirements are met, one can apply both classical and statistical model checking. Uppaal SMC features probability evaluation, hypothesis testing, probability comparison, and simulation framework. In our work we focus primarily on generating simulations.

```

const double INIT_CHARGE = 4.2e+09,
           BATTERY_C = 0.06,
           BATTERY_K = 1.3e-07;
clock a = INIT_CHARGE * BATTERY_C,
       b = INIT_CHARGE * (1 - BATTERY_C);
    
```

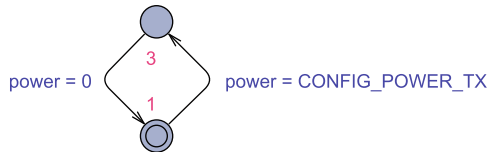
**Listing 1.1.** Uppaal declarations for a KiBaM

Let us consider a simple Uppaal SMC model with two locations shown in Fig. 2a. We assume that the power consumption is zero as long as the model stays in the initial location and equal to the current draw `CONFIG_POWER_TX` in the other location. The variable `power` keeps track of the current power consumption level. We now extend this model by introducing constant declaration in Listing 1.1 and by adding invariants, i.e., flow equations to the model locations as shown in Fig. 2b. The new declarations include the battery constants  $q_{max}$ ,  $c$ , and  $k$ , and clocks that store the available and bound charge values. We map the battery model constants to the constant double type because they do not change over time in our setup. The available charge and the bound charge have to be clocks since their evolution is continuous.



(a) Simple Uppaal SMC model

$$\begin{aligned}
 a' &== -power + BATTERY\_K * (b / (1 - BATTERY\_C) - a / BATTERY\_C) \ \&\& \\
 b' &== -BATTERY\_K * (b / (1 - BATTERY\_C) - a / BATTERY\_C)
 \end{aligned}$$



$$\begin{aligned}
 a' &== -BATTERY\_K * (b / (1 - BATTERY\_C) - a / BATTERY\_C) \ \&\& \\
 b' &== -BATTERY\_K * (b / (1 - BATTERY\_C) - a / BATTERY\_C)
 \end{aligned}$$

(b) Uppaal SMC model annotated with battery equations

**Fig. 2.** Uppaal SMC model and its annotation with KiBaM equations

This direct implementation of KiBaM in Uppaal has several practical drawbacks. First, Uppaal SMC solves the differential equations with the Euler method [4], which is less precise than the analytical solution. Second, the simulation of a hybrid model requires a considerable computational effort compared to the non-hybrid one (we will show this in the evaluation section) because Uppaal

SMC solves the flow equations numerically. The performance measurements from Table 1 show that the battery equations slow down the simulations even for our simple example by an approximate factor of 25 whereas our implementation of the analytical solution (described below) significantly improves the performance for longer simulations. Finally, a model time unit corresponds to one millisecond of the real time. Estimating battery life (which can last several months) therefore requires a very long simulation. This motivates our effort in finding an analytical solution to battery consumption as described in the next subsection.

**Table 1.** Simulation time in seconds for the model from Fig. 2

Simulation length (ms)	$10^4$	$10^5$	$10^6$	$10^7$
Model from Fig. 2a	$0.125 \pm 0.002$	$1.125 \pm 0.011$	$11.4 \pm 0.11$	$107 \pm 1.1$
Model from Fig. 2b	$2.7 \pm 0.02$	$26 \pm 0.2$	$268 \pm 2$	$2615 \pm 21$
Analytical solution to Fig. 2a	$1.6 \pm 0.05$	$4 \pm 0.1$	$30 \pm 0.6$	$457 \pm 9$

### 2.3 Analytical Solution to KiBaM

The KiBaM equations can be solved both numerically and analytically. A numerical solution, e.g., the Euler method, performs a discretization step on which the computations are performed. The discretization step should be reasonably small for precise results. In case of very long simulations, numerical methods become computationally expensive and imprecise due to error accumulations. In contrast, the analytical solution has no discretization step, i.e., the computations can be done only when the current level function changes. It means that the analytical solution can save the computational power on constant or polynomial pieces of the current. Moreover, the analytical solution does not accumulate the discretization error over the simulation time, which is beneficial for long simulations. Hence, we shall develop the analytical solution to KiBaM. In what follows, we provide solutions to KiBaM equations both in general and piecewise cases.

First, we solve Eq. 1 for an arbitrary current  $i$  in order to provide an analytical solution not only for a piecewise constant current intensity but also for piecewise polynomial intensities. Let  $i = i(t)$ , then Eq. 2 can be rewritten as follows.

$$\begin{cases} \dot{a} = -i(t) + kcb - k(1-c)a \\ \dot{b} = -kcb + k(1-c)a \end{cases} \quad (4)$$

We solve Eq. 4 by the Laplace transform and obtain the following.

$$\begin{cases} a = a_0 e^{-kt} + q_0 c(1 - e^{-kt}) - \int_0^t i(\tau) d\tau + (1-c) \int_0^t i(\tau)(1 - e^{k(\tau-t)}) d\tau \\ b = b_0 e^{-kt} + q_0(1-c)(1 - e^{-kt}) - (1-c) \int_0^t i(\tau)(1 - e^{k(\tau-t)}) d\tau \end{cases} \quad (5)$$

Equation 5 is the analytical solution to Eq. 2 for an arbitrary current  $i = i(t)$ . Next, we need to solve the integral  $\int_0^t i(\tau)(1 - e^{k(\tau-t)}) d\tau$  for a polynomial  $i(t)$

to obtain the solution to KiBaM for a polynomial current and substitute the polynomial  $i(t)$  in Eq. 5:

$$\left\{ \begin{array}{l} a = a_0 e^{-kt} + q_0 c (1 - e^{-kt}) - \sum_{j=0}^n \frac{i_j t^{j+1}}{j+1} \\ \quad + (1-c) \sum_{j=0}^n j! i_j \left[ \sum_{l=0}^{j+1} \frac{(-1)^l t^{j-l+1}}{k^l (j-l+1)!} + \frac{(-1)^j e^{-kt}}{k^{j+1}} \right] \\ b = b_0 e^{-kt} + q_0 (1-c) (1 - e^{-kt}) \\ \quad - (1-c) \sum_{j=0}^n j! i_j \left[ \sum_{l=0}^{j+1} \frac{(-1)^l t^{j-l+1}}{k^l (j-l+1)!} + \frac{(-1)^j e^{-kt}}{k^{j+1}} \right] \end{array} \right. \quad (6)$$

Equation 6 is now the analytical solution to Eq. 2 for a polynomial current  $i = \sum_{j=0}^n i_j t^j$ . For  $n = 0$  it coincides with Eq. 3 and for  $n = 1$ , or a linear current, Eq. 6 gives the final solution.

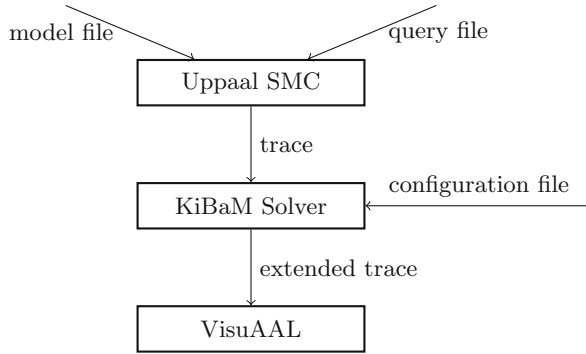
$$\left\{ \begin{array}{l} a = a_0 e^{-kt} + \frac{(q_0 k c - i_0)(1 - e^{-kt}) - i_0 c (k t - 1 + e^{-kt})}{k} - \frac{i_1 (1-c)(k t - 1 + e^{-kt})}{k^2} - \frac{c i_1 t^2}{2} \\ b = b_0 e^{-kt} + q_0 (1-c) (1 - e^{-kt}) - \frac{i_0 (1-c)(k t - 1 + e^{-kt})}{k} - \frac{i_1 (1-c)(k^2 t^2 / 2 - k t + 1 - e^{-kt})}{k^2} \end{array} \right. \quad (7)$$

Provided that the charge level of a battery does not affect the system behaviour, the KiBaM data can be computed independently based only on a given electric current function. Assuming that the current intensity is a piecewise constant function of time, we can use the analytical solution of Eq. 3 provided above to track the charge of the battery.

## 2.4 Tool Chain for KiBaM Battery Consumption

We assume that the protocols have been already modeled using the Uppaal SMC formalism. Our task is to design a backend to Uppaal SMC that captures the computed simulation trace (without the battery model but with annotated locations with their power consumption profile), computes the analytical solution to KiBaM, and transparently extends the simulation trace with available and bound charge values in the format used by Uppaal SMC. Such a way of extending the output allows to load the annotated trace back to Uppaal SMC but allows also for its visualization in other tools like VisuAAL [16] (for a demo see a video at <https://youtu.be/hGfMww97xWw> demonstrating a battery drain of network devices sharing the same protocol over the elapsed time). Moreover, we can interpret the simulation trace in a different way for solving other problems, in particular, for computing the battery lifetime or combining the trace with real data as discussed later. Figure 3 shows the data flow for our trace-extender tool. Uppaal SMC is launched in the command line mode having as input a model extended with information on its power consumption and a simulation query requesting the trace of the current. For example, the query

```
simulate 1 [<=1000]{power}
```



**Fig. 3.** KiBaM solver data flow

generates one simulation trace for the variable `power` lasting 1000 time units. Our extender then combines the settings, model parameters, and information on input and output expressions from a user-defined configuration file.

### 3 Case Study on Battery Charge in Wireless Networks

We present now two case studies in order to evaluate the performance of battery charge modelling. Our aim is to evaluate both precision and performance of our implementation of the analytical solution compared to the numerical solution provided in Uppaal SMC. We consider two wireless network protocols used in industry, namely the LMAC [8] and Neocortec’s MAC [12] protocols, both of them being examples of nonadaptive systems. The two protocols differ in the frequency of scheduled transmissions (LMAC is switching more often from sleep to power-consuming modes) and hence the power consumption of both protocols is different. The constants used in the Neocortec’s MAC protocol were modified in our model as the exact configuration of the protocol is not publicly available and hence our predicted battery lifetime in this case can differ from the performance of the actual devices sold by the company.

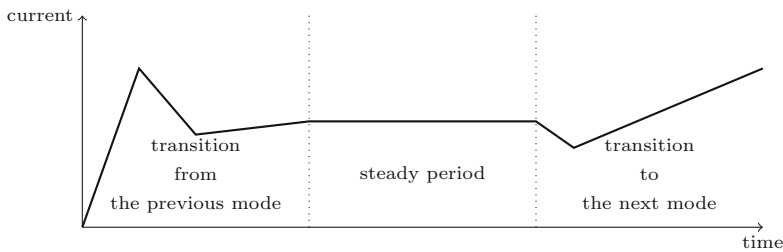
#### 3.1 Protocol Descriptions

The LMAC [8] protocol is a medium access control protocol for multi-hop, energy-constrained wireless sensor networks. This protocol addresses energy efficiency, self-configuration, and distributed operation. The main concept of the LMAC is scheduling. The protocol tries to avoid idle listening by dedicating a time slot to each node. Neocortec’s MAC protocol is an alternative attempt on designing an energy-efficient wireless network protocol. The patent [12] discloses some of the protocol details, however, a more detailed description was provided at [16]. We emphasize that this explanation is an approximation of the actual protocol behaviour and hence cannot be used for drawing conclusions

about the actual Neocortec implementation of the protocol. Neocortec’s MAC protocol is designed to broadcast data between neighbouring nodes and it synchronizes the nodes while performing beacon scans. As the LMAC, Neocortec’s MAC protocol benefits from scheduling the data transmission, however does so with a focus on energy preservation. We use both protocols for our evaluation purposes because they show two degrees of power change densities: Neocortec’s MAC protocol exhibits very sparse communications with long sleep periods, whereas the LMAC protocol has a more uniform activity with around 50 times more frequent power state changes than Neocortec’s MAC. Moreover, the protocols have different power consumption. Thus, in the setup explained in Sect. 4.2 the LMAC and Neocortec’s MAC consume the average current of 13 mA and 0.5 mA respectively.

### 3.2 Power Modes in Protocols

Power consumption modelling of network protocols by using power modes provides a simple yet precise abstraction of the underlying physical phenomenon. In our approach, a network protocol is represented as an automaton where each state is annotated with a power mode. For example, the Neocortec’s MAC protocol has the transmission (TX), the receive (RX), and the sleep (Sleep) power mode. The power modes represent the basic operations of the protocol (TX and RX) as well as their absence (Sleep). Usually, the current is assumed to be constant within a power state. This is a simplification of the actual profile of the current consumption and this simplification can have some bias regarding the current behaviour with frequent power mode shifts. Hence we propose a more precise interpretation of power modes capturing this effect.



**Fig. 4.** Three stages of a power mode

We distinguish three stages in a power mode: transition from the previous mode, a steady period, and transition to the next mode. Figure 4 illustrates this division (based on the actual measurements provided to us by Neocortec). We assume that the steady period is constant and can be of arbitrary duration. With this interpretation, we can use real measurements in our simulations and model the reality more precisely. By extending and compressing the steady

period we can fit an arbitrarily long power mode that appears in the simulation trace. However, our approach may restrict the shortest possible duration for an operation. Eliminating the steady period completely, we obtain the power mode consisting of transitions only. Its duration defines the lower bound of the power mode duration.

### 3.3 Experiments

We use the Uppaal SMC model of Neocortec’s MAC protocol described in [16] and the LMAC protocol model from [8]. We annotate the locations in the models with power consumption levels in order to evaluate the performance of the KiBaM model. We integrate the polynomial analytical solution into our tool KiBaM Solver, which is available on GitHub via link <https://github.com/DIvanov503/KiBaM-Solver.git>.

First, we evaluate how much the numerical solution computed by Uppaal SMC differs from the precise analytical solution, i.e., accumulates the error, as we increase the length of the simulation. We simulate the two protocols 50 times for 90,000 time units (90s of real time). Table 2 shows the evolution of the average absolute difference between the numerical and analytical solution with 95% confidence intervals. We can see that for both protocols, the difference grows over time as the numerical solution gets less and less precise. The difference for the LMAC protocol is on average higher than for Neocortec’s MAC because the more intensive power consumption of the LMAC protocol causes larger accumulated truncation errors of the Euler method. This relatively short simulation does not give us an estimate of the precision gain for the battery life estimation as the drop of batter capacity within 90s is negligible. However, assuming an average battery lifetime to be 100 days for Neocortec’s MAC and 3 days for the LMAC protocol, we can express the simulation time and the absolute error of the available and the bound charge as fractions of the battery life and initial charge values respectively. For Neocortec’s MAC the simulation models  $1 \cdot 10^{-3}\%$  of the average battery life, and the Euler method accumulates an error of  $5.5 \cdot 10^{-4}\%$  for the available charge, and  $3.5 \cdot 10^{-5}\%$  for the bound charge. The percentages for the LMAC protocol are  $3.5 \cdot 10^{-2}\%$ ,  $1.6 \cdot 10^{-3}\%$ , and  $1.1 \cdot 10^{-4}\%$  respectively. Assuming that the truncation error grows linearly, we can extrapolate our measurements to the duration of the average battery lifetime. Thus, we can predict the available and bound charge for the Neocortec’s MAC to have numerical errors of 55% and 3.5%. For the LMAC the estimate is better: 4.6% and 0.3%. We see that the relative error for Neocortec’s MAC gets very high. One explanation to this phenomenon is that the average battery lifetime is too long for the Euler method with the given discretization step. Same applies to the LMAC protocol experiment showing a smaller relative error because of the 8.5 times shorter average battery lifetime compared to Neocortec’s MAC. The difference between relative errors for the available and bound charges can arise because we use the parameter  $c \approx 0.06$  reflecting the available/bound charge ratio is very small meaning that the same absolute error is larger for the available charge than for the bound charge.

**Table 2.** Absolute difference between the Euler method and the analytical solution for available and bound charges

Simulation length (seconds)	10	30	50	70	90
Neocortec's MAC avail., mAms	$60 \pm 0.6$	$243 \pm 0.6$	$502 \pm 0.9$	$811 \pm 1.3$	$1401 \pm 1.8$
Neocortec's MAC bound, mAms	$66 \pm 0.1$	$270 \pm 0.4$	$527 \pm 0.7$	$837 \pm 1.1$	$1402 \pm 1.8$
LMAC avail., mAms	$85 \pm 0.3$	$788 \pm 2.5$	$2249 \pm 6.7$	$4442 \pm 13.0$	$8997 \pm 26.6$
LMAC bound, mAms	$92 \pm 0.2$	$833 \pm 2.2$	$2310 \pm 6.3$	$4520 \pm 12.5$	$9191 \pm 25.7$

Next, we compare the performance of the KiBaM model to the naive battery model. The naive model represents the battery as a single power vessel, which can be considered as a KiBaM with  $c = 1$  and non-significant  $k$ , simplifying the Uppaal SMC model for the naive battery to the one shown in Fig. 5 where `charge` represents the remaining charge of the battery. Table 3 shows the time required to simulate 10000 units of time (10s of real time) of the Uppaal SMC model annotated with the KiBaM and the naive battery model and compared to our analytical solution for KiBaM. The problems are scaled by the number of nodes in randomly generated network topologies.

The experiments show in case of Neocortec's MAC significantly faster running times of the analytical approach, compared to numerical solution. We also notice that the performance gain of the analytical solution depends on the den-

**Fig. 5.** Naive model of battery in Uppaal SMC**Table 3.** Time (in seconds) required for simulating 10,000 time units (average of 100 repetitions with 97% confidence interval)

# nodes	5	10	50	100
Neocortec's MAC, numeric. KiBaM	$54 \pm 0.5$	$119 \pm 0.8$	$686 \pm 6.6$	$2159 \pm 19.4$
Neocortec's MAC, numeric. naive	$42 \pm 0.3$	$87 \pm 0.4$	$544 \pm 4.2$	$1671 \pm 14.1$
Neocortec's MAC, analyt. KiBaM	$0.6 \pm 0.01$	$1.9 \pm 0.33$	$20.6 \pm 0.18$	$121.6 \pm 1.01$
LMAC, numeric. KiBaM	$22 \pm 0.1$	$45 \pm 0.1$	$288 \pm 1.0$	$713 \pm 2.1$
LMAC, numeric. naive	$0.8 \pm 0.02$	$2.1 \pm 0.02$	$41.6 \pm 0.23$	$145.7 \pm 0.59$
LMAC, analyt. KiBaM	$3 \pm 0.1$	$4 \pm 0.1$	$33 \pm 0.2$	$108 \pm 0.4$



sity of power mode changes—the more changes occur, the longer time it takes to compute the analytical solution. The data for LMAC indicate a lot smaller performance advantage of our analytical solution since the LMAC model changes the power state around 50 times more frequently than Neocortec’s MAC, however, the advantage becomes more obvious when we consider larger networks like e.g. 100 nodes in the table.

## 4 Simulation Traces for Estimating Long Battery Lifetime

In our approach we focus on nonadaptive protocols, meaning that the behaviour of the protocol/system does not depend on the actual charge of the battery. This allows us to extend an existing protocol model with the information on power modes and use a statistical model checker like Uppaal SMC to generate the trace for further analysis. However, the generation of battery lifetime-long simulations is problematic since the model time unit corresponds to one millisecond of the real time and that makes estimating battery life (which can last several months) to require a very long simulation exceeding the typical time and memory constraints. Hence, we designed an extrapolation technique that allows us to expand shorter simulation traces to longer ones.

### 4.1 Extension of Simulation Traces

We shall introduce an extrapolation technique where we first generate a time-bounded trace and repeat the latter half of it until the battery discharges. We exploit that network protocols have a warm-up period and a steady state. The warm-up period is not representative for the model behaviour in the long term. In contrast, the steady period has a nondeterministic regularity: the variable distributions are similar from time to time. If the steady period part of the trace is long enough to assume approximate statistical independence, it can be repeated to model the behaviour on arbitrarily long timeline. We use this idea to continue battery simulation until it fully discharges.

We assume the first half of the simulation to be a warm-up period due to the initial routines that are performed by the protocols. The fraction of the trace to be skipped as the warm-up period can be adjusted depending on the trace duration.

### 4.2 Experiments with Long Battery Lifetime Prediction

As a proof of concept, we perform two case studies investigating the battery lifetime of the already discussed protocols: Neocortec’s MAC and LMAC. We implement a battery life estimator on top of our KiBaM Solver. For battery life evaluation, we generate in VisuAAL a network consisting of 25 nodes with a random topology. We generate 100 traces of 10,000,000 time units corresponding to 2.8 h of real time in Uppaal SMC. Then we apply the trace extrapolation technique (Sect. 4.1) to continue the modelling up to the end of the battery lifetime.

We also investigate how a more complex power consumption modelling from Sect. 3.2 affects performance of battery lifetime prediction. We discretize current consumption graphs from real wireless sensor network nodes and manually divide power modes into stages. We interpolate the stages with splines and vary the number of points a spline interpolates and the maximal degree of splines. Thus, we construct a sequence of interpolation schemes of ascending complexity. First, we use constant splines and increase their number until each spline interpolates two points only. Then we replace the constant splines with linear ones. We perform these refinement steps for Neocortec’s MAC only because we do not have graphs for idle mode in LMAC.

**Table 4.** Execution time (in seconds) for generating a 10,000,000 time units long trace with Uppaal SMC and computing battery lifetimes with trace extrapolation and various current mode modelling approaches

Method	Protocol	
	Neocortec’s MAC	LMAC
trace	$1852 \pm 6$	$8746 \pm 153$
const	$238 \pm 1$	$535 \pm 9$
const6	$2127 \pm 9$	—
const5	$2551 \pm 12$	—
const4	$3624 \pm 13$	—
const3	$6178 \pm 27$	—
const2	$7428 \pm 41$	—
lin2	$11814 \pm 49$	—

Table 4 shows the computation time required for generating an input trace with Uppaal SMC and predicting the battery lifetime. Here “const” is a usual piecewise constant approach without distinguishing stages in power modes, “const $i$ ” stands for a refined piecewise constant interpolating  $i$  points with a single constant function, i.e., a smaller  $i$  corresponds to a more complex interpolation, “lin $i$ ” is same as “const $i$ ” but uses linear splines. We emphasize that computation of the KiBaM solution takes time comparable to generating a relatively short trace in Uppaal SMC but reaches a much longer prediction horizon: between 83 and 123 days for Neocortec’s MAC and between 1.8 and 4.7 days for the LMAC. The table also shows the performance of the battery life estimation involving a more precise power mode model. While decreasing  $i$  in “const $i$ ” we add constant steps to the transition state approximations and finally switch to linear splines in “lin2.” Our observations show that our trace extrapolation method allows for using more complex input for a reasonable performance penalty but most importantly, by the combination of our techniques, we are able to predict the battery lifetime in the duration of several months.

## 5 Conclusion and Future Work

We investigated two approaches to improve the precision and performance of battery modelling in hybrid automata simulation tools like Uppaal SMC. We developed an analytical solution of the flow equations allowing us to use a separate postprocessing unit and we implemented a tool for computing the analytical solution to the kinetic battery model. Moreover, we proposed a method of computing the battery lifetime by extrapolating simulation traces. This technique allows estimating battery lifetimes without simulating the model for the whole battery discharge period, which is infeasible in practice. As a proof of concept we applied our implementation of the analytical solution for comparing the battery lifetimes of two energy-efficient wireless sensor network protocols: LMAC and Neocortec's MAC.

As a part of the detailed current modelling, we solved the KiBaM equations in the general case. From the general solution we obtained the solution for polynomial currents and we proposed a method for using real current measurements with simulation traces. The evaluation part of our work included various setups. We showed that our implementation of the analytical solution has a better performance than the numerical one and we demonstrated that the difference between the numerical and the analytical solution grows over time.

We suggest several directions of the future work. First, we can continue refining the modelling approach. The trace extrapolation technique we use now is rather simplistic. More advanced statistical methods can be applied to our problem. For instance, one can try to obtain a discrete-time Markov chain from the simulation trace and perform the battery life estimation stochastically, similarly to stochastic battery models by Chiasserini and Rao [3]. Second, we can use a more advanced battery model. For example, we can take the temperature-dependent KiBaM [20] to capture the temperature influence. Furthermore, we can investigate the impact on the battery lifetime of various protocol optimizations, e.g. the node can nondeterministically select the neighbours it is going to interact with and ignore the others. Such an optimization can reduce the power consumption of the node but also can deteriorate the network characteristics.

**Acknowledgements.** We thank Kevin H. Jørgensen, Niels B. Christoffersen, Rasmus D. Petersen and Tim H. Gjøderum for their help with integrating our tool with VisuAAL and numerous discussions about the annotation of battery modes in Uppaal SMC models of the two wireless network protocols. We thank Neocortec for allowing us to use their MAC protocol in our experiments and Thomas Steen Halkier for discussions about the battery consumption patterns and for providing us with current consumption graphs. The work was funded by the center IDEA4CPS, the Innovation Fund Denmark center DiCyPS and ERC Advanced Grant LASSO. The last author is partially affiliated with FI MU, Brno.

## References

1. Bisgaard, M., Gerhardt, D., Hermanns, H., Krčál, J., Nies, G., Stenger, M.: Battery-aware scheduling in low orbit: the GomX-3 case. In: Fitzgerald, J., Heitmeyer, C., Gnesi, S., Philippou, A. (eds.) FM 2016. LNCS, vol. 9995, pp. 559–576. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-48989-6\\_34](https://doi.org/10.1007/978-3-319-48989-6_34)
2. Boker, U., Henzinger, T.A., Radhakrishna, A.: Battery transition systems. In: ACM SIGPLAN Notices, vol. 49, pp. 595–606. ACM (2014)
3. Chiasserini, C.F., Rao, R.R.: Energy efficient battery management. *IEEE J. Sel. Areas Commun.* **19**(7), 1235–1245 (2001)
4. David, A., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B.: Uppaal SMC tutorial. *Int. J. Softw. Tools Technol. Transf.* **17**(4), 397–415 (2015)
5. David, A., et al.: Statistical model checking for networks of priced timed automata. In: Fahrenberg, U., Tripakis, S. (eds.) FORMATS 2011. LNCS, vol. 6919, pp. 80–96. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-24310-3\\_7](https://doi.org/10.1007/978-3-642-24310-3_7)
6. Doyle, M., Fuller, T.F., Newman, J.: Modeling of galvanostatic charge and discharge of the lithium/polymer/insertion cell. *J. Electrochem. Soc.* **140**(6), 1526–1533 (1993)
7. Eder, K., Gallagher, J.: Energy-aware software engineering. In: ICT - Energy Concepts for Energy Efficiency and Sustainability, pp. 103–127. InTechOpen (2017)
8. Fehnker, A., van Hoesel, L., Mader, A.: Modelling and verification of the LMAC protocol for wireless sensor networks. In: Davies, J., Gibbons, J. (eds.) IFM 2007. LNCS, vol. 4591, pp. 253–272. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-73210-5\\_14](https://doi.org/10.1007/978-3-540-73210-5_14)
9. Gold, S.: A PSPICE macromodel for lithium-ion batteries. In: Twelfth Annual Battery Conference on Applications and Advances, pp. 215–222. IEEE (1997)
10. Haemmerlé, R., López-García, P., Liqat, U., Klemen, M., Gallagher, J.P., Hermenegildo, M.V.: A transformational approach to parametric accumulated-cost static profiling. In: Kiselyov, O., King, A. (eds.) FLOPS 2016. LNCS, vol. 9613, pp. 163–180. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-29604-3\\_11](https://doi.org/10.1007/978-3-319-29604-3_11)
11. Heni, M., Bouallegue, A., Bouallegue, R.: Energy consumption model in ad hoc mobile network. *Int. J. Comput. Netw. Commun.* **4**(3), 207–217 (2012)
12. Jaeger, M.G.B.: Wireless network medium access control protocol. <https://www.google.com/patents/US20120120871>. US Patent App. 12/945,989 (2012)
13. Jongerden, M., Haverkort, B., Bohnenkamp, H., Katoen, J.P.: Maximizing system lifetime by battery scheduling. In: Proceedings of 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, pp. 63–72. IEEE (2009)
14. Jongerden, M.R., Haverkort, B.R.: Which battery model to use? *IET Software* **3**(6), 445–457 (2009)
15. Jongerden, M.R., Haverkort, B.R.: Battery aging, battery charging and the kinetic battery model: a first exploration. In: Bertrand, N., Bortolussi, L. (eds.) QUEST 2017. LNCS, vol. 10503, pp. 88–103. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66335-7\\_6](https://doi.org/10.1007/978-3-319-66335-7_6)
16. Jørgensen, K.H., Christoffersen, N.B., Petersen, R.D., Gjøderum, T.H.: VisuAAL - an application for visualizing realistic mesh network protocol behavior through Uppaal simulations, Master Thesis. Aalborg University, Department of Computer Science (2017)
17. Manwell, J.F., McGowan, J.G.: Lead acid battery storage model for hybrid energy systems. *Solar Energy* **50**(5), 399–405 (1993)

18. Panigrahi, D., Dey, S., Rao, R., Lahiri, K., Chiasserini, C., Raghunathan, A.: Battery life estimation of mobile embedded systems. In: Fourteenth International Conference on VLSI Design 2001, pp. 57–63. IEEE (2001)
19. Rakhmatov, D.N., Vruthula, S.B.: An analytical high-level battery model for use in energy management of portable electronic systems. In: Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, pp. 488–493. IEEE (2001)
20. Rodrigues, L.M., Montez, C., Moraes, R., Portugal, P., Vasques, F.: A temperature-dependent battery model for wireless sensor networks. *Sensors* **17**(2), 422 (2017)
21. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis, vol. 12. Springer, New York (2013). <https://doi.org/10.1007/978-0-387-21738-3>
22. TADIRAN SL-750 Data Sheet. <https://tadiranbatteries.de/pdf/lithium-thionyl-chloride-batteries/SL-750.pdf>. Accessed 12 June 2018
23. Wognsen, E.R., Haverkort, B.R., Jongerden, M., Hansen, R.R., Larsen, K.G.: A score function for optimizing the cycle-life of battery-powered embedded systems. In: Sankaranarayanan, S., Vicario, E. (eds.) FORMATS 2015. LNCS, vol. 9268, pp. 305–320. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-22975-1\\_20](https://doi.org/10.1007/978-3-319-22975-1_20)
24. Xue, J., Yuan, Z., Zhang, Q.: Traffic load-aware power-saving mechanism for IEEE 802.16 E sleep mode. In: Proceedings of 4th International Conference on Wireless Communications, Networking and Mobile Computing 2008, pp. 1–4. IEEE (2008)



# Action and State Based Computation Tree Measurement Language and Algorithms

Yaping Jing<sup>1</sup>(✉) and Andrew S. Miner<sup>2</sup>

<sup>1</sup> Salisbury University, Salisbury, MD, USA  
yxjing@salisbury.edu

<sup>2</sup> Iowa State University, Ames, IA, USA  
asminer@iastate.edu

**Abstract.** The computation tree measurement language (CTML) is a real-valued specification formalism, designed to express performance measures and dependability properties in a single framework. It is a further extension of probabilistic model checking logic in such a way that it expands the input and output value from the range of  $[0, 1]$  to the range of  $[0, \infty)$ . Unlike probabilistic computation tree logic (PCTL) or continuous stochastic logic (CSL), CTML can *nest* real values. As such, it turns out that CTML can express a nontrivial subset of probabilistic linear time logic (PLTL) formulas that cannot be expressed by PCTL while keeping the overall computational complexity being polynomial in the size of both an input formula and model. Moreover, it can express queries such as “when a message is sent, what is the expected time until it is received?” that cannot be expressed by an existing probabilistic logic, because they are “probabilistic” at most. While powerful, CTML is a state-based specification formalism. In this work, we introduce a stronger formal query language, called *action and state based computation tree measurement language (asCTML)*. asCTML extends from CTML, for answering performance queries that operates over paths featured by a combination of states and actions. Inspired by the action and state based continuous stochastic logic (asCSL), asCTML supports multiple actions.

**Keywords:** Markov chain · Performance measures  
Probabilistic model checking

## 1 Introduction

Previously, we introduced the *computation tree measurement language* (CTML) [1] that is designed to express performance measures and dependability properties in a single framework. It takes as inputs a probabilistic model structure with real-valued “rewards” functions [2] and a real-valued specification formalism and outputs a real value. CTML differentiates from probabilistic model checking logics in such a way that it takes a real value in the range of  $[0, \infty)$

as input, and output a real value in the range of  $[0, \infty)$ . As such, it can *nest* real values, in a sense that it does not require to convert a quantitative result from an intermediate formula to 0/1 (or true/false) value before an outer-level formula is being solved. Comparing to the probabilistic model checking formalisms, CTML has at least one of the following advantages: (1) It covers PCTL [3]. (2) It can express a nontrivial subset of PLTL [4] formulas that cannot be expressed by PCTL while keeping the overall computational complexity being polynomial in both the size of an input formula and model. (3) It can express queries such as “when a message is sent, what is the expected time until it is received?”; this type of queries cannot be expressed by an existing probabilistic specification formalism, because they are “probabilistic” at most.

While powerful, CTML is a state-based specification formalism. For action driven performance queries, it is either not intuitive or simply not expressible, particularly when considering a model with multiple actions between the same pair of states. Inspired by asCSL [5], in this work, we introduce a new formalism, called *action and state based computation tree measurement language* (asCTML), that extends from CTML, in support of performance queries that operate over a combination of states and actions. As such, the expressive power expands significantly. Take the classic dining philosophers model [6] as an example, with asCTML, it is natural to express survivability queries such as: “given a philosopher is *hungry*, how much amount of *food* that the philosopher consumes on average before he/she eventually *releases* the forks?”, where *food* is a state formula representing rate rewards, *hungry* and *release* are action formulas representing impulse rewards (i.e., a real value assigned to an action received at its corresponding state) [2, 7].

The remainder of the paper is organized as follows. Section 2 presents foundations for asCTML measures. Section 3 presents the syntax and semantics of the proposed language. Section 4 presents a translation algorithm from asCTML to CTML. Section 5 presents an application example and experimental results. Section 6 discusses related work. Section 7 concludes the paper and gives some directions for future work.

## 2 Foundations

In this section, we recall some well-known definitions, introduce notation, and define the underlying structures necessary for this work.

A (time-homogeneous) discrete-time Markov chain (DTMC) [8] is a tuple  $(\mathcal{S}, \mathbf{P}, \boldsymbol{\pi}_0)$ , where  $\mathcal{S} = \{0, \dots, N\}$  is a finite set of states with  $N \in \mathbb{N}$ ,  $\boldsymbol{\pi}_0 : \mathcal{S} \rightarrow [0, 1]$  is an initial probability distribution under the constraint  $\sum_{i \in \mathcal{S}} \boldsymbol{\pi}_0(i) = 1$ , and  $\mathbf{P} : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$  is a stochastic matrix.

**Definition 1 (MAMC structure).** *A DTMC with multiple actions, or MAMC for short, is a tuple  $(\mathcal{S}, \mathcal{ACT}, \gamma, \delta, \boldsymbol{\pi}_0)$ , where*

- $\mathcal{S}$  is a finite set of states.
- $\mathcal{ACT}$  is a finite set of action symbols.

- $\gamma : \mathcal{S} \times \mathcal{ACT} \rightarrow \mathcal{S}$  specifies the state transitions.
- $\delta : \mathcal{S} \times \mathcal{ACT} \rightarrow [0, 1]$  specifies the transition probabilities, under the constraint that  $\forall s \in \mathcal{S}, \delta(s, \mathcal{ACT}) = \sum_{a \in \mathcal{ACT}} \delta(s, a) = 1$ .
- $\pi_0 : \mathcal{S} \rightarrow [0, 1]$  is an initial probability distribution with  $\sum_{s \in \mathcal{S}} \pi_0[s] = 1$ .

Note that by the definition of  $\gamma$ , if  $(s, a) = (s, b)$ , then  $\gamma(s, a) = \gamma(s, b)$  and  $t = t'$ , where  $t, t'$  are the next states that  $a, b$  lead to, respectively. That is, given a state  $s$ , different next states implies different actions associated with  $s$ .

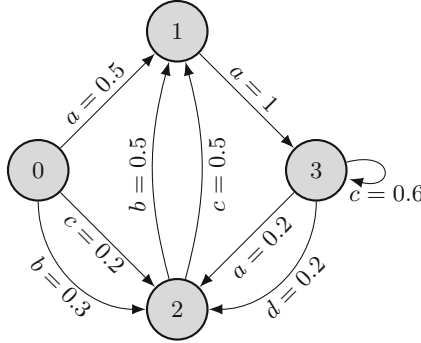


Fig. 1. An example of MAMC structure

Figure 1 shows an example of MAMC structure, with  $\mathcal{S} = \{0, 1, 2, 3\}$ ,  $\mathcal{ACT} = \{a, b, c, d\}$ ; at state 0, action  $a$  occurs, leading to state 1, with probability 0.5; when action  $b$  occurs, leading to state 2, with probability 0.3; when action  $c$  occurs, leading to state 2, with probability 0.2, so on and so forth.

**Definition 2 (path in MAMC).** Let  $Y = (\mathcal{S}, \mathcal{ACT}, \Gamma, \pi_0)$  be a MAMC structure, a path in  $Y$  is an infinite sequence  $\pi = (s_0, a_0), (s_1, a_1), \dots \in (\mathcal{S} \times \mathcal{ACT})^\omega$ , with  $\pi_i = (s_i, a_i)$ ,  $\delta(s_i, a_i) > 0$ ,  $s_{i+1} = \gamma(s_i, a_i)$ , and  $i \in \mathbb{N}$ .

**Definition 3 (path formula).** Given a MAMC structure  $Y$ , a path formula is a function  $\psi : (\mathcal{S} \times \mathcal{ACT})^\omega \rightarrow \mathbb{R}^*$ , where  $\mathbb{R}^*$  denotes nonnegative reals.

**Definition 4 (prefix).** A prefix in an MAMC structure  $Y$  is a finite sequence of state, action pairs

$$\mathbf{p} = (s_0, a_0), \dots, (s_{n-1}, a_{n-1}) \in (\mathcal{S} \times \mathcal{ACT})^n,$$

or an infinite sequence  $\mathbf{p} = (s_0, a_0), (s_1, a_1) \dots \in (\mathcal{S} \times \mathcal{ACT})^\omega$ , where  $\mathbf{p}_i$  is the  $i^{\text{th}}$  element in  $\mathbf{p}$ , and  $|\mathbf{p}| = n \in \mathbb{N} \cup \{\omega\}$  is the length of the sequence.

Let  $\Omega^\omega$  denote a set of paths in MAMC. For a given prefix  $\mathbf{p}$ , define  $\Omega_{\mathbf{p}}^\omega$  as the set of all infinite length paths that start with prefix  $\mathbf{p}$ . If  $|\mathbf{p}| = n \in \mathbb{N}$ , we have

$$\Omega_{\mathbf{p}}^\omega = (s_0, a_0) \times \dots \times (s_{n-1}, a_{n-1}) \times \Omega^\omega \tag{1}$$

otherwise, if  $|\mathbf{p}| = \omega$ , then we have  $\Omega_{\mathbf{p}}^\omega = \{\mathbf{p}\}$ .



**Definition 5 (determines  $\psi$  on MAMC).** We say a prefix  $\mathbf{p}$  determines  $\psi$  if, for any paths  $\mathbf{x}, \mathbf{x}' \in \Omega_{\mathbf{p}}^{\omega}$ ,  $\psi(\mathbf{x}) = \psi(\mathbf{x}')$ ; since all paths must have the same value for  $\psi$  in this case, we denote this quantity as  $\psi(\mathbf{p})$ . Note that any infinite prefix determines  $\psi$ .

**Definition 6 (finitely measurable  $\psi$  on MAMC).** We say a path formula  $\psi$  is finitely measurable on a MAMC structure  $Y$  if, for every path  $\mathbf{x} \in \Omega_{\mathbf{p}}^{\omega}$  with  $\psi(\mathbf{x}) > 0$ , either there exists a finite prefix  $\mathbf{p}$  with  $\mathbf{x} \in \Omega_{\mathbf{p}}^{\omega}$  that determines  $\psi$ , or the probability measure for path  $\mathbf{x}$  is zero.

Like CTML, we wish to define a measure of the expected value of a path formula  $\psi$ . Let  $\mathcal{X}$  be any set. A  $\sigma$ -algebra is a subset  $\mathcal{G}$  of the power set of a set  $\mathcal{X}$  such that:

1.  $\mathcal{X}, \emptyset \in \mathcal{G}$ .
2. if  $C \in \mathcal{G}$ , then  $\mathcal{X} \setminus C \in \mathcal{G}$ . That is,  $\mathcal{G}$  is closed under complementation.
3. if  $C_0, C_1, \dots$  with  $C_i \in \mathcal{G}$ , then  $\bigcup_{i=0}^{\infty} C_i \in \mathcal{G}$ .

An important property is that, for any collection  $\mathcal{C}$  of subsets of  $\mathcal{X}$ , there is a unique smallest  $\sigma$ -algebra on  $\mathcal{X}$  that includes (i.e., is a superset of)  $\mathcal{C}$ , called the  $\sigma$ -algebra *generated by*  $\mathcal{C}$ . Then, a function  $\mu : \mathcal{G} \rightarrow [0, \infty]$  is a *measure* if  $\mu(\emptyset) = 0$ , and for any sequence  $C_0, C_1, \dots$  of disjoint sets in  $\mathcal{G}$ ,

$$\mu \left( \bigcup_{i=0}^{\infty} C_i \right) = \sum_{i=0}^{\infty} \mu(C_i). \quad (2)$$

A more rigorous treatment of measure theory may be found for example in [9].

**Definition 7 (measure of the expected value of  $\psi$  on MAMC).** For any finitely measurable formula  $\psi$  on  $Y = (\mathcal{S}, \mathcal{ACT}, \gamma, \delta, \pi_0)$ , we define the measure of the expected value  $\mu_{\psi} : \mathcal{G}_{\mathcal{S} \times \mathcal{ACT}} \rightarrow \mathbb{R}^*$  by

$$\mu_{\psi}(\Omega_{\mathbf{p}}^{\omega}) = \begin{cases} \psi(\mathbf{p}) \prod_{i=1}^{|\mathbf{p}|-1} \delta(s_i, a_i) & \text{if } \mathbf{p} = ((s_0, a_0), (s_1, a_1), \dots) \text{ determines } \psi \\ \sum_{(s,a) \in \mathcal{S} \times \mathcal{ACT}} \mu_{\psi}(\Omega_{(\mathbf{p}, (s,a))}^{\omega}) & \text{otherwise} \end{cases}$$

with  $\mu_{\psi}(\emptyset) = 0$ .

Note that in this measure definition of the extended model of MAMC, since the starting point is now a pair of state, action, the probability of the initial action associated with  $(s_0, a_0)$  is not included, rather, it is treated as part of the initial distribution and captured at the end.

Similar to CTML, for this work, every path formula  $\psi$  is finitely measurable. The finitely measurable property helps us to avoid the difficulty of having to observe an infinitely-long prefix to determine the value of  $\psi$ , since any such path with  $\mathbf{p} = \pi$  must have  $\mu_{\psi}(\Omega_{\pi}^{\omega}) = 0$ .

### 3 Action and State Based Computation Tree Measurement Language

In this section, we introduce a new formal language called *Action and state based computation measurement language*, or asCTML for short. asCTML is designed to express performance queries based on action and/or state properties over probabilistic systems. asCTML is syntactically the same, but semantically more powerful than CTML, for supporting multiple actions, and paths featured by a combination of states and actions. The main differences between asCTML and CTML are in their semantics, the interpretation structure, the underlying paths, and the basic set of atomic functions. The new language is largely inspired by asCSL [5] and CSL<sup>TA</sup> [10], for reasoning about both state and action properties over probabilistic systems. Unlike asCSL and CSL<sup>TA</sup>, however, this language works naturally with the performance measures of *reward* functions [2, 7, 11] that are used to be specified within high level models. In the following, we first give the definition of the basic formulas. Then, we present syntax and semantics of the languages.

**Definition 8 (state+action formula).** *A state+action formula  $\varphi$  is defined as a function that maps from a pair of state and action to a nonnegative real value.*

$$\varphi : \mathcal{S} \times \mathcal{ACT} \rightarrow \mathbb{R}^*$$

**Definition 9 (restricted state+action formula).** *A restricted state+action formula  $\varphi_r$  is defined as a function that maps from a pair of state and action to a real value in interval  $[0, 1]$ .*

$$\varphi_r : \mathcal{S} \times \mathcal{ACT} \rightarrow [0, 1]$$

In principle, a state+action formula  $f$  ranges over  $\mathcal{S}$  and  $\mathcal{ACT}$ , meaning they are defined on every pair of  $(s, a) \in \mathcal{S} \times \mathcal{ACT}$ ; in practice, however, only the pairs with positive value of  $f$  are given explicitly. To this end, we assume the following sets of atomic items that are required for the specification of asCTML, but not required to be tied to a model.

- a finite set  $\mathcal{AF}$  of atomic *state+action* formulas,
- $\mathcal{AR} \subseteq \mathcal{AF}$  of restricted atomic *state+action* formulas.

For convenience, we also assume *one, zero*  $\in \mathcal{AR}$ , that evaluate to 1 and 0, respectively, over any element  $(s, a) \in (\mathcal{S} \times \mathcal{ACT})$ .

#### 3.1 asCTML Syntax

Given the definition of atomic *state+action* formula, the syntax of asCTML and the meanings of operators  $M$  (“expected value” of a path formula),  $X$  (“next”),  $U$  (“until”), and  $V$  (“weak until”) are kept the same as CTML, except:

- asCTML operates on state+action formulas, not merely state formulas.
- asCTML supports paths of combination of states and actions, whereas CTML supports paths of sequences of states only.
- asCTML distinguishes among multiple actions, whereas CTML does not.

**Definition 10 (syntax of asCTML).** *Let  $f$  be a state+action formula, let  $r$  be a restricted state+action formula, and let  $\psi_r$  be a restricted path formula defined as  $\psi_r : (\mathcal{S} \times \mathcal{ACT})^\omega \rightarrow [0, 1]$ , the syntax of asCTML can be recursively defined as follows:*

$$\begin{aligned}
 \varphi &::= f \mid \varphi_r \mid \varphi \odot \varphi \mid M\psi \\
 \varphi_r &::= r \mid 1 - \varphi_r \mid \varphi \bowtie \varphi \mid \varphi_r \times \varphi_r \mid M\psi_r \\
 \psi &::= X\varphi \mid \varphi U_{\odot}^{\leq t} \varphi \mid \varphi V_{\odot}^{\leq t} \varphi \mid \varphi U_+ \varphi \mid \varphi_r U_{\times} \varphi \mid \varphi_r V_{\times} \varphi \\
 \psi_r &::= X\varphi_r \mid \varphi_r U_{\odot}^{\leq t} \varphi_r \mid \varphi_r V_{\odot}^{\leq t} \varphi_r \mid \varphi_r U_{\times} \varphi_r \mid \varphi_r V_{\times} \varphi_r
 \end{aligned}$$

where  $\bowtie \in \{\leq, \geq, <, >\}$ ,  $\odot \in \{+, \times\}$ ,  $f \in \mathcal{AF}$ , and  $f_r \in \mathcal{AR}$ .

Finally we note that asCTML's top level formula is a *state+action* formula. Also, for the same rationale as CTML, the unbounded operators  $\varphi_r U_{\times} \varphi$  and  $\varphi_r V_{\times} \varphi$  require that  $\varphi_r$  is restricted to values no greater than one.

### 3.2 Semantics of asCTML

We now give the formal semantics of the operators appearing in the language. Unlike model checking logics such as PCTL or asCSL, which define a satisfaction relation, asCTML formulas are defined as real-valued functions.

**Definition 11 (Semantics of asCTML).** *Semantics of asCTML are inductively defined as follows:*

- If  $\varphi = f$ , then  $\varphi(s, a) = f(s, a)$ ,  $\forall f \in \mathcal{AF}$ .
- If  $\varphi = \varphi_1 \times \varphi_2$ , then  $\varphi(s, a) = \varphi_1(s, a) \times \varphi_2(s, a)$ .
- If  $\varphi = \varphi_1 + \varphi_2$ , then  $\varphi(s, a) = \varphi_1(s, a) + \varphi_2(s, a)$ .
- If  $\varphi = \varphi_1 \bowtie \varphi_2$ , with  $\bowtie \in \{>, \geq, <, \leq\}$  then  $\varphi(s, a) = 1$  if  $\varphi_1(s, a) \bowtie \varphi_2(s, a)$  holds;  $\varphi(s, a) = 0$  otherwise.
- If  $\varphi = 1 - \varphi_1$ , then  $\varphi(s, a) = 1 - \varphi_1(s, a)$ .
- If  $\varphi = M\psi$ , then  $\varphi(s, a) = \mu_{\psi}(\Omega_{(s,a)}^{\omega})$ .
- If  $\psi = X\varphi$ , then  $\psi(\pi_0, \pi_1, \dots) = \varphi(\pi_1)$ ,
- If  $\psi = \varphi_1 U_{\odot}^{\leq t} \varphi_2$ , then
  - if  $\exists j \leq t$ ,  $\varphi_2(\pi_j) > 0$ , and  $\forall i < j$ ,  $\varphi_2(\pi_i) = 0$ ,

$$\psi(\pi_0, \pi_1, \dots) = \left( \bigodot_{i=0}^{j-1} \varphi_1(\pi_i) \right) \cdot \varphi_2(\pi_j)$$

- otherwise,  $\psi(\pi_0, \pi_1, \dots) = 0$ .

Also, we have  $\varphi_1 U_{\odot}^{\leq \infty} \varphi_2 \equiv \varphi_1 U_{\odot} \varphi_2$ .

- If  $\psi = \varphi_1 V_{\odot}^{\leq t} \varphi_2$ , then

- if  $\exists j \leq t$ ,  $\varphi_2(\pi_j) > 0$ , and  $\forall i < j$ ,  $\varphi_2(\pi_i) = 0$ ,

$$\psi(\pi_0, \pi_1, \dots) = \left( \bigodot_{i=0}^{j-1} \varphi_1(\pi_i) \right) \cdot \varphi_2(\pi_j)$$

- otherwise, if  $\forall i \leq t$ ,  $\varphi_2(i) = 0$  then

$$\psi(\pi_0, \pi_1, \dots) = \bigodot_{i=0}^t \varphi_1(\pi_i)$$

Also, we have  $\varphi_1 V_{\times}^{\leq \infty} \varphi_2 \equiv \varphi_1 V_{\times} \varphi_2$ .

Note that the finitely measurable property of asCTML path formula  $\psi$  can be proved exactly the same CTML [1].

Finally, we define the measure of the top-level formula  $\varphi$  on MAMC structure  $M$  based on the initial distribution as follow.

**Definition 12 (measure of  $M_{\varphi}$ ).** Given a MAMC structure  $M = (\mathcal{S}, \mathcal{ACT}, \gamma, \delta, \pi_0)$ , the total value of a state+action formula  $\varphi$  on  $M$  is defined by

$$M_{\varphi} = \sum_{s \in \mathcal{S}} \left( \sum_{a \in \mathcal{ACT}} \varphi(s, a) \cdot \delta(s, a) \right) \cdot \pi_0(s). \quad (3)$$

## 4 asCTML Computation

The computation of asCTML can be reduced to the computation of CTML [1] via a translation algorithm described below. The algorithm takes a MAMC structure and a set of asCTML formula, and produces a DTMC structure and a corresponding set of CTML formula. It is then proved that the translated DTMC+CTML formula always gives the same value as the MAMC+asCTML formula.

**MAMC+asCTML to DTMC+CTML Translation Algorithm:**

- **MAMC to DTMC:** Given a MAMC structure  $M = (\mathcal{S}, \mathcal{ACT}, \gamma, \delta, \pi_0)$ , a corresponding DTMC  $D = (\mathcal{S}', \mathbf{P}, \pi'_0)$  can be built by the following:

- $\mathcal{S}' = \{(sa) \mid (sa) \in \mathcal{S} \times \mathcal{ACT}, \delta(s, a) > 0\}$ .
- For all  $(sa), (s'a') \in \mathcal{S}'$ ,

$$\mathbf{P}[(sa), (s'a')] = \begin{cases} \delta(s', a') & \text{if } \gamma(s, a) = s', \\ 0 & \text{otherwise.} \end{cases}$$

- $\pi'_0[(sa)] = \pi_0[s] \cdot \delta(s, a)$ .

Given this model translation, it is clear that the number of states in the translated DTMC equals to the number of actions (arcs) in MAMC. The number of arcs in the translated DTMC is bounded by the number of actions in MAMC multiplied by the largest degree of outgoing arcs of a certain state.

– **asCTML to CTML:**

- Given a set of atomic state+action formulas  $\mathcal{AF}$ , for each  $f \in \mathcal{AF}$  on  $M$ , we have  $f' \in \mathcal{AF}'$  on  $D$ , with  $f(s, a) = f'(sa)$ , for all  $(sa) \in \mathcal{S}'$ .

Note that since the two sets  $\mathcal{AF}$  and  $\mathcal{AF}'$  are exactly the same except one is defined on  $M$  with the parameter  $(s, a)$  being a pair of state and action and the other is defined on  $D$  with parameter  $(sa)$  being a state, for the following discussion, we use notations such as  $f^M$  and  $f^D$ , respectively, to refer to the same formulas on different model structures.

- Any asCTML formula  $\phi$  on  $M$ , denoted by  $\phi^M$ , is translated to the same formula  $\phi$  on  $D$ , denoted by  $\phi^D$ , such that for each atomic state+action formula  $f^M$  in  $\phi^M$ , it is replaced by the corresponding  $f^D$  in  $\phi^D$ .

**Theorem 1 (asCTML reduction to CTML).** *Let  $\phi^M$  be an asCTML formula on MAMC structure  $M$ , and  $D$  be the translated DTMC structure, then  $\phi^M(s, a)$  evaluate the same as  $\phi^D(sa)$  on  $D$ , where  $\phi^D$  is the translation of  $\phi^M$ .*

*Proof.* According to the translation method, there is a one to one mapping between state+action pair  $(s, a)$  on  $M$  and state  $(sa)$  on  $D$ , and each atomic state+action formula  $f$  on  $M$  is mapped to an atomic state formula  $f$  on  $D$ . As such,  $f^M = f^D$  holds trivially. By structural induction, assume that  $\phi^M = \phi^D$ ,  $\phi_1^M = \phi_1^D$ , and  $\phi_2^M = \phi_2^D$ . Then,

- $(\phi_1 \times \phi_2)^M = (\phi_1 \times \phi_2)^D$ ,  $(\phi_1 + \phi_2)^M = (\phi_1 + \phi_2)^D$ ,  $(\phi_1 \bowtie \phi_2)^M = (\phi_1 \bowtie \phi_2)^D$ , and  $(1 - \phi)^M = (1 - \phi)^D$  hold trivially, by the assumption.

Note that for succinctness, when the context is clear (e.g., when the parameter is given as a state such as  $(sa)$ , then it is a CTML formula; if the parameter is given as a pair of state and action such as  $(s, a)$ , then it is an asCTML formula), we drop the superscript  $D$  and  $M$ , respectively.

Also, for the following proof of  $M\psi$ , we utilize the fact that there exists a one to one mapping between each state, action pair  $(s, a)$  on MAMC and the translated state  $(sa)$  on DTMC. Then, each element  $(s_i, a_i)$  in a given MAMC path  $\pi^M = (s, a), (s_1, a_1), \dots$  matches the state  $(s_i a_i)$  in the corresponding path  $\pi^D = sa, s_1 a_1, \dots$  of the translated DTMC.

- $(MX\phi)^M = (MX\phi)^D$ . By asCTML semantics on  $MX\phi$ , for all paths:  $\pi = (s, a), (s_1, a_1), \dots$ , we have:

$$(MX\phi)^M(s, a) = \sum_{(s_1, a_1) \in \{\gamma(s, a)\} \times ACT} \phi(s_1, a_1) \delta(s_1, a_1)$$

By the assumption, and translations given  $s_1 = \gamma(s, a)$ ,

$$= \sum_{(s_1 a_1) \in \mathcal{S}' } \phi(s_1 a_1) \cdot \mathbf{P}[sa, s_1 a_1]$$

$$= (MX\phi)^D(sa).$$

- $(M\phi_1 U_{\odot}^{\leq t} \phi_2)^M = (M\phi_1 U_{\odot}^{\leq t} \phi_2)^D$ . By asCTML semantics on  $M\phi_1 U_{\odot}^{\leq t} \phi_2$ , for all paths of  $\pi = (s, a), (s_1, a_1), \dots$ , with  $(s, a) = (s_0, a_0)$ , if  $\exists j : 0 \leq j \leq t$ , s.t.  $\phi_2(s_j, a_j) > 0$  and for all  $0 \leq i \leq j$ ,  $\phi_2(s_i, a_i) = 0$ , we have:

$$\begin{aligned} & (M\phi_1 U_{\odot}^{\leq t} \phi_2)^M(s, a) \\ &= \sum_{j=0}^t \sum_{\substack{((s_0, a_0), \dots, (s_j, a_j)) \in (\mathcal{S} \times \mathcal{ACT})^j, \\ s_{i+1} = \gamma(s_i, a_i)}} \left( \bigcirc_{i=0}^{j-1} \phi_1(s_i, a_i) \right) \cdot \phi_2(s_j, a_j) \cdot \prod_{i=1}^j \delta(s_i, a_i) \end{aligned}$$

By the assumption, and translations given  $s_i = \gamma(s_{i-1}, a_{i-1})$ ,

$$= \sum_{j=0}^t \sum_{(s_0 a_0, \dots, s_j a_j) \in \mathcal{S}'^j} \left( \bigcirc_{i=0}^{j-1} \phi_1(s_i a_i) \right) \cdot \phi_2(s_j a_j) \cdot \prod_{i=1}^j \mathbf{P}[s_{i-1} a_{i-1}, s_i a_i]$$

$$= (M\phi_1 U_{\odot}^{\leq t} \phi_2)^D(sa).$$

- $(M\phi_1 V_{\odot}^{\leq t} \phi_2)^M = (M\phi_1 V_{\odot}^{\leq t} \phi_2)^D$ . According to the asCTML and CTML semantics of  $M\phi_1 V_{\odot}^{\leq t} \phi_2$ , there are two cases for this formula. One case is that if  $\exists j : 0 \leq j \leq t$ , s.t.  $\phi_2(s_j, a_j) > 0$  and for all  $0 \leq i \leq j$ ,  $\phi_2(s_i, a_i) = 0$ . In this case, we have just established that  $(M\phi_1 U_{\odot}^{\leq t} \phi_2)^M = (M\phi_1 U_{\odot}^{\leq t} \phi_2)^D$ . The second case is when  $\forall i : i \leq t, \phi_2(s_i, a_i) = 0$ . In this case, we have:

$$\begin{aligned} & (M\phi_1 V_{\odot}^{\leq t} \phi_2)^M(s, a) \\ &= \sum_{\substack{((s_0, a_0), \dots, (s_t, a_t)) \in (\mathcal{S} \times \mathcal{ACT})^t \\ s_{i+1} = \gamma(s_i, a_i)}} \left( \bigcirc_{i=0}^t \phi_1(s_i, a_i) \right) \cdot \prod_{i=1}^t \delta(s_i, a_i) \end{aligned}$$

By the assumption, and translations given  $s_i = \gamma(s_{i-1}, a_{i-1})$ ,

$$= \sum_{(s_0 a_0, \dots, s_t a_t) \in \mathcal{S}'^t} \left( \bigcirc_{i=0}^t \phi_1(s_i a_i) \right) \cdot \prod_{i=1}^t \mathbf{P}[s_{i-1} a_{i-1}, s_i a_i]$$

$$= (M\phi_1 V_{\odot}^{\leq t} \phi_2)^D(sa).$$

- Finally, according to the CTML definition in [1] for the final value of  $\phi$  on  $D$ , denoted by  $D_\phi$  and asCTML Definition 12 for the final value of  $\phi$  on  $M$ , denoted by  $M_\phi$ , we have:

$$\begin{aligned} M_\phi &= \sum_{s \in \mathcal{S}} \left( \sum_{a \in \mathcal{ACT}} \phi(s, a) \cdot \delta(s, a) \right) \cdot \pi_0(s) \\ &= \sum_{(sa) \in \mathcal{S}'} \phi(sa) \cdot \pi'_0(sa) \quad \text{by the translation algorithm} \\ &= D_\phi \end{aligned}$$

We now compare the expressiveness between asCTML and CTML. First of all, there are asCTML queries that are not expressible by CTML; see Sect. 5 for

example queries 1, 2, and 5. On the other hand, although CTML cannot be directly expressed by asCTML, any CTML formula can be simulated by a corresponding asCTML formula. Algorithm 1 gives details about the simulation. The overall idea about the simulation is that, for each case of CTML formula, we set the value of atomic formulas such that  $f(s, a) = f(s)$ , then compute the corresponding asCTML formula, then sum over the actions of the computed asCTML formula. The correctness of this algorithm is proven. Due to the limited space, the proof is omitted here.

We are ready to discuss the expressiveness between asCTML and the established probabilistic model checker (PRISM) [12, 13]. In general, asCTML and PRISM are incomparable. First of all, PRISM is a comprehensive tool that incorporates PCTL, CSL, PLTL, and PCTL\*, which means there are a lot of queries that can be expressed by PRISM cannot be expressed by asCTML. For example, PLTL formulas that have LTL formula of the form  $(a_1Ub_1) \wedge \dots \wedge (a_nUb_n)$  (where  $a_i, b_i, 1 \leq i \leq n$  are atomic propositions [14]) are not expressible by asCTML or CTML, because those are proven to be NP-hard [15], whereas the overall computational complexity of both asCTML and CTML are linear in the size of the operators in a formula, and polynomial in the size of a model. In addition, PRISM has mechanisms to associate a real-valued rewards to actions and/or states on DTMC models. It has  $R_{=?}$  and  $P_{=?}$  operators that denote rewards and probability, respectively, and can express formulas such as  $R_{=?}[\cdot]$  or  $P_{=?}[\cdot]$ , where  $[\cdot]$  denotes a classic temporal logic path formula (see examples in [16]). Nonetheless, the core formalisms in PRISM still defines a satisfaction relation; in another words, they cannot express formulas such as  $R_{=?}[f_1U[R_{=?}f_2Uf_3]]$ , or  $R_{=?}[f_1U[P_{=?}f_2Uf_3]]$  (where  $f_1, f_2, f_3$  are atomic formulas associated with actions and/or states), because unlike asCTML or CTML they cannot nest real values naturally in general (see the most recent PRISM work announced here: <http://www.prismmodelchecker.org/>).

## 5 The University Graduation Application

We have a prototype software tool implemented in Java, for the evaluation of asCTML and CTML queries. The heart of the software tool is the CTML engine that implements all the CTML algorithms discussed in [1]. For asCTML queries, we add an additional translator that translates from MAMC to DTMC and asCTML to CTML as discussed in Sect. 4. The software has been tested on several applications, including large dining philosopher models with millions of states and edges. Due to the limited space, for this work, we select to present a small model structure to illustrate capabilities of asCTML and CTML. With this small structure, we tend to show users more clearly the quantification of action driven paths, how action dependent queries can be expressed by asCTML but not CTML, how action independent queries can be expressed by CTML, and their syntactic similarities. More importantly, through this small application example, we tend to deliver the overall specification idea of asCTML with respect to CTML.

---

**Algorithm 1.** *simulate*( $\phi'$ ,  $M$ )

---

**Input:**  $\phi'$ ,  $M$ **Output:** vector  $\pi$  such that  $\pi[s] = \phi'(s)$ **switch**  $\phi'$  **do**  **case**  $f'$     for  $\forall s \in \mathcal{S}$  do       $\pi[s] = f'(s)$ ;     $f = \text{setOperand}(\pi, M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} f(s, a)\delta(s, a)$ ;    return  $\pi$ ;  **case**  $\phi'_1 \odot \phi'_2$      $\phi_1 = \text{setOperand}(\text{simulate}(\phi'_1, M), M)$ ;     $\phi_2 = \text{setOperand}(\text{simulate}(\phi'_2, M), M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} (\phi_1 \odot \phi_2)(s, a)\delta(s, a)$ ;    return  $\pi$ ;  **case**  $\phi'_1 \bowtie \phi'_2$      $\phi_1 = \text{setOperand}(\text{simulate}(\phi'_1, M), M)$ ;     $\phi_2 = \text{setOperand}(\text{simulate}(\phi'_2, M), M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} (\phi_1 \bowtie \phi_2)(s, a)\delta(s, a)$ ;    return  $\pi$ ;  **case**  $1 - \phi'$      $\phi = \text{setOperand}(\text{simulate}(\phi', M), M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} (1 - \phi)(s, a)\delta(s, a)$ ;    return  $\pi$ ;  **case**  $MX\phi'$      $\phi = \text{setOperand}(\text{simulate}(\phi', M), M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} (MX\phi)(s, a)\delta(s, a)$ ;    return  $\pi$ ;  **case**  $M\phi'_1 U_{\odot}^{\leq t} \phi'_2$      $\phi_1 = \text{setOperand}(\text{simulate}(\phi'_1, M), M)$ ;     $\phi_2 = \text{setOperand}(\text{simulate}(\phi'_2, M), M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} (M\phi_1 U_{\odot}^{\leq t} \phi_2)(s, a)\delta(s, a)$ ;    return  $\pi$ ;  **case**  $M\phi'_1 V_{\odot}^{\leq t} \phi'_2$      $\phi_1 = \text{setOperand}(\text{simulate}(\phi'_1, M), M)$ ;     $\phi_2 = \text{setOperand}(\text{simulate}(\phi'_2, M), M)$ ;    for  $\forall s \in \mathcal{S}$  do       $\pi[s] = \sum_{a \in \mathcal{ACT}} (M\phi_1 V_{\odot}^{\leq t} \phi_2)(s, a)\delta(s, a)$ ;    return  $\pi$ ;

---

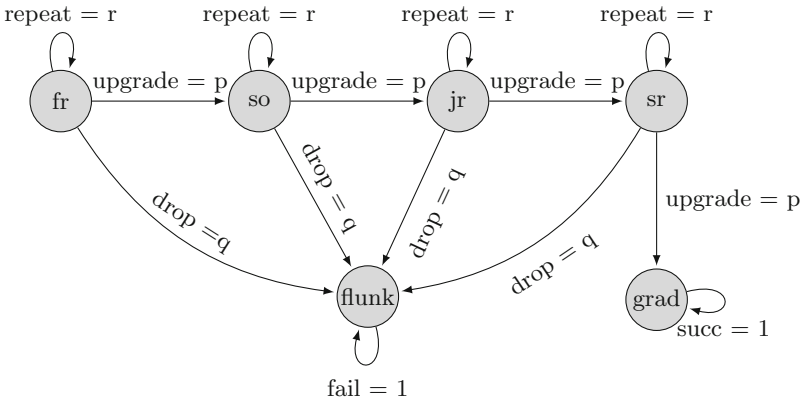


**Algorithm 2.** *setOperand*( $\pi, M$ )

**Input:** vector  $\pi, M$   
**Output:** asCTML atomic formula  $f$  such that  $f(s, a) = \pi[s]$   
 for  $\forall s \in \mathcal{S}$  do  
     for  $\forall a \in \mathcal{ACT}$  do  
          $f(s, a) = \pi[s]$ ;  
 return  $f$ ;

**5.1 Specification Examples**

The *University Graduation* example as shown in Fig. 2 is from [17]. Each year, at a fictitious four year undergraduate university, a student has probability  $p$  to move up to a higher grade, has probability  $r$  to repeat for the same grade, and probability  $q$  to drop the school. The following presents interesting queries for this model.



**Fig. 2.** University graduation example

1. “What’s the average number of years of repeating at the freshman level?”  
 First of all, the query requires to count the **repeat** action at the freshman year only. So we define an atomic formula *repeats-fr* such that it has value 1 on  $(fr, repeat)$ , and 0 on all others. Also, for this model, there are only two destinations which are either finish successfully indicated by **grad** or **flunk**. So we define another atomic formula *finish* such that it has value 1 on  $(grad, succ)$  and  $(flunk, fail)$ , and 0 otherwise. The query can then be expressed as

$$M \text{ repeats-fr } U_+ \text{ finish}$$

In general, this type of query is not expressible by CTML, especially if we change the value of  $repeats\text{-}fr(fr, repeat)$  to some other positive value other than 1. In this particular case, however, we can use the CTML expression  $M\text{ }fr\text{ }U_+(1 - fr)$  to get the average number of years spent as a freshman, and then we subtract one to get the number of repeats.

2. “What’s the average number of repeating at all levels of grades given that students graduate successfully?” Unlike the previous query, this query asks for the quantification of the repeating behavior at all levels of grades, so we define an atomic formula  $repeats$  such that  $repeats(\{fr, so, jr, sr\}, repeat) = 1$  and 0 for all others. Also, we need an atomic formula  $graduate(grad, succ) = 1$  and 0 otherwise. Then the query can be expressed as

$$M\text{ }repeats\text{ }U_+\text{ }graduate$$

By the definition of the conditional expectation equation [18]:

$$E(B | A) = \sum_{\forall b} b \cdot \Pr(B = b | A) = \frac{\sum_{\forall b} b \cdot \Pr(B = b \cap A)}{\Pr(A)} \tag{4}$$

the result of the expression must be divided by the quantity of  $M\text{ }one\text{ }U_\times\text{ }graduate$ . The query is not expressible by CTML in general.

3. “What’s the probability to quit the school after sophomore year?” Let  $quit\text{-}after\text{-}soph$  be an atomic formula such that  $quit\text{-}after\text{-}soph(\{so, jr, sr\}, drop) = 1$ , and 0 for all others. Then the query can be expressed as

$$M\text{ }one\text{ }U_\times\text{ }quit\text{-}after\text{-}soph$$

This query is expressible by CTML as

$$M\text{ }one\text{ }U_\times(M\text{ }so\text{ }U_\times\text{ }flunk)$$

and by PLTL as

$$P_{?}=F(so\text{ }U\text{ }flunk)$$

In this case, asCTML requires one path operator  $U_\times$ , whereas both CTML and PLTL requires two path operators.

4. “What’s the probability of graduation in 6 years?” Let  $graduate$  be an atomic formula with  $graduate(grad, succ) = 1$  and 0 otherwise. Then the query can be expressed as

$$M\text{ }one\text{ }U_\times^{\leq 6}\text{ }graduate$$

This query is expressible by CTML, with the same syntax as for asCTML, except that with CTML the atomic formulas  $graduate$  and  $one$  quantify the corresponding states only, rather than the (state, action) pairs.

5. “Given that a student reached the junior state and never repeated a year before, what’s the probability that the student will graduate in 3 years?” This query requires to only quantify the *upgrade* action at the *fr* and *so* states as the conditional statement imposes. Let  $no-repeat-soph(\{fr, so\}, upgrade) = 1$ , and 0 otherwise. Let  $graduate(grad, succ) = 1$  and 0 otherwise. Let  $junior(jr, \{repeat, upgrade, drop\}) = 1$ , and 0 otherwise. Then this query can be expressed as

$$M \text{ no-repeat-soph } U_{\times} (junior \times M \text{ one } U_{\times}^{\leq 3} graduate)$$

and the result must be divided by  $M \text{ no-repeat-soph } U_{\times} junior$ , according to the conditional probability definition [18]. Since the query depends on actions, it is not expressible by CTML.

Note that in case there are multiple actions between the same pair of states, the idea of the specification would be the same.

## 5.2 Experimental Results

We now discuss experimental results for the queries. First, we come up with a MAMC model structure for the University Graduation example, and a set of atomic state+action formulas for the queries. we (arbitrary) set the probabilities  $p, r, q$  to be 0.1, 0.8, and 0.1, respectively. The MAMC model and the atomic state+action formulas are then translated into DTMC model and CTML atomic state formulas, which are then plugged into CTML software for evaluation of each query. Since the specifications are syntactically the same for asCTML and CTML, they can either be included in MAMC model and carried over to the DTMC model, or be put in the translated DTMC model directly. For this small model, we omit the discussion of testing environment and CPU time. Table 1 presents the numerical results for each query discussed above. All outputs are verified by the numerical solutions computed by hand.

**Table 1.** Numerical results of asCTML queries for Fig. 2

#	Query	Initial state	Numerical result
1	$M \text{ repeats-fr } U_{+} \text{ finish}$	fr	0.11111
2	$\frac{M \text{ repeats } U_{+} \text{ graduate}}{M \text{ one } U_{\times} \text{ graduate}}$	fr	0.44443
3	$M \text{ one } U_{\times} \text{ quit-after-soph}$	fr	0.26459
4	$M \text{ one } U_{\times}^{\leq 6} \text{ graduate}$	fr	0.57344
5	$\frac{M \text{ no-repeat-soph } U_{\times} (junior \cdot M \text{ one } U_{\times}^{\leq 3} \text{ graduate})}{M \text{ no-repeat-soph } U_{\times} \text{ junior}}$	fr	0.64000

## 6 Related Work

There is a body of past work on stochastic model checking formalisms. The most relevant work such as PRISM and CTML has already been discussed in Sect. 4. Other rewards augmented stochastic formalisms include PRCTL [19] and CSRL [20] (which is similar to PRCTL, but operates on continuous-time Markov chains). They are more related to asCTML’s prior work, CTML, since all those are the extensions of PCTL with rewards functions, but they don’t have action formulas. For action and state based stochastic formalisms, CSL<sup>TA</sup> [10] and asCSL [5] (which is an extension of CSL [21] and aCSL [22] that work with continuous-time Markov chains) are closely related to asCTML in a sense that both allow multiple actions between a pair of states and can quantify paths featured by a combination of states and actions. CSL<sup>TA</sup> is more expressive than asCSL for reasoning state and action properties over probabilistic systems. Unlike asCTML, however, they don’t support real-valued rewards. Another work we would like to mention is *performance trees* [23], as it can describe results of various types (real-valued or otherwise, including distributions). Unlike CTML or asCTML, however, performance trees is a more general framework that relies on the existing performance evaluation algorithms as well as some of the existing model checking algorithms (such as PCTL, CSL, etc.) for the expression of both logic and real-valued measures.

## 7 Conclusions and Future Work

We have introduced a formal query language, asCTML, for describing quantitative performance measures. The combination of nesting and real-valued state/action formulas, particularly the nesting of action formulas, unique to asCTML, extend the expressive power of the familiar temporal operators. Through proper translation, asCTML can simulate any CTML formula. Our asCTML algorithm has complexity similar to that of PCTL, but better than PLTL. We are currently working on to apply asCTML for continuous time Markov chains or semi-Markov processes. For unbounded until and unbounded weak until queries, this is fairly straightforward: since asCTML can handle real-valued state/action formulas, we can instead analyze the embedded DTMC using asCTML, and scale the state/action formulas by the expected time spent in each state or action. The time bounded versions of these formulas are not so straightforward to handle, and will likely require significant changes.

## References

1. Miner, A.S., Jing, Y.: A formal language toward the unification of model checking and performance evaluation. In: Al-Begain, K., Fiems, D., Knottenbelt, W.J. (eds.) *ASMTA 2010*. LNCS, vol. 6148, pp. 130–144. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-13568-2\\_10](https://doi.org/10.1007/978-3-642-13568-2_10)
2. Ciardo, G., Trivedi, K.S.: A decomposition approach for stochastic reward net models. *Perform. Eval.* **18**, 37–59 (1993)
3. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects Comput.* **6**(5), 512–535 (1994)
4. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. *J. ACM* **42**(4), 857–907 (1995)
5. Baier, C., Cloth, L., Haverkort, B.R., Kuntz, M., Siegle, M.: Model checking Markov chains with actions and state labels. *IEEE Trans. Softw. Eng.* **33**, 209–224 (2007)
6. Dijkstra, E.W.: Hierarchical ordering of sequential processes. *Acta Informatica* **1**, 115–138 (1971)
7. Obal, W.D., Sanders, W.H.: State-space support for path-based reward variables. *Perform. Eval.* **35**(3–4), 233–251 (1999)
8. Stewart, W.J.: *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, Princeton (1994)
9. Cohn, D.L.: *Measure Theory*. Birkhäuser, Boston (1980)
10. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL<sup>TA</sup>. *IEEE Trans. Softw. Eng.* **35**(2), 224–240 (2009)
11. Clark, G., Gilmore, S., Hillston, J.: Specifying performance measures for PEPA. In: Katoen, J.-P. (ed.) *ARTS 1999*. LNCS, vol. 1601, pp. 211–227. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48778-6\\_13](https://doi.org/10.1007/3-540-48778-6_13)
12. Kwiatkowska, M.: Quantitative verification: models, techniques and tools. In: *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, New York, NY, USA, pp. 449–458. ACM (2007)
13. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *CAV 2011*. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
14. Pnueli, A.: The temporal semantics of concurrent programs. *Theor. Comput. Sci.* **13**, 45–60 (1981)
15. Schnoebelen, P.: The complexity of temporal logic model checking. *Adv. Modal Log.* **4**, 393–436 (2002)
16. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic verification of Herman’s self-stabilisation algorithm. *Formal Aspects Comput.* **24**(4–6), 661–670 (2012)
17. Kemeny, J.G., Snell, J.L.: *Finite Markov Chains*. D. Van Nostrand Company Inc., Princeton (1960)
18. Pitman, J.: *Probability*. Springer, New York (1993). <https://doi.org/10.1007/978-1-4612-4374-8>
19. Andova, S., Hermanns, H., Katoen, J.P.: Discrete-time rewards model-checked. In: Larsen, K.G., Niebert, P. (eds.) *Formal Modeling and Analysis of Timed Systems*, pp. 88–104. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-40903-8\\_8](https://doi.org/10.1007/978-3-540-40903-8_8)

20. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: On the logical characterisation of performability properties. In: Montanari, U., Rolim, J.D.P., Welzl, E. (eds.) ICALP 2000. LNCS, vol. 1853, pp. 780–792. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-45022-X\\_65](https://doi.org/10.1007/3-540-45022-X_65)
21. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Softw. Eng.* **29**(6), 524–541 (2003)
22. Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., Siegle, M.: Towards model checking stochastic process algebra. In: Grieskamp, W., Santen, T., Stoddart, B. (eds.) IFM 2000. LNCS, vol. 1945, pp. 420–439. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-40911-4\\_24](https://doi.org/10.1007/3-540-40911-4_24)
23. Suto, T., Bradley, J.T., Knottenbelt, W.J.: Performance trees: expressiveness and quantitative semantics. In: Proceedings of the 4th International Conference on Quantitative Evaluation of Systems, Washington DC, USA, pp. 41–50. IEEE Computer Society (2007)



# Model Checking for Safe Navigation Among Humans

Sebastian Junges<sup>1</sup>, Nils Jansen<sup>2</sup>(✉), Joost-Pieter Katoen<sup>1</sup>, Ufuk Topcu<sup>3</sup>,  
Ruohan Zhang<sup>3</sup>, and Mary Hayhoe<sup>3</sup>

<sup>1</sup> RWTH Aachen University, Aachen, Germany

<sup>2</sup> Radboud University, Nijmegen, The Netherlands  
nilsjansen123@gmail.com

<sup>3</sup> The University of Texas at Austin, Austin, TX, USA

**Abstract.** We investigate the use of probabilistic model checking to synthesise optimal strategies for autonomous systems that operate among uncontrollable agents such as humans. To formally assess such uncontrollable behaviour, we use models obtained from reinforcement learning. These behaviour models are, e.g., based on data collected in experiments in which humans execute dynamic tasks in a virtual environment. We first describe a method to translate such behaviour models into Markov decision processes (MDPs). The composition of these MDPs with models for (controllable) autonomous systems gives rise to stochastic games (SGs). MDPs and SGs are amenable to probabilistic model checking which enables the synthesis of strategies that provably adhere to formal specifications such as probabilistic temporal logic constraints. Experiments with a prototype provide (1) systematic insights on the credibility and the characteristics of behavioural models and (2) methods for automated synthesis of strategies satisfying guarantees on their required characteristics in the presence of humans.

## 1 Introduction

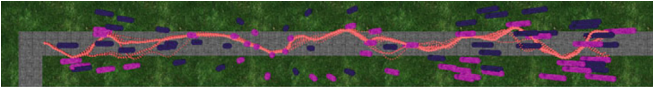
The control of autonomous agents like robots often necessitates to account for other, potentially uncontrollable, agents such as humans. Examples of such *partially controlled multi-agent systems* [1] include self-driving cars [2], autonomous trading agents in the stock market [3], and service robots in presence of humans [4]. Physical interaction with humans makes them safety-critical.

Dependability requirements in safety-critical autonomous systems call for guarantees beyond confidence intervals with soft bounds, such as statistically obtained by simulations or experiments. For example, a self-driving car has to safely operate among pedestrians and human-operated cars. Repeatedly running physical experiments is costly and may put humans at risk.

---

Supported by the CDZ project CAP (GZ 1023), the DFG RTG 2236 “UnRAVeL”, and the NSF grants 1652113, 1651089, and 1550212.

To address the need for certified bounds on safety requirements in autonomous systems, we synthesise strategies to move a controllable agent towards a goal while limiting the probability of crashing into other dynamic (and uncontrollable) agents, such as humans. We exemplify the synthesis by means of a case study in which a human moves along a board walk, see Fig. 1. We extract a Markov Decision Process (MDP) [5] from an existing *behavioural model*, for instance of humans, obtained by *reinforcement learning* (RL) [6]. We refer to such a model for uncontrollable agents as an RL model. Roughly, states in the MDP are a Cartesian product of the state of the environment and the location of the agent. The stochastic transitions are based on observed experimental data, as computed by RL. Due to lack of observations in corner cases and abstraction of complicated scenarios, accurate stochastic descriptions of the behaviour are not always available, that is, the agent behaviour is *under-specified*. Instead of guessing, we explicitly model the agents’ behaviour as non-deterministic in such situations. MDPs are a suitable modelling formalism, as they allow the co-existence of the stochastic and non-deterministic elements in the RL model.



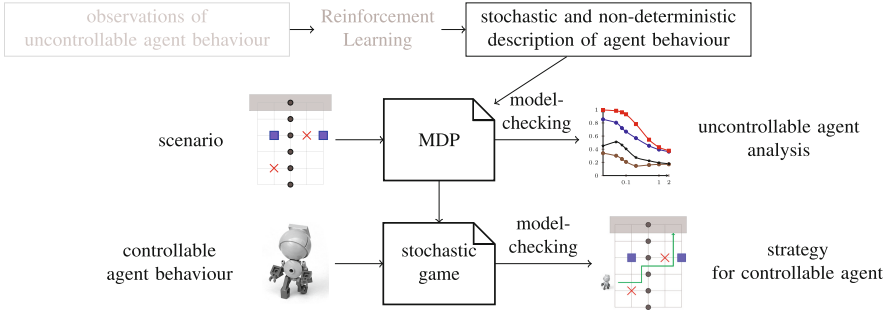
**Fig. 1.** A multi-task sidewalk environment in simulation. The objective is split into three modular tasks (i) approach targets (purple), (ii) avoid obstacles (blue), and (iii) follow walkway (grey). (Color figure online)

Two key tasks are to (1) *analyse the learned behaviour of uncontrollable agents* and to (2) *synthesise a strategy for the controllable agents* that provably adheres to safety and/or performance specifications. We propose to use *probabilistic model checking* (PMC) for these tasks. PMC is a formal verification technique tailored to systems exhibiting randomness and uncertainty [7], and fully automated tools—e.g., PRISM [8] or Storm [9]—are readily available to provide an off-the-shelf solution. Figure 2 shows an overview of our approach.

First, given an RL behaviour model and an arbitrary fixed concrete scenario, we generate an MDP *automatically*. To address task (1), we phrase measures that assess the behaviour of the uncontrollable agent in PCTL, a probabilistic temporal logic [10]. For instance, we consider the minimum/maximum probability—depending on how the non-determinism is resolved—to not bump into obstacles, or the expected number of steps to reach a region in the environment. As a side product, PMC computes strategies which resolve non-determinism that obtain extremal probabilities. These strategies yield insights where the underspecification is critical. *PMC thus provides quantifiable feedback about the behaviour of the uncontrollable agent, as modelled by the MDP for an RL model.*

To address task (2), we add a controllable agent which moves in the presence of the aforementioned uncontrollable agents. We describe the actions of the controllable agent, together with probabilities for possible uncertain outcomes





**Fig. 2.** The conceptual framework of our approach for the partially controlled multi-agent problem.

of these actions, as an MDP. Composing such an MDP with the MDP for the RL models, yields a two-player *stochastic game* (SG) [11]. One player takes the actions of the controllable agent, and one player resolves the (non-deterministic) uncertainty in the behaviour of the uncontrollable agents.

PMC [12] *automatically synthesises strategies for the controllable agent in the SG that, if possible, provably adhere to temporal logic constraints, for any resolution of the non-determinism in the uncontrollable agents*. Such constraints may enforce the controllable agent to reach, with high probability, a certain area without a collision, or impose a bound on the expected number of steps. PMC supports such automatic strategy synthesis. In addition, PMC yields Pareto-curves to analyse the trade-off between several, potentially conflicting, measures.

A practical challenge is that the resulting MDPs and SGs are necessarily large in the context of real-world applications. Moreover, the data obtained from observations may induce models that do not exhibit the common features (e.g., structural symmetries) that usually help suppress the size of the MDPs and SGs. They also contain a large number of different probabilities. Consequently, off-the-shelf abstraction techniques employed in PMC tools [13] barely reduce the state space of such models. We alleviate this challenge by several optimisations in the encoding of the models, and adequately invoking available tools.

*Contributions.* This paper presents a framework to employ PMC for partially controlled multi-agent systems where a behaviour model for the uncontrollable agents is present and obtained by RL. The main technical contributions are: (1) a general recipe to cast behaviour models from RL into MDPs, (2) the use of PMC to obtain dependable quantitative insights of such models, (3) a technique to combine MDP models from controllable and uncontrollable agents into an SG, and (4) the synthesis of strategies by applying PMC on SGs. Either the resulting strategies are guaranteed to be safe and/or efficient, or the approach reports on the impossibility to synthesise such strategies.

*Case Study.* We apply the proposed approach to an existing case study in human-robot interaction, a visiomotor multi-task model from [14, 15]. Here, RL

describes quantitative aspects of human behaviour such as the likelihood to take an action in a certain situation. To account for the limitation in human cognition, global tasks are decomposed into modular sub-tasks. This modelling of multi-task behaviour is called *modular (inverse) RL* [15–18]. Behaviour in such a setting may be observed in mixed-reality environments and cast into a general behaviour model. The goal is to synthesise a *strategy* for a robot navigating safely, i. e., without interfering with the human, through the same environment.

*Related Work.* PMC has been applied to autonomous robot systems, see e.g. [19–23]. Trade-off analysis has been advocated in [22, 24]. These applications all consider MDP models of robots. This paper instead considers MDP model checking of behaviour models obtained from RL. SGs have been used to model multi-agent learning problems [25, 26]. Finally, [27] uses statistical model checking to evaluate self-assembly strategies of autonomous systems. We reflect on the usage of statistical model checking for our setting in Sect. 6.

## 2 Preliminaries

**Definition 1 (Probabilistic models).** A stochastic game (SG) is a tuple  $\mathfrak{M} = (S, s_I, Act, \mathcal{P})$  with a finite set  $S$  of states with  $S = S_\circ \uplus S_\square$  where players  $\circ$  and  $\square$  control the corresponding states, an initial state  $s_I \in S$ , a finite set  $Act$  of actions, and a transition function  $\mathcal{P}: S \times Act \times S \rightarrow [0, 1]$  with  $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') \in \{0, 1\} \quad \forall s \in S, \alpha \in Act$ . For each state  $s \in S$ , the set of enabled actions is  $Act(s) = \{\alpha \in Act \mid \exists s' \in S. \mathcal{P}(s, \alpha, s') \neq 0\}$ .

- $\mathfrak{M}$  is a Markov decision process (MDP) if  $S_\circ = \emptyset$  or  $S_\square = \emptyset$ .
- MDP  $\mathfrak{M}$  is a Markov chain (MC) if  $|Act(s)| = 1$  for all  $s \in S$ .

Players *nondeterministically* choose an action at their state; successor states are determined probabilistically according to transition probabilities. MDPs and MCs are one- and zero-player SGs, respectively. Probabilistic models are commonly specified in a guarded command language developed for PRISM [8].

Nondeterministic choices of actions in SGs are resolved by so-called *strategies*; resolving all nondeterminism yields *induced MCs*. In general, strategies (for each player) are functions  $\sigma: S^* \rightarrow Distr(Act)$ , such that finite paths of probabilistic models are mapped to distributions over actions. For many properties, however, simpler strategies suffice [28].

## 3 Partially Controlled Multi-agent Setting

The considered setting consists of an environment with a set of (controllable or uncontrollable) agents. Here, we define an environment to be a 2D-grid with *features* of different types from a set of types  $Tp$  which influence the agents' behaviour. One example for such a feature type is a (potentially moving) obstacle.

**Definition 2 (Environment).** An environment  $Env = (Loc, Feat)$  consists of a finite set of locations  $Loc \in [0, Grid_x] \times [0, Grid_y]$  with  $Grid_x, Grid_y \in \mathbb{N}$ , and a set of features  $Feat \subseteq Tp \times Loc$ . A feature  $f = (tp_f, \ell_f) \in Feat$  consists of a type and a (feature-)location.

An agent  $h$  is represented by its position  $\text{pos}_h = (\ell_h, \alpha_h)$ , with location  $\ell_h$  and an orientation  $\alpha_h \in \mathbb{R}$ , e.g., north (0) or south ( $\pi$ ). The set of all positions is denoted  $\text{Pos}$ . Two agents crash, if they have the same location. Agent  $h$  starts in position  $\text{init}_h$ , and has an associated set  $M_h$  of movements<sup>1</sup>. A movement  $m \in M_h$  deterministically updates the position of agent  $h$  as described by a partial function  $\text{post}_m: \text{Pos} \rightarrow \text{Pos}$ ;  $m$  is defined in  $\text{pos}_h$  only if  $\text{post}_m(\text{pos}_h)$  describes a *valid* position, i.e., a position on the grid. While moving, an agent can perceive *knowledge* (as described by a set of atomic propositions) about the features or other agents. The *state*  $s$  of agent  $h$  consists of its position and the perceived knowledge,  $s = (\text{pos}_h, \text{know}_h)$ . The set of all states is denoted  $S_h$ . Let  $\text{eff}_m: S_h \rightarrow S_h$  computes the successor state of taking movement  $m$  in a state by updating the position and the perceived knowledge, exemplified later.

**Reinforcement Learning.** Intuitively, RL lets a controllable agent *explore* its environment by sequential decision-making [6]. The objective is to (approximatively) optimize the expected reward for the agent in the underlying MDP or SG [25]. During the exploration, a strategy may be *unsafe* in the sense that it harms the agent or the environment. This shortcoming restricts the application of RL mainly to application areas where safety is not a concern and has triggered the particular direction of *safe RL* [29]. Most approaches rely on “tweaking” the reward functions such that a learning agent behaves in a desired, potentially safe, manner. As rewards are often specialized for particular environments, such reward engineering runs the risk of triggering negative side effects or hiding potential bugs [30]. Therefore, we separate the concerns by first using RL to obtain a general behaviour model which may then be analysed with respect to both expected rewards and safety in concrete scenarios.

**Behaviour of Uncontrollable Agents.** We assume an *estimation of the behaviour* of uncontrollable agents, derived from an existing RL model in two steps.

The first step determines a set  $V(s)$  of *movement-value-vectors* for each state  $s$ . Each movement-value-vector  $\mathbf{q} \in V(s)$  contains a *score* for every movement, i. e.,  $\mathbf{q}: M_h \rightarrow \mathbb{R}_\infty$ . Roughly, scores correspond to likelihoods for the movements, obtained from the RL model. Sometimes  $|V(s)| > 1$ , as either:

- A specific situation may have never been accounted for. Because of this *lack of data*, the learned model is short of dependable estimations of specific movements scores. Or,

<sup>1</sup> We use the term movements to clarify the distinction from *actions* in MDPs.

- the learned behaviour model is based on *relative information* regarding the features of the setting. For instance, the distance to an obstacle may influence the score of a movement. As the closest obstacle is not necessarily unique, several different scores for a state are possible.

The second step defines the stochastic behaviour of the agent by translating any movement-value-vector  $\mathbf{q} \neq -\infty$  to a *distribution over movements*. For the translation, we use a *softmax*-function  $\mathbb{R}^{|M_h|} \rightarrow [0, 1]^{|M_h|}$ , which attributes most, but not all probability to the maximum [6]. The distribution is defined as:

$$\text{softmax}_\tau(\mathbf{q})_i = e^{q_i/\tau} / \sum_{i \leq |q|} e^{q_i/\tau} \quad (\text{with } e^{-\infty} := 0, \text{ and } \tau > 0).$$

Invalid movements ( $q_i$ ) have zero probability due to the numerator. The parameter  $\tau$  is called the *temperature*. For a temperature close to zero, the function yields a (hard) maximum; for a high temperature it yields an almost uniform distribution over the valid movements.

**MDP for One Uncontrollable Agent.** The behaviour can be captured in an MDP by using the position and the perceived knowledge as a state. The obtained distributions over movements yield the transition function. In particular, each movement-value-vector  $\mathbf{q}$  corresponds to an action in the MDP. The successor states of a an action corresponding to  $\mathbf{q}$  is determined by applying the associated movements, and the distribution is given by  $\text{softmax}_\tau(\mathbf{q})$  as described above. Due to the nondeterminism, the behaviour MDP can be seen as a partial strategy for uncontrollable agents provided by the behaviour model.

**MDP for Multiple (Uncontrollable) Agents.** We assume that all agents move in alternating fashion. This abstraction yields a pessimistic over-approximation of the non-determinism over synchronous movements, and allows a straightforward mapping from movement-vectors to actions. The over-approximation compared to synchronous movements diminishes to less non-determinism the model exhibits. A state of the joint MDP is given by the Cartesian product, together with an entry which agent is moving. The actions and their effects are determined by the transition function of the MDP of the current agent, as in, e.g., [31].

**SG for Controllable and Uncontrollable Agents.** We assume descriptions of the controllable agent are available, either via RL or directly modelled. The joint SG is similar to the joint MDP. States where a controllable agent is moving correspond to Player  $\circ$ , the uncontrollable non-determinism to  $\square$ , respectively. The controllable agents move first, to prevent them from taking the upcoming move by  $\square$  into account. A strategy for the controllable agent adhering to a given property is robust against both stochastic and nondeterministic actions of the uncontrollable agents. If the MDP for the uncontrollable agents does not contain any non-determinism, the joint SG collapses: all choices belong to the controllable player, therefore, the joint agent SG is an MDP.

**Analysis.** PMC supports a wide range of properties for MDPs and SGs to analyse [28]. We focus on determining strategies that induce maximal or minimal probabilities to safely reach target states and the expected cost until the target is reached. Richer specifications such as probabilistic temporal logic constraints or  $\omega$ -regular properties are commonly reduced to such reachability objectives [28].

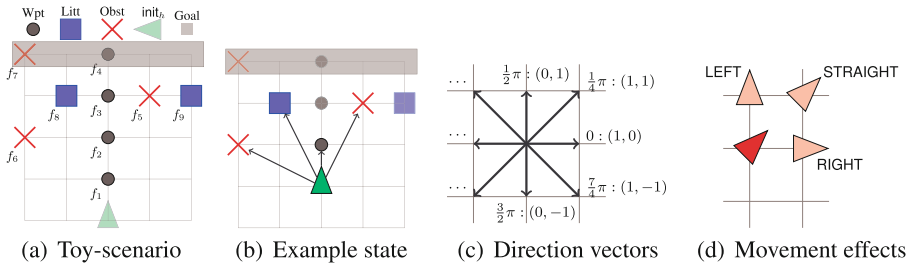
To analyse the MDP, we compute the minimal and maximal values for a property, depending on how non-determinism is resolved. A large difference between these values indicates that the uncertainty (underspecification) in the data is significantly affecting results. For the SG, we consider the best-case behaviour of the controllable agent. Often, the best-case behaviour for one property induces sub-optimal behaviour with respect to another property. To illustrate these trade-offs, we compute Pareto-curves via multi-objective PMC [32]. The model, however, can be checked for other properties without adaptations.

### 4 Case Study

The specific example we study is a visiomotor multi-task model from [14,15]. We describe the concrete adaption of this model towards a behaviour MDP and the inclusion of a controllable agent in line with the concepts of Sect. 3.

**Scenario.** We consider a scenario involving an uncontrollable human agent going along a sidewalk encountering *obstacles* and *litter*, see Fig. 1. The human is given three modular objectives: while FOLLOW a sidewalk (represented as a set of *waypoints*) to get to the other end, she should AVOID walking into obstacles and aim to COLLECT litter. Waypoints, obstacles and litter are the *features*. Features are initially present, but disappear when visited, run over, or collected. We mark regions of the environment as *target areas*. A toy scenario is given in Fig. 3(a). The scenario is abstracted to a discrete-state model to characterise human behaviour in the model. Recall that the behaviour is determined by movement-values, which are translated into distributions over movements.

We specify both the environment and the human behaviour. Throughout the section,  $\text{Env} = (\text{Loc}, \text{Feat})$  is an environment with an arbitrarily sized grid and a finite set of features. Features have a type in  $Tp = \{\text{Obst}, \text{Litt}, \text{Wpt}\}$ .



**Fig. 3.** Grid worlds, orientations, and human movement.

*Example 1.* Consider Fig. 3(a). The environment is  $\text{Env} = (\text{Loc}, \text{Feat})$  with  $\text{Loc} = \{(x, y) \mid x \in [0, 4] \ y \in [0, 4]\}$ , and

$$\begin{aligned} \text{Feat} = & \{f_i = (\text{Wpt}, (2, i)) \mid i \in \{1 \dots 4\}\} \\ & \cup \{f_5 = (\text{Obst}, (3, 3)), f_6 = (\text{Obst}, (0, 2)), f_7 = (\text{Obst}, (0, 4))\} \\ & \cup \{f_8 = (\text{Litt}, (1, 3)), f_9 = (\text{Litt}, (4, 3))\} \end{aligned}$$

The human's orientation has 8 possible values,  $\alpha_h \in \text{Orient} = \{i \cdot \frac{1}{4}\pi \mid i \in [0, 7]\}$ , and is associated with a direction  $\text{Dir}: \text{Orient} \rightarrow \{-1, 0, 1\}^2 \setminus \{(0, 0)\}$ , see Fig. 3(c). The possible movements are  $M_h = \{\text{LEFT}, \text{STRAIGHT}, \text{RIGHT}\}$ , we depict the associated position updates in Fig. 3(d). Formally, we have

$$\text{post}_m(\text{pos}_h) = (\ell_h + \text{Dir}(\alpha_h + \beta_m), \quad (\alpha_h + \beta_m) \bmod 2\pi)$$

for  $\text{pos}_h = (\ell_h, \alpha_h)$  and movement  $m$  with angle  $\beta_m$ , where  $\beta_m \in \{-\frac{1}{4}\pi, 0, \frac{1}{4}\pi\}$ . Thus, for some movement, we first update the position based on the current orientation, and then the orientation.

The *perceived knowledge*  $\text{know}_h$  concerns the presence of features, i. e., if they have not been collected or run over. Here, the set  $\text{know}_h = \text{PFeat} \subseteq 2^{\text{Feat}}$  contains the present features. Consequently, a state for a human is a tuple  $(\text{pos}_h, \text{PFeat})$ . Features that coincide with the updated human location disappear and are removed from the set of present features:

$$\text{fu}_m(s) = \text{PFeat} \setminus \{(\text{tp}, \text{post}_m(\text{pos}_h))\} \subseteq \text{Feat}.$$

Updating the position and the perceived knowledge yields the successor state.

**Definition 3 (Successor state for human).** *The successor state of movement  $m$  in state  $s = (\text{pos}_h, \text{PFeat})$  is  $\text{eff}_m(s) = (\text{post}_m(\text{pos}_h), \text{fu}_m(s))$ .*

**Human Behaviour.** We discuss how to obtain a distribution over different movements within a modular RL framework, where the weights are obtained via inverse RL. From [15], we obtain Q-tables and weights describing human behaviour, cf. Fig. 4(a). Figure 4(b) outlines how to obtain values for the possible movements for each objective. Figure 4(c) describes how to combine values for different objectives and their translation into distributions. The estimation depends on the relative location with respect to some features. Details of the steps are below.

*Distance and Angle.* The distance  $d_h(f)$  between the human at location  $\ell_h$  and a feature  $f$  at location  $\ell$  is the Euclidean norm of their locations.  $\theta_h(f)$  denotes the signed angle between the human orientation and  $\ell_h - \ell$ .

*Relevant Features.* For each objective, only a subset of the features are relevant for the human behaviour in a state. First, for each objective  $o$ , we have exactly one *corresponding feature-type*  $F(o) \in \text{Tp}$ ; Waypoints, obstacles and litter correspond to FOLLOW, AVOID and COLLECT, respectively. The set

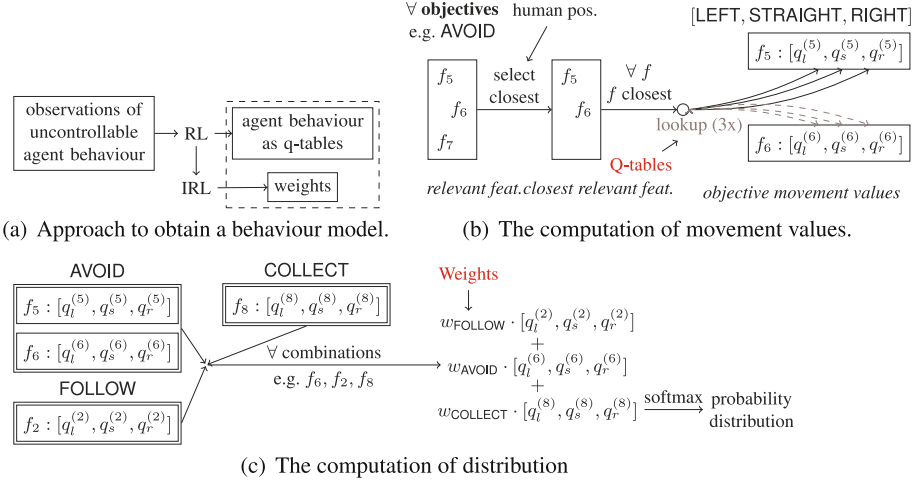


Fig. 4. How to construct sets of distributions.

$\text{RelFeat}_o(s) = \{(F(o), \ell) \in \text{PFeat}\}$  contains the relevant features for objective  $o$  in state  $s = (\text{pos}_h, \text{PFeat})$ . Second, we adopt the assumption from [15] stating that for each objective  $o$ , only the closest feature of type  $f(o)$  is relevant for the behaviour with respect to  $o$ .

*Remark.* This rather strong assumption reduces the number of hidden parameters a learning method has to estimate. The closest relevant features  $\text{Close}(s, o)$  for objective  $o$  in state  $s$  are:

$$\text{Close}(s, o) = \{f \in \text{RelFeat}_o(s) \mid \forall f' \in \text{RelFeat}_o(s). d_h(f) \leq d_h(f')\}.$$

*Example 2.* Recall state  $s$  in Fig. 3(b). The closest relevant features are, e.g.,  $\text{Close}(s, \text{AVOID}) = \{f_5, f_6\}$ , and  $\text{Close}(s, \text{FOLLOW}) = \{f_2\}$ .

As we observe from the example, the human behaviour model is under-specified: It is not clear which feature is relevant, i.e., the set  $\text{Close}(s, o)$  is not necessarily a singleton, as several nearby objects may be located at similar distances. Any feature in the set may be the one that is being considered.

*Movement-Values.* For each movement  $m$  and each objective  $o$ , we assume we are given a Q-table  $Q_o^m : \mathbb{R} \times \mathbb{R}_+ \rightarrow \mathbb{R}$  which maps the angle  $\theta_h(f)$  and distance  $d_h(f)$  between a human and relevant feature  $f$  to an objective-movement-value, constructed via [15]. We map the closest relevant features into a set of objective-movement-vectors  $V^o(s)$  by a lookup in the Q-table, and store the feature:

$$V^o(s) = \{(f, \mathbf{q} = [q_l, q_s, q_r]) \mid f \in \text{Close}(s, o), q_l = Q_o^{\text{LEFT}}(\theta_h(f), d_h(f)), q_s = Q_o^{\text{STRAIGHT}}(\theta_h(f), d_h(f)), q_r = Q_o^{\text{RIGHT}}(\theta_h(f), d_h(f))\}.$$

Vector entries collect movement-values for a fixed feature.

*Combining Movement Values.* A probability distribution over objectives is given as a *objective-weight vector*  $w = [w_{\text{AVOID}}, w_{\text{COLLECT}}, w_{\text{FOLLOW}}]$ , e.g. obtained by modular IRL as in [15]. The objective-movement-vectors are translated into a set  $V(s)$  of movement vectors  $\mathbf{q}$  annotated with sets of relevant features  $F$  by calculating a weighted sum over all combinations and annotating them with the union of the features. For invalid movements, we set  $-\infty$  as movement-value. If no movement is possible, we remove the vector from the movement values.

*Distribution.* The last step (cf. Fig. 4(c)) is to transfer movement values into a distribution over the movements.

#### 4.1 Operational Model

**Behaviour MDP.** We automatically generate an MDP for the human behaviour. The human starts in its initial position and all features are present. Recall that the nondeterminism is caused by underspecification of the model, resulting in multiple movement vectors, and that each vector is reflected by an action.

**Definition 4.** *The MDP  $\mathcal{M} = (S, s_I, Act, \mathcal{P})$  reflecting the human behaviour starting in  $init_h$  on an environment  $Env = (Loc, Feat)$  with temperature  $\tau$  is  $S = \{(pos_h, P) \in (Loc \times Orient) \times 2^{Feat}\}$  with  $s_I = (init_h, Feat)$ ,  $Act = Feat \times Feat \times Feat$  and*

$$\mathcal{P}(s, a, s') = \begin{cases} \text{softmax}_{\tau}(\mathbf{q})_i & \text{if } (a, \mathbf{q}) \in V(s), s' = \text{eff}_{(M_h)_i}(s) \text{ for an } i \\ 0 & \text{otherwise.} \end{cases}$$

**Joint Human-Robot SG.** As controllable agent, we consider a robot either turning  $90^\circ$  (left or right) in place, or moving forward. The human-robot SG is the parallel composition of the MDP for the human and the MDP for the robot. We consider two cases for the human behaviour in the presence of a robot. First, we assume the behaviour is not influenced by the robot, which yields a strategy for the robot not counting on the human to actively evade the robot. Second, we model the robot as an obstacle-feature: The human avoids the robot, and the robot strategy takes the human ‘aversion’ of the robot into account. For the latter case, we assume that the human treats the robot as a static obstacle.

**Joint Human-Robot MDP.** The SG collapses to an MDP if there is no non-determinism in the description of the uncontrollable agents. One way of *eliminating* underspecification is to create *unique* closest relevant features by selecting the one with the smallest absolute angle, and if tied, left-of-the-human over right. This abstraction allows to treat larger benchmarks on a less precise model.

*Example 3.* Eliminated underspecification yields for state  $s$  from Fig. 3(b),  $\text{Close}(s, \text{AVOID}) = \{f_5\}$ , as  $|\theta_h(f_5)| \approx 26^\circ < |\theta_h(f_6)| \approx 63^\circ$ .



## 4.2 Experimental Setup

Our prototype realises the framework as in Fig. 2 using PRISM-Games [12] for evaluation of the human and synthesising strategies, and storm for trade-off analysis of multiple specifications. Results are obtained on a HP BL685C G7, using 48 cores, 2.0 GHz each, and 192 GB of RAM.

*State and Transition Encoding.* We encode the models using the PRISM-language, where *parallel composition* of modules yields concrete MDP or SG models<sup>2</sup>. We define MDPs describing human and robot behaviour in form of such modules; parallel composition yields the SG describing joint behaviour. Roughly, the two MDP modules are encoded as follows. A global flag indicates whether the human or the robot moves next; preconditions in both modules ensure a turn-based movement. The module for the human consists of 3 integers to represent the human position and flags  $b_f$  indicating the presence of feature  $f$ . The transition relation for each human position is listed explicitly. Although the number of reachable states is exponential in the number of features, the encoding is cubic as the behaviour depends on the nearest features only. The robot module defines any given model over the same environment as for the human. The SG state space is the product of the two modules, the size of the encoding equals their sum.

*Optimizations.* The encoding in the PRISM language enables using a variety of tools, but the *size of the encoding* grows rapidly with the size of the environment. In fact, the model cannot be represented as succinctly (up to 100K lines) as typical examples (up to 1K lines); therefore *parsing and model building* take significant time. We employed some performance improvements, among them: (1) Only the Q-table for obstacle avoidance shows equal values for the far-away columns—indicating that human behaviour does not consider far-away obstacles. It is not necessary to distinguish which obstacle is nearest if all induce the same score. (2) As every location in a concrete scenario occurs in a large number of commands, a symbolic substitution of similar positions reduces the overhead of specifying locations repeatedly. These improvements reduce the parsing time by 40%. It is important to choose the right tool configuration, in particular the engine and preprocessing.

## 5 Results

To give an indication on the sizes of the MDPs and SGs we handle, consider Fig. 5(e–f) presenting the grid size, the number of features (obstacles/litter/waypoints), the number of states, and global actions of the models using  $\tau = 0.075$ .

*Evaluating the Behaviour Model.* For the behaviour MDP, we compute *minimum and maximum* probabilities for reaching a target area. These probabilities depend on how the nondeterminism concerning underspecification is resolved.

<sup>2</sup> Available at [https://github.com/moves-rwth/human\\_factor\\_models](https://github.com/moves-rwth/human_factor_models).

The gap between these probabilities indicates the actual relevance of underspecification. Moreover, we analyse the *number of steps* a human needs to reach the target area to get an indication if the behaviour is realistic. We discuss results for an MDP induced by a  $20 \times 20$  grid with 2 pieces of litter, 2 obstacles, and 7 waypoints (2/2/7), see Fig. 5(a). The human is only told to follow the waypoints.

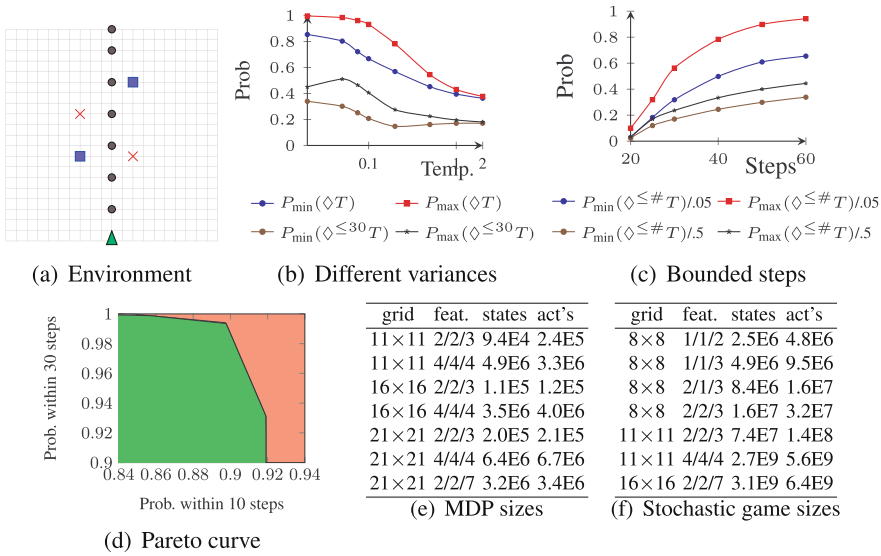


Fig. 5. MDP analysis using PMC.

*Relevance of Underspecification.* Figure 5(b) plots the min/max probability to reach the target area against the temperature (controlling the variability in the softmax function). It shows that, with low variability, the fragment of executions indeed reaching the other side without leaving the grid is high. With higher variability (where features are mostly ignored), this number quickly drops.

*Performance of the Human.* Figure 5(c) indicates a similar behaviour for step-bounded reachability for different number of steps (x-axis) and temperatures 0.05 and 0.5. Most executions take more than 30 steps to reach the goal, indicating that, based on the given data, humans *very unlikely walk in straight lines*. This phenomenon occurs due to the lack of a *notion of progress* in visiting waypoints—it does not penalize walking in circles, as only positive reward is earned on visiting waypoints. The gap due to underspecification is significant as long as the variability is not too high. With low variability, most executions reach the goal within 60 steps. Detailed analyses considering the obtained strategies show where in the model underspecification has the largest effect.

*Robot Strategy Synthesis.* For robot strategies, we use PMC to maximise measures regarding *safety*, where the robot never occupies the same cell as the

human, and *performance*, where a target area is reached within a certain number of steps. However, these objectives may conflict with each other: If the goal has to be reached in a few steps, there is not much time for precautions, and the probability of safety is affected negatively. In contrast, safer strategies reduce the performance. To obtain insights, we perform a *trade-off analysis*.

*Analysing Joint Human-Robot MDPs.* On a  $8 \times 8$  grid with  $2/2/3$  features, we considered two objectives: To reach the other side within 10 steps without crashing and to reach the other side within 30 steps without crashing. The first gives only few opportunities to evade the human, while the latter gives plenty opportunities. The two objectives thus may conflict with each other. Figure 5(d) has been automatically generated with PMC. It shows in green combinations of objectives which can be achieved by some strategy. We see that optimising the first condition yields a strategy with 92% chance to satisfy the former condition and a 93% chance to satisfy the latter, while another strategy yields a 99% chance to satisfy the latter and a 90% chance to satisfy the former condition.

*Scalability of Analysing SGs.* Using our aforementioned optimisations, we are able to analyse all SGs in Fig. 5(f). The symbolic engine of PRISM-Games finishes (model) building within 30 s for all these models, but (model) checking to find a good strategy takes significantly more time: between 270 and 4800 s for the  $8 \times 8$  grids, respectively, and 70 h for the smaller  $11 \times 11$  grid. Using an explicit engine, checking is faster, but building suffers from the large size of the encoding: For the  $8 \times 8$  grids, checking requires between 300 and 2100 s, but building takes between 40 min and 20 h, respectively. The smaller  $11 \times 11$  grid takes up to 200 h of building. Once a model is built, multiple properties can be checked on this model efficiently. Currently, only the explicit engine allows to actually extract an optimal strategy. Therefore, we only obtain the strategies for the  $8 \times 8$  grids and the smaller  $11 \times 11$  grid. For MDPs, the performance is superior on the sparse engine, which does model-building via the symbolic engine and checking via an explicit representation.

## 6 Discussion

*The Model.* Based on the PMC results, we obtained five lessons about the weighted Q-table model. **(1)** According to the provided model, humans most likely walk in wavy lines. Following a line and giving a penalty for any diverging move (as in [15]) would provide a notion of progress. **(2)** As the Q-tables do not *take into account the border of the environment*, they are not avoiding a deadlock (or unspecified behaviour): The probability for leaving the grid is substantial in many cases. **(3)** For the discretised model we use, there is a potentially significant difference in behaviour based on how the underspecification is resolved; *any analysis on the learned model has to take this underspecification into account*. **(4)** Modelling variability over human behaviour by a single softmax and *using a memoryless model* are rough estimates. Therefore it is quite likely that the human model contains behaviours which contradict statements about

the observed behaviour from e.g. [14]. **(5)** Finally, the Q-tables contain some unexpected *outliers* which in some configurations lead to unexpected behaviour.

*The Method.* The description for the human behaviour including variability can be translated into a formal model; allowing a variety of properties to be easily analysed—in particular, it allows for analysing underspecifications and ill-defined data. The generalization to a robot planning scenario (and to a SG) is straightforward, and enables to compute plans fulfilling specific properties. Probabilistic verification of this model raised *five challenges*: **(1)** The *large number of different probabilities* in learned data blows up the encoding and the symbolic state-space representation, and prevents successful application reduction techniques such as bisimulation. It would be interesting to *regularise the model* on the learning side. **(2)** The softmax function serves the only purpose of introducing variability; its differentiability is not utilised here. *Sensitivity analysis over Q-table values would be of interest.* **(3)** Despite the lack of typical symmetries in the scenarios, its encoding is significantly smaller than enumerating all states, as (here) the behaviour only depends on the nearest obstacles. Even for three features of every kind, a reduction of a factor over 50 is possible. **(4)** While the approach yields promising results for MDPs, the SG suffers from a state space explosion. The regularity of the turn-based game is exploited by a *symbolic engine*, but suffers from the large symbolic description, see (1). **(5)** Finally, strategies are computed for the full state space, whereas *we are often only interested in a fragment of the full state space*; for many states we can take any action.

Using simulation-based alternatives to PMC such as statistical model checking [33] bears the problem that it is unclear how to resolve two levels of non-determinism. Moreover, the model we investigate in our case study contains probabilities which are rather small. Therefore, rare-event simulation has to be used, and will increase the simulation effort.

## 7 Conclusion

We have laid the framework for translating a (learned) behaviour model to a formal setting, and used it to compute control strategies for agents moving among uncontrollable agents handling complex tasks. We discussed a concrete model as well as several (open) challenges. For future work, we propose to investigate abstraction techniques such as [34] and restrict exploration of the model via [35].

## References

1. Brafman, R.I., Tennenholtz, M.: On partially controlled multi-agent systems. *J. Artif. Intell. Res.* **4**, 477–507 (1996)
2. Dresner, K., Stone, P.: A multiagent approach to autonomous intersection management. *J. Artif. Intell. Res.* **31**, 591–656 (2008)
3. Wellman, M.P., Wurman, P.R., O'Malley, K., Bangera, R., Reeves, D., Walsh, W.E.: Designing the market game for a trading agent competition. *IEEE Internet Comput.* **5**(2), 43–51 (2001)

4. Khandelwal, P., et al.: Bwibots: a platform for bridging the gap between AI and human-robot interaction research. *Int. J. Robot. Res.* **36**, 635–659 (2017)
5. Puterman, M.L.: *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, New York (1994)
6. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge (1998)
7. Kwiatkowska, M.Z.: Model checking for probability and time: from theory to practice. In: *LICS*, p. 351. IEEE Computer Society (2003)
8. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *CAV 2011*. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
9. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A STORM is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčák, V. (eds.) *CAV 2017*. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_31](https://doi.org/10.1007/978-3-319-63390-9_31)
10. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects Comput.* **6**(5), 512–535 (1994)
11. Condon, A.: The complexity of stochastic games. *Inf. Comput.* **96**(2), 203–224 (1992)
12. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-Games 2.0: a tool for multi-objective strategy synthesis for stochastic games. In: Chechik, M., Raskin, J.-F. (eds.) *TACAS 2016*. LNCS, vol. 9636, pp. 560–566. Springer, Heidelberg (2016). [https://doi.org/10.1007/978-3-662-49674-9\\_35](https://doi.org/10.1007/978-3-662-49674-9_35)
13. Dean, T.L., Givan, R.: Model minimization in Markov decision processes. In: *AAAI/IAAI*, pp. 106–111. AAAI Press/The MIT Press (1997)
14. Tong, M.H., Zohar, O., Hayhoe, M.M.: Control of gaze while walking: task structure, reward, and uncertainty. *J. Vis.* **17**(1), 28 (2017)
15. Rothkopf, C.A., Ballard, D.H.: Modular inverse reinforcement learning for visuomotor behaviour. *Biol. Cybern.* **107**(4), 477–490 (2013)
16. Sprague, N., Ballard, D.: Multiple-goal reinforcement learning with modular sarsa (0). *IJCA I*, 1445–1447 (2003)
17. Ballard, D.H., Kit, D., Rothkopf, C.A., Sullivan, B.: A hierarchical modular architecture for embodied cognition. *Multisens. Res.* **26**(1–2), 177–204 (2013)
18. Leong, Y.C., Radulescu, A., Daniel, R., DeWoskin, V., Niv, Y.: Dynamic interaction between reinforcement learning and attention in multidimensional environments. *Neuron* **93**(2), 451–463 (2017)
19. Konur, S., Dixon, C., Fisher, M.: Analysing robot swarm behaviour via probabilistic model checking. *Robot. Auton. Syst.* **60**(2), 199–213 (2012)
20. Johnson, B., Kress-Gazit, H.: Analyzing and revising synthesized controllers for robots with sensing and actuation errors. *Int. J. Robot. Res.* **34**(6), 816–832 (2015)
21. Giaquinta, R., Hoffmann, R., Ireland, M., Miller, A., Norman, G.: Strategy synthesis for autonomous agents using PRISM. In: Dutle, A., Muñoz, C., Narkawicz, A. (eds.) *NFM 2018*. LNCS, vol. 10811, pp. 220–236. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-77935-5\\_16](https://doi.org/10.1007/978-3-319-77935-5_16)
22. Chen, T., Kwiatkowska, M., Simaitis, A., Wiltsche, C.: Synthesis for multi-objective stochastic games: an application to autonomous urban driving. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P.R. (eds.) *QEST 2013*. LNCS, vol. 8054, pp. 322–337. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40196-1\\_28](https://doi.org/10.1007/978-3-642-40196-1_28)

23. Feng, L., Wiltsche, C., Humphrey, L., Topcu, U.: Synthesis of human-in-the-loop control protocols for autonomous systems. *IEEE Trans. Autom. Sci. Eng.* **13**(2), 450–462 (2016)
24. Lacerda, B., Parker, D., Hawes, N.: Optimal policy generation for partially satisfiable co-safe LTL specifications. In: *IJCAI*, pp. 1587–1593. AAAI Press (2015)
25. Littman, M.L.: Markov games as a framework for multi-agent reinforcement learning. *ICML* **157**, 157–163 (1994)
26. Bowling, M., Veloso, M.: Multiagent learning using a variable learning rate. *Artif. Intell.* **136**(2), 215–250 (2002)
27. Bruni, R., Corradini, A., Gadducci, F., Lluch Lafuente, A., Vandin, A.: Modelling and analyzing adaptive self-assembly strategies with maude. In: Durán, F. (ed.) *WRLA 2012. LNCS*, vol. 7571, pp. 118–138. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-34005-5\\_7](https://doi.org/10.1007/978-3-642-34005-5_7)
28. Katoen, J.P.: The probabilistic model checking landscape. In: *LICS*, pp. 31–45. ACM (2016)
29. Garcia, J., Fernández, F.: A comprehensive survey on safe reinforcement learning. *J. Mach. Learn. Res.* **16**(1), 1437–1480 (2015)
30. Sculley, D., Phillips, T., Ebner, D., Chaudhary, V., Young, M.: *Machine learning: the high-interest credit card of technical debt* (2014)
31. Winterer, L., et al.: Motion planning under partial observability using game-based abstraction. In: *CDC*, pp. 2201–2208. IEEE (2017)
32. Etesami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. *Log. Methods Comput. Sci.* **4**(4), 1–21 (2008)
33. Agha, G., Palmskog, K.: A survey of statistical model checking. *ACM Trans. Model. Comput. Simul.* **28**(1), 6:1–6:39 (2018)
34. Wachter, B., Zhang, L., Hermanns, H.: Probabilistic model checking modulo theories. In: *QEST*, pp. 129–140. IEEE CS (2007)
35. Brázdil, T., et al.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J.-F. (eds.) *ATVA 2014. LNCS*, vol. 8837, pp. 98–114. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-11936-6\\_8](https://doi.org/10.1007/978-3-319-11936-6_8)



# Automated Verification of Concurrent Stochastic Games

Marta Kwiatkowska<sup>1</sup>, Gethin Norman<sup>2(✉)</sup>, David Parker<sup>3</sup>, and Gabriel Santos<sup>1</sup>

<sup>1</sup> Department of Computing Science, University of Oxford, Oxford, UK

<sup>2</sup> School of Computing Science, University of Glasgow, Glasgow, UK  
gethin.norman@glasgow.ac.uk

<sup>3</sup> School of Computer Science, University of Birmingham, Birmingham, UK

**Abstract.** We present automatic verification techniques for concurrent stochastic multi-player games (CSGs) with rewards. To express properties of such models, we adapt the temporal logic rPATL (probabilistic alternating-time temporal logic with rewards), originally introduced for the simpler model of turn-based games, which enables quantitative reasoning about the ability of coalitions of players to achieve goals related to the probability of an event or reward measures. We propose and implement a modelling approach and model checking algorithms for property verification and strategy synthesis of CSGs, as an extension of PRISM-games. We evaluate the performance, scalability and applicability of our techniques on case studies from domains such as security, networks and finance, showing that we can analyse systems with probabilistic, cooperative and competitive behaviour between concurrent components, including many scenarios that cannot be analysed with turn-based models.

## 1 Introduction

Stochastic multi-player games are a versatile modelling framework for systems that exhibit cooperative and competitive behaviour in the presence of adversarial or uncertain environments. They can be viewed as a collection of players (agents) with strategies for determining their actions based on the execution so far. These models combine nondeterminism, representing the adversarial, cooperative and competitive choices, stochasticity, modelling uncertainty due to noise, failures or randomness, and concurrency, representing simultaneous execution of interacting agents. Examples of such systems appear in many domains, from robotics and autonomous transport, to security and computer networks.

*Formal verification* for stochastic games provide a means of producing quantitative guarantees on the correctness of a system (e.g. “the control software can always safely stop the vehicle with probability at least 0.99, regardless of the actions of other road users”), where the required behavioural properties are specified precisely in quantitative extensions of temporal logic. The closely related problem of *strategy synthesis* constructs an optimal strategy for a player, or coalition of players, that guarantees a property is satisfied.

Automatic verification and strategy synthesis for models exhibiting nondeterminism and stochasticity are well established and implemented, e.g., in tools such as PRISM [14] and STORM [11]. Recently, these techniques have also been formulated and implemented in PRISM-games [16], an extension of PRISM, for *turn-based* stochastic multi-player games (TSGs), which can be viewed as Markov decision processes whose states are partitioned among a set of players, with exactly one player taking control of each state. Properties are specified in the logic rPATL (probabilistic alternating-time temporal logic with rewards) [9], a quantitative extension of the game logic ATL [4]. This allows us to specify that a coalition of players achieve a high-level objective, regarding the probability of an event’s occurrence or the expectation of a cumulative reward measure, irrespective of the strategies of the other players.

*Concurrent* stochastic multi-player games (CSGs) generalise TSGs by permitting players to choose their actions concurrently in each state of the model. This can provide a more realistic model of interactive agents operating concurrently, and making action choices without already knowing the actions that are being taken by other agents. However, although algorithms for verification and strategy synthesis of CSGs have been known for some time (e.g., [2, 3, 6]), their implementation and application to real-world examples is lacking.

This paper develops the first approach for the modelling, verification and strategy synthesis of CSGs that is implemented in software and applied to a selection of in-depth case studies. We first adapt the logic rPATL to CSGs and provide a formal semantics. Then, we propose a model checking algorithm, building upon the existing techniques for TSGs and adapting to CSGs by integrating techniques for solving matrix games. Next, we develop an approach to modelling CSGs as an extension of the PRISM-games model checking tool and implement algorithms for construction, verification and strategy synthesis.

Finally, we investigate the performance, scalability and applicability of our implementation using a selection of real-life case studies. We demonstrate that our CSG modelling and verification techniques facilitate insightful analysis of quantitative aspects of systems taken from diverse set of application domains: finance, computer security, computer networks and communication systems. These illustrate examples of systems whose modelling and analysis requires stochasticity *and* competitive or adversarial behaviour between concurrent components or agents, as provided by CSGs; in several cases, we explicitly highlight the differences between our use of CSGs and existing models verified using TSGs.

**Related Work.** Various verification algorithms have been proposed for CSGs, e.g. [2, 3, 6], but without implementations, tool support or case studies. PRISM-games [16], upon which we build in this work, provides modelling and verification for a wide range of properties of stochastic multi-player games, including those in the logic rPATL, and multi-objective extensions of it, but focuses purely on the turn-based variant of the model (TSGs). GIST [8] allows the analysis of  $\omega$ -regular properties on probabilistic games, but again focuses on turn-based, not concurrent, games. GAVS+ [10] is a general-purpose tool for algorithmic game solving, supporting TSGs and (non-stochastic) concurrent games, but not CSGs.



Two further tools, PRALINE [5] and EAGLE [25], allow the computation of Nash equilibria [20] for the restricted class of (non-stochastic) concurrent games.

## 2 Preliminaries

We begin with some basic background from game theory, and then describe concurrent stochastic games. For any finite set  $X$ , we will write  $\text{Dist}(X)$  for the set of probability distributions over  $X$ . We first introduce *normal form games*, which are simple one-shot games where players make their choices concurrently.

**Definition 1 (Normal form game).** A (finite,  $n$ -person) normal form game (also known as strategic form) is a tuple  $\mathbf{N} = (N, A, u)$  where:

- $N = \{1, \dots, n\}$  is a finite set of players;
- $A = A_1 \times \dots \times A_n$  and  $A_i$  is a finite set of actions available to player  $i \in N$ ;
- $u = (u_1, \dots, u_n)$  and  $u_i : A \rightarrow \mathbb{R}$  is a utility function for player  $i \in N$ .

In a game  $\mathbf{N}$ , players select actions simultaneously, with player  $i \in N$  choosing from the action set  $A_i$ . If each player  $i$  selects action  $a_i$ , then player  $j$  receives the utility  $u_j(a_1, \dots, a_n)$ . A (mixed) strategy  $\sigma_i$  for player  $i$  is a distribution over its actions, i.e.  $\sigma_i \in \text{Dist}(A_i)$ . Let  $\Sigma_{\mathbf{N}}^i$  denote the set of strategies for player  $i$ . A *strategy profile* is a tuple of strategies for each player. Under a strategy profile  $\sigma = (\sigma_1, \dots, \sigma_n)$ , the expected utility of player  $j$  is defined as follows:

$$u_j(\sigma) \stackrel{\text{def}}{=} \sum_{(a_1, \dots, a_n) \in A} u_j(a_1, \dots, a_n) \cdot \left( \prod_{i=1}^n \sigma_i(a_i) \right).$$

A two-player normal form game  $\mathbf{N} = (\{1, 2\}, A, u)$  is *zero-sum* if for each action tuple  $\alpha \in A$  we have  $u_1(\alpha) + u_2(\alpha) = 0$ . Such a game is often called a *matrix game* as it can be represented by a matrix  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  where  $A_1 = \{a_1, \dots, a_l\}$ ,  $A_2 = \{b_1, \dots, b_m\}$  and  $z_{ij} = u_1(a_i, b_j) = -u_2(a_i, b_j)$ .

We require the following classical result concerning matrix games.

**Theorem 1 (Minimax theorem [21, 22]).** For any matrix game  $\mathbf{Z} \in \mathbb{R}^{l \times m}$ , there exists  $v^* \in \mathbb{R}$ , called the value of the game and denoted  $\text{val}(\mathbf{Z})$ , such that:

- there is a strategy  $\sigma_1^*$  for player 1, called an optimal strategy of player 1, such that under this strategy the player's expected utility is at least  $v^*$  regardless of the strategy of player 2, i.e.  $\inf_{\sigma_2 \in \Sigma_{\mathbf{N}}^2} u_1(\sigma_1^*, \sigma_2) \geq v^*$ ;
- there is a strategy  $\sigma_2^*$  for player 2, called an optimal strategy of player 2, such that under this strategy the player's expected utility is at least  $-v^*$  regardless of the strategy of player 1, i.e.  $\inf_{\sigma_1 \in \Sigma_{\mathbf{N}}^1} u_2(\sigma_1, \sigma_2^*) \geq -v^*$ .

The value of a matrix game  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  can be found by solving the following linear programming (LP) problem [21, 22]. Maximise  $v$  subject to the constraints:

$$\begin{aligned} v &\leq p_1 \cdot z_{1j} + \dots + p_l \cdot z_{lj} \quad \text{for } 1 \leq j \leq m \\ p_i &\geq 0 \quad \text{for } 1 \leq i \leq l \quad \text{and} \quad p_1 + \dots + p_l = 1 \end{aligned}$$

In addition, the solution for  $(p_1, \dots, p_l)$  yields an optimal strategy for player 1. The value of the game can also be calculated as the solution of the following dual LP problem. Minimise  $v$  subject to the constraints:

$$v \geq q_1 \cdot z_{i1} + \dots + q_m \cdot z_{im} \text{ for } 1 \leq i \leq l$$

$$q_j \geq 0 \text{ for } 1 \leq j \leq m \text{ and } q_1 + \dots + q_m = 1$$

and the solution  $(q_1, \dots, q_m)$  yields an optimal strategy for player 2.

We next introduce *concurrent stochastic games*, in which players repeatedly make choices simultaneously to determine the next state of the game.

**Definition 2 (Concurrent stochastic game).** A concurrent stochastic multi-player game (CSG) is a tuple  $G = (N, S, \bar{s}, A, \Delta, \delta, AP, L)$  where:

- $N = \{1, \dots, n\}$  is a finite set of players;
- $S$  is a finite set of states and  $\bar{s} \in S$  is an initial state;
- $A = (A_1 \cup \{\perp\}) \times \dots \times (A_n \cup \{\perp\})$  where  $A_i$  is a finite set of actions available to player  $i \in N$  and  $\perp$  is an idle action disjoint from the set  $\cup_{i=1}^n A_i$ ;
- $\Delta : S \rightarrow 2^{\cup_{i=1}^n A_i}$  is an action assignment function;
- $\delta : S \times A \rightarrow \text{Dist}(S)$  is a probabilistic transition function;
- $AP$  is a set of atomic propositions and  $L : S \rightarrow 2^{AP}$  is a labelling function.

In any state  $s$  of a CSG  $G$ , each player  $i \in N$  chooses an action from the set  $A_i(s)$  which equals  $\Delta(s) \cap A_i$  if this set is non-empty and equals  $\{\perp\}$  otherwise. If each player  $i$  selects action  $a_i$ , then the next state of the game is chosen according to the probability distribution  $\delta(s, (a_1, \dots, a_n))$ . TSGs are a restricted class of CSGs for which in any state  $s$  there is a unique player  $i$  such that  $A_i(s) \neq \{\perp\}$ .

A path  $\pi$  of a CSG  $G$  is a sequence  $\pi = s_0 \xrightarrow{\alpha_0} s_1 \xrightarrow{\alpha_1} \dots$  where  $s_i \in S$ ,  $\alpha_i = (a_1^i, \dots, a_n^i) \in A$ ,  $a_j^i \in A_j(s_i)$  for  $j \in N$  and  $\delta(s_i, \alpha_i)(s_{i+1}) > 0$  for all  $i \geq 0$ . We denote by  $\pi(i)$  the  $(i+1)$ th state of  $\pi$ ,  $\pi[i]$  the action associated with the  $(i+1)$ th transition and, if  $\pi$  is finite,  $last(\pi)$  the final state. The length of a path  $\pi$ , denoted  $|\pi|$ , is the number of transitions appearing in the path. Let  $FPaths_G$  and  $IPaths_G$  ( $FPaths_{G,s}$  and  $IPaths_{G,s}$ ) be the sets of finite and infinite paths (starting in state  $s$ ). A strategy  $\sigma_i$  for player  $i$  of  $G$  is a way of resolving the choices of player  $i$  based on the execution so far. Formally, a strategy for player  $i$  is a function  $\sigma_i : FPaths_G \rightarrow \text{Dist}(A_i)$  such that if  $\sigma_i(\pi)(a_i) > 0$ , then  $a_i \in A_i(last(\pi))$ . The set of all strategies of player  $i$  is denoted  $\Sigma_G^i$ . A strategy for player  $i$  is deterministic if it always selects actions with probability 1 and memoryless if it makes the same choice for paths that end in the same state.

A strategy profile for CSG  $G$  is a tuple  $\sigma = (\sigma_1, \dots, \sigma_n) \in \Sigma_G^1 \times \dots \times \Sigma_G^n$  yielding a strategy for each player of the game. We use  $FPaths_{G,s}^\sigma$  and  $IPaths_{G,s}^\sigma$  for the sets of finite and infinite paths corresponding to the choices made by the strategy profile  $\sigma$  when starting in state  $s$ . For a given strategy profile  $\sigma$  and starting state  $s$ , the behaviour of  $G$  is fully probabilistic and we can define a probability measure  $Prob_{G,s}^\sigma$  over the set of infinite paths  $IPaths_{G,s}^\sigma$  [13]. Given a random variable  $X : IPaths_G \rightarrow \mathbb{R}$ , the expected value of  $X$  with respect to profile  $\sigma$  when starting in state  $s$  is given by  $\mathbb{E}_{G,s}^\sigma(X) \stackrel{\text{def}}{=} \int_{\pi \in IPaths_{G,s}^\sigma} X(\pi) dProb_{G,s}^\sigma$ .

We augment CSGs with *reward structures* of the form  $r = (r_A, r_S)$  where  $r_A : S \times A \rightarrow \mathbb{R}$  is an action reward function (which maps each state and action tuple pair to a real value that is accumulated when the action tuple is selected in the state) and  $r_S : S \rightarrow \mathbb{R}$  is a state reward function (which maps each state to a real value that is accumulated when the state is reached). We allow both positive and negative rewards; however, we will later require certain restrictions to ensure the correctness of the presented model checking algorithm.

**Example 1.** Consider the CSG shown in Fig. 1(a) corresponding to two players repeatedly playing the rock-paper-scissors game. Transitions are labelled with action-pairs where  $A_1 = A_2 = \{r, p, s, t\}$ , with  $r$ ,  $p$  and  $s$  representing playing rock, paper and scissors, respectively, and  $t$  restarting the game. The CSG starts in state  $s_0$  and states  $s_1$ ,  $s_2$  and  $s_3$  are labelled with atomic propositions corresponding to when a player wins or there is a draw in a round of the rock-paper-scissors game. The matrix game representation of the rock-paper-scissors game is presented in Fig. 1(b). The optimal value for the matrix game is the solution to the following LP problem. Maximise  $v$  subject to the constraints:

$$v \leq p_2 - p_3, v \leq p_3 - p_1, v \leq p_1 - p_2, p_1 + p_2 + p_3 = 1, p_1, p_2, p_3 \geq 0$$

which yields the solution  $v^* = 0$  with corresponding optimal strategy  $\sigma_1^* = (1/3, 1/3, 1/3)$  for player 1 (the optimal strategy for player 2 is the same).

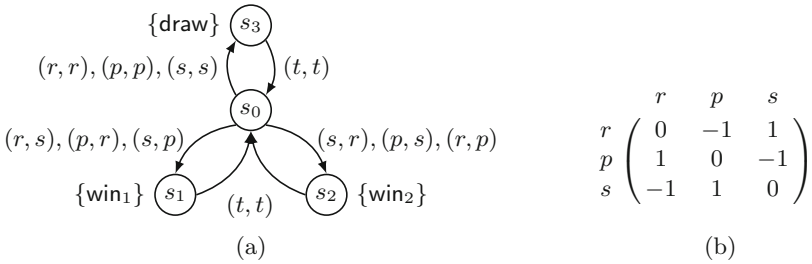


Fig. 1. (a) Rock-paper-scissors CSG. (b) Rock-paper-scissors matrix game.

### 3 rPATL for Concurrent Stochastic Games

In this section, we discuss the temporal logic rPATL, previously proposed for specifying properties of TSGs [9], and adapt it to CSGs.

**Definition 3 (rPATL syntax).** *The syntax of rPATL is given by the grammar:*

$$\begin{aligned} \phi &::= \mathbf{true} \mid \mathbf{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle P_{\sim q}[\psi] \mid \langle\langle C \rangle\rangle R_{\sim x}[\rho] \\ \psi &::= \mathbf{X}\phi \mid \phi \mathbf{U}^{\leq k} \phi \mid \phi \mathbf{U} \phi \\ \rho &::= \mathbf{I}^k \mid \mathbf{C}^{\leq k} \mid \mathbf{C} \mid \mathbf{F}^c \phi \end{aligned}$$

where  $\mathbf{a} \in AP$  is an atomic proposition,  $C \subseteq N$  is a coalition of players,  $\sim \in \{<, \leq, \geq, >\}$ ,  $q \in [0, 1]$ ,  $x \in \mathbb{R}$ ,  $r$  is a reward structure and  $k \in \mathbb{N}$ .

The rPATL syntax distinguishes between state ( $\phi$ ), path ( $\psi$ ) and reward ( $\rho$ ) formulae. State formulae are evaluated over states of a CSG, while path and reward formulae are both evaluated over paths. A state satisfies a formula  $\langle\langle C \rangle\rangle P_{\sim q}[\psi]$  if the *coalition* of players  $C$  can ensure the probability of the path formula  $\psi$  being satisfied is  $\sim q$ , and satisfies a formula  $\langle\langle C \rangle\rangle R_{\sim x}^r[\rho]$  if the players in  $C$  can ensure the expected value of the reward formula  $\rho$  for rewards structure  $r$  is  $\sim x$ . For path formulae, we allow next ( $\mathbf{X}\phi$ ), time-bounded until ( $\phi \mathbf{U}^{\leq k}\phi$ ) and unbounded until ( $\phi \mathbf{U}\phi$ ). For reward formulae, we allow instantaneous (state) reward at the  $k$ th step ( $\mathbf{I}^k$ ), reward accumulated over  $k$  steps ( $\mathbf{C}^{\leq k}$ ), total cumulative reward ( $\mathbf{C}$ ) and reward accumulated until a formula is satisfied ( $\mathbf{F}^c\phi$ ).

There are two differences from the rPATL syntax of [9]. First, we add several reward operators ( $\mathbf{I}^k$ ,  $\mathbf{C}^{\leq k}$  and  $\mathbf{C}$ ), adapted from the property specification language of PRISM [14]. These proved to be useful for the case studies we present later in Sect. 6. On the other hand, for simplicity, we restricted our attention to a single variant ( $c$ ) of the reward operator  $\mathbf{F}^*\phi$ : two other variants are included in [9], labelled 0 and  $\infty$ , which define the accumulated reward to be 0 or infinity, respectively, if no state satisfying  $\phi$  is reached along a path. We intend to add these variants to our CSG verification implementation at a later date.

To introduce the semantics of rPATL, for any CSG  $\mathbf{G}$  and coalition of players  $C \subseteq N$ , we require the following definition of a two-player *coalition game*. Without loss of generality, we assume the coalition is of the form  $C = \{1, \dots, n'\}$ .

**Definition 4 (Coalition game).** For CSG  $\mathbf{G} = (N, S, \bar{s}, A, \Delta, \delta, AP, L)$  and coalition  $C = \{1, \dots, n'\} \subseteq N$ , the coalition game induced by  $C$  of  $\mathbf{G}$  is the two-player game  $\mathbf{G}^C = (\{1, 2\}, S, \bar{s}, A^C, \Delta, \delta^C, AP, L)$  where:

- $A^C = A_1^C \times A_2^C$ ,  $A_1^C = (A_1 \cup \{\perp\}) \times \dots \times (A_{n'} \cup \{\perp\})$  and  $A_2^C = (A_{n'+1} \cup \{\perp\}) \times \dots \times (A_n \cup \{\perp\})$ ;
- for any  $s \in S$ ,  $a_1^C = (a_1, \dots, a_{n'}) \in A_1^C$  and  $a_2^C = (a_{n'+1}, \dots, a_n) \in A_2^C$  we have  $\delta^C(s, (a_1^C, a_2^C)) = \delta(s, (a_1, \dots, a_n))$ .

Furthermore, for any reward structure  $r = (r_A, r_S)$  of  $\mathbf{G}$ , the corresponding reward structure  $r^C = (r_A^C, r_S^C)$  of  $\mathbf{G}^C$  is constructed in a similar manner.

**Definition 5 (rPATL semantics).** For CSG  $\mathbf{G}$  the satisfaction relation  $\models$  for rPATL is defined inductively on the structure of the formula. The cases of **true**, atomic propositions, negations and conjunction of formulae are defined in the usual way. For temporal operators and  $s \in S$  we have:

$$s \models \langle\langle C \rangle\rangle P_{\sim q}[\psi] \Leftrightarrow \exists \sigma_1 \in \Sigma_{\mathbf{G}^C}^1. \forall \sigma_2 \in \Sigma_{\mathbf{G}^C}^2. \mathbb{P}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}(\psi) \sim q$$

$$s \models \langle\langle C \rangle\rangle R_{\sim x}^r[\rho] \Leftrightarrow \exists \sigma_1 \in \Sigma_{\mathbf{G}^C}^1. \forall \sigma_2 \in \Sigma_{\mathbf{G}^C}^2. \mathbb{E}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}[\text{rew}(r^C, \rho)] \sim x$$

where  $\mathbb{P}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}(\psi) = \text{Prob}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2} \{ \pi \in \text{IPaths}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2} \mid \pi \models \psi \}$  and for  $\pi \in \text{IPaths}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}$ :

$$\begin{aligned}
 \pi \models \mathbf{X} \phi &\Leftrightarrow \pi(1) \models \phi \\
 \pi \models \phi_1 \mathbf{U}^{\leq k} \phi_2 &\Leftrightarrow \exists i \leq k. (\pi(i) \models \phi_2 \wedge \forall j < i. \pi(j) \models \phi_1) \\
 \pi \models \phi_1 \mathbf{U} \phi_2 &\Leftrightarrow \exists i \in \mathbb{N}. (\pi(i) \models \phi_2 \wedge \forall j < i. \pi(j) \models \phi_1) \\
 \text{rew}(r^C, \mathbf{I}^{\leq k})(\pi) &= r_S(\pi(k)) \\
 \text{rew}(r^C, \mathbf{C}^{\leq k})(\pi) &= \sum_{i=0}^{k-1} (r_A(\pi(i), \pi[i]) + r_S(\pi(i))) \\
 \text{rew}(r^C, \mathbf{C})(\pi) &= \sum_{i=0}^{\infty} (r_A(\pi(i), \pi[i]) + r_S(\pi(i))) \\
 \text{rew}(r^C, \mathbf{F}^c \phi)(\pi) &= \begin{cases} \sum_{i=0}^{\infty} (r_A(\pi(i), \pi[i]) + r_S(\pi(i))) & \text{if } \forall j \in \mathbb{N}. \pi(j) \not\models \phi \\ \sum_{i=0}^{k_\phi} (r_A(\pi(i), \pi[i]) + r_S(\pi(i))) & \text{otherwise} \end{cases}
 \end{aligned}$$

and  $k_\phi = \min\{k-1 \mid \pi(k) \models \phi\}$ .

For CSG  $\mathbf{G}$ , coalition  $C$ , state  $s$ , path formula  $\psi$ , reward structure  $r$  and reward formula  $\rho$ , we define the following optimal values of  $\mathbf{G}^C$ :

$$\begin{aligned}
 \mathbb{P}_{\mathbf{G}, s}^C(\psi) &\stackrel{\text{def}}{=} \sup_{\sigma_1 \in \Sigma_{\mathbf{G}^C}^1} \inf_{\sigma_2 \in \Sigma_{\mathbf{G}^C}^2} \mathbb{P}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}(\psi) \\
 \mathbb{E}_{\mathbf{G}, s}^C(r, \rho) &\stackrel{\text{def}}{=} \sup_{\sigma_1 \in \Sigma_{\mathbf{G}^C}^1} \inf_{\sigma_2 \in \Sigma_{\mathbf{G}^C}^2} \mathbb{E}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}(r, \rho).
 \end{aligned} \tag{1}$$

As in the case of TSGs [9], negated path formulae can be represented by inverting the probability threshold, e.g.:  $\langle\langle C \rangle\rangle_{\mathbb{P}_{\geq q}}[\neg\psi] \equiv \langle\langle C \rangle\rangle_{\mathbb{P}_{\leq 1-q}}[\psi]$ , notably allowing the ‘globally’ operator to be specified:  $\mathbf{G} \phi \equiv \neg(\mathbf{F} \neg\phi)$ .

As for other probabilistic temporal logics, it is useful to consider *numerical* state formulae of the form  $\langle\langle C \rangle\rangle_{\mathbb{P}_{\min=?}}[\psi]$ ,  $\langle\langle C \rangle\rangle_{\mathbb{P}_{\max=?}}[\psi]$ ,  $\langle\langle C \rangle\rangle_{\mathbb{R}_{\min=?}}^r[\rho]$  and  $\langle\langle C \rangle\rangle_{\mathbb{R}_{\max=?}}^r[\rho]$ . For example, for state  $s$  we have:

$$\begin{aligned}
 \langle\langle C \rangle\rangle_{\mathbb{P}_{\min=?}}[\psi] &\stackrel{\text{def}}{=} \inf_{\sigma_1 \in \Sigma_{\mathbf{G}^C}^1} \sup_{\sigma_2 \in \Sigma_{\mathbf{G}^C}^2} \mathbb{P}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}(\psi) \\
 \langle\langle C \rangle\rangle_{\mathbb{P}_{\max=?}}[\psi] &\stackrel{\text{def}}{=} \sup_{\sigma_1 \in \Sigma_{\mathbf{G}^C}^1} \inf_{\sigma_2 \in \Sigma_{\mathbf{G}^C}^2} \mathbb{P}_{\mathbf{G}^C, s}^{\sigma_1, \sigma_2}(\psi).
 \end{aligned}$$

As CSGs are *determined* with respect to the properties we consider [18], i.e. the maximum value coalition  $C$  can ensure equals the minimum value coalition  $N \setminus C$  can ensure, the above values are the *optimal values* for the respective properties in  $\mathbf{G}$ . The determinacy result also yields the following equivalences:

$$\langle\langle C \rangle\rangle_{\mathbb{P}_{\max=?}}[\psi] \equiv \langle\langle N \setminus C \rangle\rangle_{\mathbb{P}_{\min=?}}[\psi] \quad \text{and} \quad \langle\langle C \rangle\rangle_{\mathbb{R}_{\max=?}}^r[\rho] \equiv \langle\langle N \setminus C \rangle\rangle_{\mathbb{R}_{\min=?}}^r[\rho].$$

**Example 2.** Returning to Example 1, we can use rPATL to specify the following properties of the rock-paper-scissors CSG:

- $\langle\langle \{1\} \rangle\rangle_{\mathbb{P}_{\geq 1}}[\mathbf{F} \text{win}_1]$  player 1 can ensure it eventually wins a round of the game with probability 1;
- $\langle\langle \{2\} \rangle\rangle_{\mathbb{P}_{\max=?}}[\neg \text{win}_1 \mathbf{U} \text{win}_2]$  the maximum probability with which player 2 can ensure it wins a round of the game before player 1;
- $\langle\langle \{2\} \rangle\rangle_{\mathbb{R}_{\max=?}}^{\text{utility}2}[\mathbf{C}^{\leq 2 \cdot K}]$  the maximum expected utility player 2 can ensure over  $K$  rounds (*utility2* is the reward structure that assigns rewards  $-1, 0$  and  $1$  to the states labelled  $\text{win}_1, \text{draw}$  and  $\text{win}_2$ , respectively).

## 4 Model Checking CSGs Against rPATL

Next, we present an algorithm for model checking an rPATL formula  $\phi$  against a CSG  $G$ . The overall approach is the standard one for branching-time logics, i.e., determining the set  $Sat(\phi)$  recursively. The computation of this set for atomic propositions and logical connectives is straightforward, and therefore we concentrate on state formulae of the form  $\langle\langle C \rangle\rangle P_{\sim q}[\psi]$  and  $\langle\langle C \rangle\rangle R_{\sim x}[\rho]$ . For these, the problem reduces to computing optimal values for the coalition game  $G^C$ , see (1). In particular, for  $\sim \in \{\geq, >\}$  and  $s \in S$  we have:

$$s \models \langle\langle C \rangle\rangle P_{\sim q}[\psi] \Leftrightarrow \mathbb{P}_{G,s}^C(\psi) \sim q \quad \text{and} \quad s \models \langle\langle C \rangle\rangle R_{\sim x}[\rho] \Leftrightarrow \mathbb{E}_{G,s}^C(\rho) \sim x.$$

and, since CSGs are determined for rPATL properties, for  $\sim \in \{<, \leq\}$  we have:

$$s \models \langle\langle C \rangle\rangle P_{\sim q}[\psi] \Leftrightarrow \mathbb{P}_{G,s}^{N \setminus C}(\psi) \sim q \quad \text{and} \quad s \models \langle\langle C \rangle\rangle R_{\sim x}[\rho] \Leftrightarrow \mathbb{E}_{G,s}^{N \setminus C}(\rho) \sim x.$$

Therefore, we focus on computing  $\mathbb{P}_{G,s}^C(\psi)$  and  $\mathbb{E}_{G,s}^C(\rho)$  for a fixed CSG  $G$ , coalition  $C$  and state  $s$ . We assume that the available actions of players 1 and 2 of the (two-player) CSG  $G^C$  in state  $s$  are  $\{a_1, \dots, a_l\}$  and  $\{b_1, \dots, b_m\}$ , respectively.

**Matrix Games.** The computation of optimal probabilities and expected reward values requires finding values of matrix games. These values can be found through the solution to an LP problem as presented in Sect. 2. Solution methods for such problems include Simplex, branch-and-bound and interior point.

**Probabilistic Formulae.** We now show how to compute the optimal probability  $\mathbb{P}_{G,s}^C(\psi)$  for each state  $s$  and path formula  $\psi$ . If  $\psi = \mathbf{X}\phi$ , then for any state  $s$  we have that  $\mathbb{P}_{G,s}^C(\mathbf{X}\phi) = val(\mathbf{Z})$  where  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  is the matrix game with:

$$z_{i,j} = \sum_{s' \in Sat(\phi)} \delta(s, (a_i, b_j))(s').$$

If  $\psi = \phi_1 \mathbf{U}^{\leq k} \phi_2$ , the values can be computed recursively as follows.

- $\mathbb{P}_{G,s}^C(\phi_1 \mathbf{U}^{\leq 0} \phi_2) = 1$  if  $s \in Sat(\phi_2)$  and 0 otherwise;
- $\mathbb{P}_{G,s}^C(\phi_1 \mathbf{U}^{\leq k+1} \phi_2)$  equals 1 if  $s \in Sat(\phi_2)$ , else equals 0 if  $s \notin Sat(\phi_1)$  and otherwise equals  $val(\mathbf{Z})$  where  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  is the matrix game with:

$$z_{i,j} = \sum_{s' \in S} \delta(s, (a_i, b_j))(s') \cdot \mathbb{P}_{G,s'}^C(\phi_1 \mathbf{U}^{\leq k} \phi_2).$$

If  $\psi = \phi_1 \mathbf{U} \phi_2$ , the probability values can be computed through *value iteration* [23], i.e. using the fact that  $(\mathbb{P}_{G,s}^C(\phi_1 \mathbf{U}^{\leq k} \phi_2))_{k \in \mathbb{N}}$  is a non-decreasing sequence converging to  $\mathbb{P}_{G,s}^C(\phi_1 \mathbf{U} \phi_2)$  and computing  $\mathbb{P}_{G,s}^C(\phi_1 \mathbf{U}^{\leq k} \phi_2)$  for sufficiently large  $k$ , i.e. terminating the computation when the difference between the values for  $k$  and  $k+1$  are less than some threshold  $\varepsilon$ . Alternatively, *policy iteration* could be used, e.g., see [6]. In both cases, the qualitative algorithms of [1] can be used to precompute states for which the probability is either 0 or 1.

**Reward Formulae.** We next show how to compute the expected reward values  $\mathbb{E}_{G,s}^C(r, \rho)$  for each state  $s$  and path formula  $\rho$ . If  $\rho = \mathbf{I}^{\leq k}$ , the values can be computed recursively as follows.

- $\mathbb{E}_{\mathbf{G},s}^C(r, \mathbf{I}^=0) = r_S(s)$ ;
- $\mathbb{E}_{\mathbf{G},s}^C(r, \mathbf{I}^=k+1)$  equals  $\text{val}(\mathbf{Z})$  where  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  is the matrix game with:

$$z_{i,j} = \sum_{s' \in S} \delta(s, (a_i, b_j))(s') \cdot \mathbb{E}_{\mathbf{G},s'}^C(r, \mathbf{I}^=k).$$

If  $\rho = \mathbf{C}^{\leq k}$ , then the values can be computed recursively as follows.

- $\mathbb{E}_{\mathbf{G},s}^C(r, \mathbf{C}^{\leq 0}) = 0$ ;
- $\mathbb{E}_{\mathbf{G},s}^C(r, \mathbf{C}^{\leq k+1})$  equals  $\text{val}(\mathbf{Z})$  where  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  is the matrix game with:

$$z_{i,j} = r_A(s, (a_i, b_j)) + r_S(s) + \sum_{s' \in S} \delta(s, (a_i, b_j))(s') \cdot \mathbb{E}_{\mathbf{G},s'}^C(r, \mathbf{C}^{\leq k}).$$

For the remaining reward formulae we restrict the use of negative rewards to ensure correctness. For total rewards ( $\rho = \mathbf{C}$ ) we require that all negative rewards are associated with state-action pairs that reach an absorbing state (a state where all rewards are zero and which cannot be left) with probability 1. For reachability rewards ( $\rho = \mathbf{F}^c \phi$ ) we require all negative rewards are associated with state-action pairs that reach a target or absorbing state with probability 1.

If  $\rho = \mathbf{C}$ , we first find the states for which the optimal expected reward values are infinite. Similarly to [9], we can use the qualitative algorithms of [1] to find these states as they can also be defined as those states for which the coalition  $C$  can ensure the probability of reaching states with positive reward infinitely often is greater than 0. After removing these states from  $\mathbf{G}^C$ , the remaining values can be computed as the limit of the (non-decreasing sequence of) bounded cumulative reward values:

$$\mathbb{E}_{\mathbf{G},s}^C(r, \mathbf{C}) = \lim_{k \rightarrow \infty} \mathbb{E}_{\mathbf{G},s}^C(r, \mathbf{C}^{\leq k}).$$

Finally, if  $\rho = \mathbf{F}^c \phi$ , then we first make all states of  $\mathbf{G}^C$  satisfying  $\phi$  absorbing. Next, as in the case above, we find the states of  $\mathbf{G}^C$  for which the optimal expected reward values are infinite and remove them from the game. The values for the remaining states can then be computed through value iteration where  $\mathbb{E}_{\mathbf{G},s}^C(\mathbf{F}^c \phi) = \lim_{k \rightarrow \infty} \mathbb{E}_{\mathbf{G},s}^C(\mathbf{F}^c \phi)_k$ ,  $\mathbb{E}_{\mathbf{G},s}^C(\mathbf{F}^c \phi)_0 = 1$  if  $s \in \text{Sat}(\phi)$  and 0 otherwise and  $\mathbb{E}_{\mathbf{G},s}^C(\mathbf{F}^c \phi)_{k+1}$  equals 1 if  $s \in \text{Sat}(\phi_2)$  and otherwise equals  $\text{val}(\mathbf{Z})$ , and  $\mathbf{Z} \in \mathbb{R}^{l \times m}$  is the matrix game with:

$$z_{i,j} = r_A(s, (a_i, b_j)) + r_S(s) + \sum_{s' \in S} \delta(s, (a_i, b_j))(s') \cdot \mathbb{E}_{\mathbf{G},s'}^C(\mathbf{F}^c \phi)_k.$$

**Strategy Synthesis.** In addition to verifying rPATL formulae, it is typically also very useful to perform *strategy synthesis*, i.e., to construct a witness to the satisfaction of a property. For example, in the case of the probabilistic property  $\langle\langle C \rangle\rangle \mathbb{P}_{\sim q}[\psi]$ , this means finding a strategy  $\sigma_1^*$  for the coalition  $C$  such that:

$$\mathbb{P}_{\mathbf{G}_C}^{\sigma_1^*, \sigma_2'}(\psi) \sim q \quad \text{for all } \sigma_2' \in \Sigma_{\mathbf{G}_C}^2$$

and, in the case of a numerical state formula  $\langle\langle C \rangle\rangle \mathbb{P}_{\max=?}[\psi]$ , means finding an optimal strategy  $\sigma_1^*$  for the coalition  $C$ , i.e. a strategy such that:

$$\mathbb{P}_{\mathbf{G}_C}^{\sigma_1^*, \sigma_2'}(\psi) \geq \mathbb{P}_{\mathbf{G},s}^C(\psi) = \sup_{\sigma_1 \in \Sigma_{\mathbf{G}_C}^1} \inf_{\sigma_2 \in \Sigma_{\mathbf{G}_C}^2} \mathbb{P}_{\mathbf{G}_C}^{\sigma_1, \sigma_2}(\psi) \quad \text{for all } \sigma_2' \in \Sigma_{\mathbf{G}_C}^2.$$

For unbounded properties, we can synthesise a strategy which is memoryless, but needs randomisation; bounded properties require both finite-memory and randomisation. This differs from TSGs where deterministic strategies are sufficient in both cases. We can synthesise such strategies using the approach above for computing optimal values and keeping track of the optimal strategy of player 1 for the matrix game solved in each state. Since we use value iteration, only  $\varepsilon$ -optimal strategies can be synthesised for unbounded properties, where  $\varepsilon$  is the convergence criterion used when performing value iteration [24].

**Correctness and Complexity.** We conclude this section with a discussion of correctness and complexity. The overall (recursive) approach and the reduction to solution of a two-player game is essentially the same as for TSGs [9], and therefore the same correctness arguments apply. The correctness of value iteration for unbounded properties follows from [23] and for bounded properties from Definition 5 and the solution of matrix games (see Sect. 2). Regarding complexity, due to the recursive nature of the algorithm, it is linear in the size of the formula  $\phi$ , while in the worst case finding the optimal values of a 2-player CSG is PSPACE [7]. In practice, we use value iteration, which solves an LP problem of size  $|A|$  for each state at each iteration, with the number of iterations depending on the convergence criterion. The efficiency in practice is reported in Sect. 6.

## 5 Implementation and Tool Support

We have implemented support for modelling and automated verification of CSGs as an extension of PRISM-games [16], which only handled TSGs. The tool, and the files for the case studies described in the next section, are available from [28].

**Modelling.** CSGs are specified using the same language as for TSGs, itself an extension of the original PRISM modelling language (Fig. 2 shows an example). It allows multiple parallel components, called *modules*, operating both asynchronously and synchronously. Each module's state is defined by a number of finite-valued variables, and its behaviour by a set of probabilistic guarded commands  $[a] g \rightarrow u$ , comprising an action label  $a$ , guard  $g$  and probabilistic update  $u$ . If the guard (a predicate over the variables of all modules) is satisfied, then the module can (probabilistically) update its variables according to  $u$ . Modules interact by either reading the values of each other's variables, or synchronising (moving simultaneously) on commands labelled with the same action.

To specify a CSG, the model description must also define a list of players and the (disjoint) sets of actions each controls. We adopt the syntax from PRISM-games, but the semantics differs from TSGs. In a state of a CSG, each player chooses to perform one of the commands that is enabled (the guard is true) and is labelled by an action under its control (if no command is enabled, the player idles, i.e. chooses  $\perp$ ). Unlike standard PRISM models, the chosen commands for all players then execute synchronously, despite being labelled with distinct actions. To remain consistent with PRISM's conventions, we require that each variable is updated by at most one player and each player's updates are independent of



the other players' choices. This has not proven restrictive (see, e.g., the range of examples modelled in Sect. 6), but we plan to relax these constraints.

**Example 3.** Figure 2 shows a model description for the rock-paper-scissors CSG of Example 1. Player 1 is represented by module *player1*, with variable *m1*, and its commands are labelled with the actions *r1*, *p1*, *s1*, *t1* (corresponding to *r*, *p*, *s*, *t* in Fig. 1(a)). In this example, the updates are all non-probabilistic. Player 2 is identical in structure to player 1 and is defined using PRISM's module renaming feature. Labels (defining the atomic propositions from Fig. 1(a)) and reward structures are also defined in the standard way for PRISM models.

```

csg

player player1 [r1], [p1], [s1], [t1] endplayer
player player2 [r2], [p2], [s2], [t2] endplayer

module player1
  m1 : [0..3];
  [r1] m1=0 → (m1'=1); // rock
  [p1] m1=0 → (m1'=2); // paper
  [s1] m1=0 → (m1'=3); // scissors
  [t1] m1>0 → (m1'=0); // restart
endmodule

// second player constructed through renaming
module player2 = player1[m1=m2, r1=r2, p1=p2, s1=s2, t1=t2] endmodule

label "win1" = (m1=1 & m2=3) | (m1=2 & m2=1) | (m1=3 & m2=2); // player 1 wins round
label "win2" = (m2=1 & m1=3) | (m2=2 & m1=1) | (m2=3 & m1=2); // player 2 wins round
label "draw" = (m2=1 & m1=1) | (m2=2 & m1=2) | (m2=3 & m1=3); // draw

rewards "utility2" // utility for player 2
  [t1] (m1=1 & m2=3) | (m1=2 & m2=1) | (m1=3 & m2=2) : -1; // player 1 wins
  [t1] (m1=1 & m2=2) | (m1=2 & m2=3) | (m1=3 & m2=1) : 1; // player 2 wins
endrewards

```

Fig. 2. PRISM language specification of the CSG from Example 1.

**Implementation.** Our tool constructs a CSG from a given model specification and implements the rPATL model checking and strategy synthesis algorithms from Sect. 4. We adapt the existing modelling and property language parsers and various other pieces of basic model checking functionality from PRISM-games. We store and verify CSGs using an extension of PRISM's explicit-state (sparse matrix based) model checking engine, which is implemented in Java. A notable addition to this is the solution of values of matrix games, which is performed via linear programming (see Sect. 2) using the LPSolve library [17].

## 6 Case Studies and Experimental Results

To demonstrate the applicability of our techniques and tool, and to evaluate their performance, we now present results from a variety of case studies. This also illustrates the utility of CSGs over TSGs. As mentioned earlier, the tool and examples (models and properties) are available from [28].

**Efficiency and Scalability.** We begin by presenting a selection of results regarding the performance of our implementation. The models on which these are based are described in more detail in subsequent sections. Table 1 shows results for a representative selection of models and rPATL properties, verified using a 2.8 GHz Intel X5660 with 32 GB RAM. We give model statistics (number of players, states and transitions) and the times for model construction, qualitative and quantitative verification (i.e. value iteration). Our tool is able to analyse models with up to approximately 3 million states. Models with 10,000 states can be built and verified in a few seconds; the largest takes little over an hour. The most significant cost in terms of time is the need to (repeatedly) solve a matrix game in each state, but verification times are not quite linear in the model size due to variations in the number of iterations needed.

**Table 1.** Statistics for a representative set of CSG verification instances.

Case study & rPATL property [parameters]	Param. values	CSG statistics			Const. time (s)	Verif. time (s)	
		Players	States	Transitions		Qual.	Quant.
<i>Future markets investors</i> $\langle\langle i1 \rangle\rangle_{R_{\max=?}}^{\text{profit}} [F^c \text{ cashed\_in}_1]$ [months]	3	5	3,664	13,482	0.4	0.2	0.8
	5	5	18,671	82,494	1.1	1.1	5.1
	7	5	53,799	247,807	2.3	6.6	20.3
	9	5	116,838	561,538	4.7	25.9	59.2
<i>User-centric networks</i> $\langle\langle user \rangle\rangle_{R_{\max=?}}^{\text{unpaid}} [F^c \text{ services}=K]$ [td, K]	1,1	7	1,029	2,386	0.4	0.1	0.6
	1,3	7	18,218	50,181	1.9	0.3	8.9
	1,5	7	145,561	458,169	11.0	3.3	110.5
	1,7	7	755,531	2,651,829	61.6	12.4	814.6
	1,9	7	2,993,308	11,461,723	269.0	57.1	4,205.3
<i>Intrusion detection system</i> $\langle\langle policy \rangle\rangle_{R_{\min=?}}^{\text{damage}} [C]$ [rounds]	25	4	581	1,616	0.2	0.1	1.6
	50	4	1,181	3,316	0.3	0.1	4.2
	100	4	2,381	6,716	0.3	0.4	15.3
	200	4	4,781	13,516	0.4	0.5	71.3
<i>Jamming radio systems</i> $\langle\langle user \rangle\rangle_{P_{\max=?}} [F \text{ sent} \geq \text{slots}/2]$ [slots]	10	3	7,921	1,038,384	3.8	6.6	2.8
	15	3	17,281	2,421,504	9.9	22.3	9.1
	20	3	30,241	4,380,624	13.3	60.1	22.5
	25	3	46,801	6,915,744	22.8	144.5	43.3

**Futures Market Investors.** The first of our case studies is a futures market investor model [19], which represents the interactions between investors and a stock market. For the TSG model of [19], in successive months, a single investor chooses whether to invest, next the market decides whether to bar the investor, and then the values of shares and a cap on values are updated probabilistically. We have built and analysed several CSGs variants of the model, analysing optimal strategies for investors under adversarial conditions. First, we made the

investor and market take their decisions concurrently, and verified that this yielded no additional gain for the investor (see [28]). This is because the market and investor have the same information, and so the market knows when it is optimal for the investor to invest without needing to see its decision.

We next modelled two competing investors who simultaneously decide whether to invest (and, as above, the market simultaneously decides whether to bar each of them). If the two investors cash in their shares in the same month, then their profits are reduced. We also consider several distinct profit models: ‘normal market’, ‘later cash-ins’ and ‘later cash-ins with fluctuation’. The first is from [19] and the latter two reward postponing cashing in shares (see [28] for details). The CSG has 5 players: one for each investor, one deciding on the barring of investors, one controlling share values and one updating the month. We study both the maximum profit of one investor and the maximum combined profit of both investors. For comparison, we also build a TSG model in which the investors first take turns to decide whether to invest (the ordering decided by the market) and then the market decides on whether to bar any of the investors.

Figure 3 shows the maximum expected value over a fixed number of months under the ‘normal market’ for both the profit of first investor and the combined profit of the two investors. For the former, we show results for the first investor acting alone ( $\langle\langle i1 \rangle\rangle$ ) and when in a coalition with the second investor ( $\langle\langle i1, i2 \rangle\rangle$ ). We plot the corresponding results from the TSG model for comparison. Figure 4 shows the maximum expected combined profit for the other two profit models.

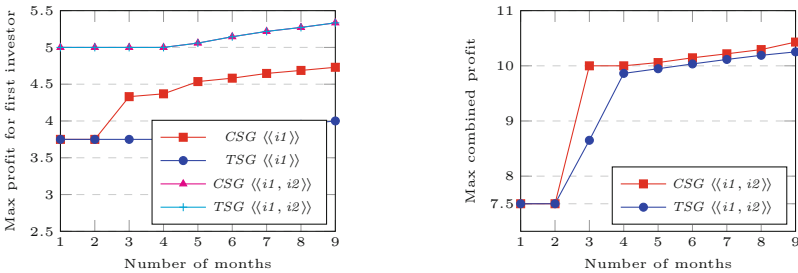


Fig. 3. Futures market investors: normal market.

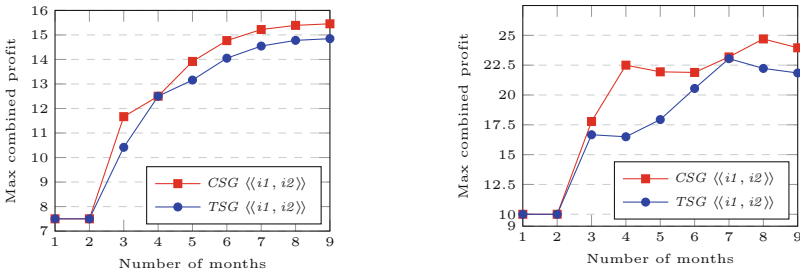


Fig. 4. Futures market: later cash-ins without (left) and with (right) fluctuations.

When investors cooperate to maximise the profit of the first, results for the CSG and TSG models coincide. This follows from the discussion above since all the second investor can do is make sure it does not invest at the same time as the first. For the remaining cases and given sufficient months, there is always a strategy in the concurrent setting that outperforms all turn-based strategies. The increase in profit for a single investor in the CSG model is due to the fact that, as the investors decisions are concurrent, the second cannot ensure it invests at the same time as the first, and hence decrease the profit of the first. In the case of combined profit, the difference arises because, although the market knows when it is optimal for one investor to invest, in the CSG model the market does not know which one will, and therefore may choose the wrong investor to bar.

We performed strategy synthesis to study the optimal actions of investors. By way of example, consider  $\langle\langle i1 \rangle\rangle \mathbf{R}_{\max=?}^{profit1} [F^c \text{ cashed\_in}_1]$  over three months and for a normal market (see Fig. 3 left). The optimal TSG strategy for the first investor is to invest in the first month (which the market cannot bar) ensuring an expected profit of 3.75. The optimal (randomised) CSG strategy is to invest:

- in the first month with probability 0.494949;
- in the second month with probability 1, if the second investor has cashed in;
- in the second month with probability 0.964912, if the second investor did not cash in at the end of the first month and the shares went up;
- in the second month with probability 0.954023, if the second investor did not cash in at the end of the first month and the shares went down;
- in the third month with probability 1 (this is the last month to invest).

Following this strategy, the first investor ensures an expected profit of  $\sim 4.33$ .

**Trust Models for User-Centric Networks.** Trust models for user-centric networks were analysed previously using TSGs in [15]. The analysis considered the impact of different parameters on the effectiveness of cooperation mechanisms between service providers. The providers share information on the measure of *trust* for users in a *reputation*-based setting. Each measure of trust is based on the service’s previous interactions with the user (which services they paid for). In the original TSG model, a single user can either make a request to one of three service providers or buy the service directly by paying maximum price. If the user makes a request to a service provider, then the provider decides to accept or deny the request based on the user’s trust measure. If the request was accepted, the provider would next decide on the price again based on the trust measure, and the user would then decide whether to pay for the service and finally the provider would update its trust measure based on whether there was a payment. This sequence of steps would have to take place before any other interactions occurred between the user and other providers.

Here we consider CSG models allowing the user to make requests and pay different service providers simultaneously and for the different providers to execute requests concurrently. There are 7 players: one for the user’s interaction with each service provider, one for the user buying services directly and one for each of the 3 service providers. Three trust models were considered. In the

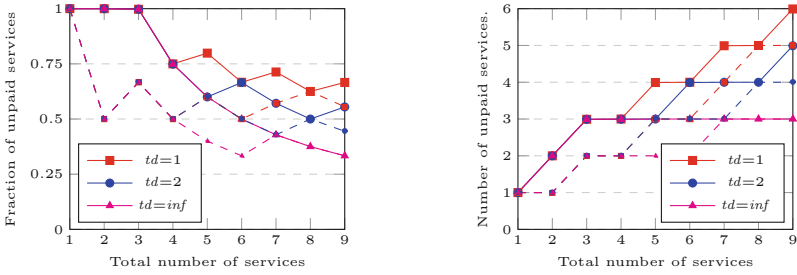


Fig. 5. User-centric network results (CSG/TSG values as solid/dashed lines).

first, the trust level was decremented by 1 ( $td = 1$ ) when the user does not pay, decremented by 2 in the second ( $td = 2$ ) and reset to 0 in the third ( $td = inf$ ).

Figure 5 presents results for the maximum fraction and number of unpaid services the user can ensure for each trust model. The results for the original TSG model are included as dashed lines. The results demonstrate that the user can take advantage of the fact that in the CSG model it can request multiple services at the same time, and obtain more services without paying before the different providers get a chance to inform each other about non-payment. In addition, the results show that having a more severe penalty on the trust measure for non-payment decreases the unpaid services the user can obtain.

**Intrusion Detection Policies.** In [26], CSGs are used to model the interaction between an intrusion detection policy and attacker. The policy has a number of libraries it can use to detect attacks and the attacker has a number of different attacks which can incur different levels of damage if not detected. Furthermore, each library can only detect certain attacks. In the model, in each round the policy chooses a library to deploy and the attacker chooses an attack. A reward structure is specified representing the level of damage when an attack is not detected. The goal is to find optimal intrusion detection policies which correspond to finding a strategy for the policy that minimises damage. We have constructed CSG models with 4 players (representing the policy, attacker, system and time) for the two scenarios outlined in [26]. We have synthesised optimal policies which ensure the minimum cumulative damage over a fixed number of rounds and damage in a specific round. Here concurrency is required for the game to be meaningful, otherwise it is easy for the player whose turn follows the other player's to 'win'. For example, if the attacker knows what library is being deployed, then it can simply choose an attack the library cannot detect.

**Jamming Multi-channel Radio Systems.** A CSG model for jamming multi-channel cognitive radio systems is presented in [27]. The system consists of a number of channels which can be an occupied or idle state. The state of each channel remains fixed within a time slot and between slots is Markovian (i.e. the state changes randomly based only on the state of the channel in the previous slot). A secondary user has a subset of available channels and at each time-slot

must decide which to use. There is a single attacker which again has a subset of available channels and at each time slot decides to send a jamming signal over one of them. The CSG has 3 players representing the secondary user, attacker and environment. We synthesise strategies for the secondary user which maximise the probability of ensuring at least half the messages are sent correctly. Again concurrency is required as otherwise, e.g., the attacker can observe the user and then jam the chosen channel.

## 7 Conclusion

We have designed and implemented an approach for the automatic verification of CSGs. We have extended the semantics of the temporal logic rPATL to CSGs and presented a new modelling approach based on the PRISM language to specify such games. We have proposed and implemented algorithms for verification and strategy synthesis as an extension of the PRISM-games model checker. Finally, we have evaluated the approach on a range of case studies.

There are a number of directions for future work. First, we plan to consider additional properties (e.g. Nash equilibria and multi-objective queries). We are also working on extending the implementation to consider alternative solution methods (e.g. policy iteration and using CPLEX [12] to solve matrix games) and a symbolic (binary decision diagram based) implementation. Lastly, we are considering extending the approach to partially observable strategies.

**Acknowledgements.** This work is partially supported by the EPSRC Programme Grant on Mobile Autonomy and the PRINCESS project, under the DARPA BRASS programme.

## References

1. de Alfaro, L., Henzinger, T.: Concurrent omega-regular games. In: LICS 2000 (2000)
2. de Alfaro, L., Henzinger, T., Kupferman, O.: Concurrent reachability games. TCS **386**(3), 188–217 (2007)
3. de Alfaro, L., Majumdar, R.: Quantitative solution of omega-regular games. JCSS **68**(2), 374–397 (2004)
4. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5), 672–713 (2002)
5. Brenguier, R.: PRALINE: a tool for computing nash equilibria in concurrent games. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 890–895. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39799-8\\_63](https://doi.org/10.1007/978-3-642-39799-8_63)
6. Chatterjee, K., de Alfaro, L., Henzinger, T.: Strategy improvement for concurrent reachability and turn-based stochastic safety games. JCSS **79**(5), 640–657 (2013)
7. Chatterjee, K., Henzinger, T.: A survey of stochastic  $\omega$ -regular games. JCSS **78**(2), 394–413 (2012)
8. Chatterjee, K., Henzinger, T.A., Jobstmann, B., Radhakrishna, A.: GIST: a solver for probabilistic games. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 665–669. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14295-6\\_57](https://doi.org/10.1007/978-3-642-14295-6_57)

9. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. *FMSD* **43**(1), 61–92 (2013)
10. Cheng, C.-H., Knoll, A., Luttenberger, M., Buckl, C.: GAVS+: an open platform for the research of algorithmic game solving. In: Abdulla, P.A., Leino, K.R.M. (eds.) *TACAS 2011*. LNCS, vol. 6605, pp. 258–261. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-19835-9\\_22](https://doi.org/10.1007/978-3-642-19835-9_22)
11. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčák, V. (eds.) *CAV 2017*. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-63390-9\\_31](https://doi.org/10.1007/978-3-319-63390-9_31)
12. ILOG CPLEX. [ibm.com/products/ilog-cplex-optimization-studio](http://ibm.com/products/ilog-cplex-optimization-studio)
13. Kemeny, J., Snell, J., Knapp, A.: *Denumerable Markov Chains*. Graduate Texts in Mathematics, vol. 40. Springer, New York (1976). <https://doi.org/10.1007/978-1-4684-9455-6>
14. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *CAV 2011*. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
15. Kwiatkowska, M., Parker, D., Simaitis, A.: Strategic analysis of trust models for user-centric networks. In: *Proceedings of SR 2013*. EPTCS, p. 112 (2013)
16. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-games: verification and strategy synthesis for stochastic multi-player games with multiple objectives. *STTT* **20**(2), 195–210 (2018)
17. LPSolve (version 5.5). [lpsolve.sourceforge.net/5.5/](http://lpsolve.sourceforge.net/5.5/)
18. Martin, D.: The determinacy of Blackwell games. *J. Symb. Log* **63**(4), 1565–1581 (1998)
19. McIver, A., Morgan, C.: Results on the quantitative mu-calculus  $\mu$ mu. *ACM Trans. Comput. Log.* **8**(1), 3 (2007)
20. Nash, J.: Equilibrium points in  $n$ -person games. *Proc. Natl. Acad. Sci* **36**, 48–49 (1950)
21. von Neumann, J.: Zur theorie der gesellschaftsspiele. *Mathematische Annalen* **100**, 295–320 (1928)
22. von Neumann, J., Morgenstern, O., Kuhn, H., Rubinstein, A.: *Theory of Games and Economic Behavior*. Princeton University Press, Princeton (1944)
23. Raghavan, T., Filar, J.: Algorithms for stochastic games – a survey. *Zeitschrift für Oper. Res.* **35**(6), 437–472 (1991)
24. Svorenova, M., Kwiatkowska, M.: Quantitative verification and strategy synthesis for stochastic games. *Eur. J. Control* **30**, 15–30 (2016)
25. Toumi, A., Gutierrez, J., Wooldridge, M.: A tool for the automated verification of nash equilibria in concurrent games. In: Leucker, M., Rueda, C., Valencia, F.D. (eds.) *ICTAC 2015*. LNCS, vol. 9399, pp. 583–594. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-25150-9\\_34](https://doi.org/10.1007/978-3-319-25150-9_34)
26. Zhu, Q., Başar, T.: Dynamic policy-based IDS configuration. In: *CDC 2009* (2009)
27. Zhu, Q., Li, H., Han, Z., Basar, T.: A stochastic game model for jamming in multi-channel cognitive radio systems. In: *Proceedings of the ICC 2010*. IEEE (2010)
28. Supporting material. [www.prismmodelchecker.org/files/qest18/](http://www.prismmodelchecker.org/files/qest18/)



# Bounded Verification of Reachability of Probabilistic Hybrid Systems

Ratan Lal<sup>(✉)</sup> and Pavithra Prabhakar<sup>(✉)</sup>

Kansas State University, Manhattan, USA  
{ratan,pprabhakar}@ksu.edu

**Abstract.** In this paper, we consider the problem of bounded reachability analysis of probabilistic hybrid systems which model discrete, continuous and probabilistic behaviors. The discrete and probabilistic dynamics are modeled using a finite state Markov decision process (MDP), and the continuous dynamics is incorporated by annotating the states of the MDP with differential equations/inclusions. We focus on polyhedral dynamical systems to model continuous dynamics. Our broad approach for computing probabilistic bounds on reachability consists of the computation of the exact minimum/maximum probability of reachability within  $k$  discrete steps in a polyhedral probabilistic hybrid system by reducing it to solving an optimization problem with satisfiability modulo theory (SMT) constraints.

We have implemented analysis algorithms in a Python toolbox, and use the Z3opt optimization solver at the backend. We report the results of experimentation on a case study involving the analysis of the probability of the depletion of the charge in a battery used in the nano-satellite.

## 1 Introduction

Cyber-physical systems consist of software that control safety critical physical systems as in driverless cars, unmanned aerial vehicles and smart grids. In these systems, software exhibits discrete behaviors and interacts with physical processes, such as, the vehicle dynamics, which evolve continuously. Hybrid automata [16] have been traditionally used to model the mixed discrete-continuous behaviors in cyber-physical systems, with the aim of rigorous analysis. Another important feature of cyber-physical systems is uncertainty, which manifests due to the complex environment with which these systems interact. For instance, driverless cars need to navigate safely in an uncertain environment involving pedestrians. Hence, it is important to incorporate uncertainty into the model for the analysis of safety critical cyber-physical systems. However, analyzing for worst case uncertainty is often too conservative to obtain practically useful insights about the behaviors of the systems. On the other hand, often, it is possible to obtain distributions on the sources of uncertainty, which can in turn be used to provide probabilistic guarantees about the behaviors of the system.

In this paper, we consider probabilistic hybrid systems [25] (PHS) to capture discrete, continuous, and probabilistic behaviors. A PHS consists of a finite



number of modes and a finite number of continuous variables. Each mode is associated with a continuous dynamics that specifies the evolution of the continuous variables using differential equations or inclusions. This is similar to that of a hybrid automaton. In addition, a PHS consists of transitions that are non-deterministic as well as probabilistic. Hence, our underlying model is a Markov Decision Process (MDP), which is then extended with continuous dynamics in each mode. We are interested in analyzing the probability of reaching a target set of states  $\mathbb{F}$  within a given amount of time  $T$  and a bound on the number of discrete/probabilistic transitions  $k$ . Note that since our model encompasses both non-deterministic as well as probabilistic transitions, depending on how a scheduler resolves the non-determinism, there are different probabilities associated with reaching the target set  $\mathbb{F}$ . Hence, we consider the problem of computing the maximum and minimum probability of reaching the target set among all schedulers. The bounded verification problem, in general, captures the behavior of an under-approximation of the system where the number of transitions and the time of execution is constrained. However, it provides conservative bounds on the actual probabilities of an actual system. Also, in many cases, there is a practical upper bound on the total time, and a lower bound on the dwell time (how fast the system can switch), which justifies the problem of bounded verification as a complete method for verification of the full system.

In this paper, we restrict ourselves to polyhedral dynamics that are an important and widely prevalent class of dynamics for modeling physical processes. The main challenge towards the probabilistic analysis of reachability is the computation of maximum and minimum probability of reachability involves solving a complex optimization problem. To address the problem, we consider an encoding into a problem that involves optimization over SMT formulas with linear constraints, that can be solved efficiently using recent tools such as Z3opt [5] and SYMBA [20]. More precisely, our broad approach consists of computing exact bounds on the probability of reachability in the system. Next, we encode the computation trees of polyhedral PHS using SMT formulae. We use Z3opt and SYMBA solvers to find the maximum and minimum probabilities of reachability by optimizing the probability over constraints encoded in SMT. We have implemented our approach in a Python toolbox, and conducted experimental evaluation on a case study involving the depletion of charge in a kinetic battery used in a nano-satellite. The experimental results demonstrate the feasibility of the proposed approach. The main challenge towards scaling the approach will be the exponential growth of the variables and the size of the encoding with the number of discrete transitions. In many cases, certain timing constraints can enforce a practical bound on the number of transitions. However, this bound could be large and the SMT solver might fail to return. Our future work will focus on efficient encodings and heuristics for reducing/pruning the computation tree.

*Related work.* Stochastic hybrid systems [9] and their reachability problem [8] captures uncertain systems and safety properties. In general, probabilistic reachability analysis of stochastic hybrid systems is undecidable, however, it is

decidable for certain subclass of the systems [17, 19, 23]. Probabilistic reachability analysis of the following systems Markov decision processes [7, 10, 15, 21, 28], discrete time stochastic hybrid systems [1–4, 26], continuous time stochastic hybrid systems [6, 24, 27] and probabilistic hybrid systems [11, 29, 30] have been explored. Probabilistic reachability problem for the general class of probabilistic hybrid system has been solved using abstraction techniques such as [12, 22] have been applied in papers [29, 30], that give the bound on the maximum/minimum probability of reachability. Although the paper [11] has discovered to solve the bounded probabilistic reachability problem using bounded reachability of non-probabilistic hybrid systems [13, 14], it is only applicable for deterministic probabilistic choices. However, we study polyhedral probabilistic hybrid systems, where non-deterministic probabilistic choices have been considered. Although abstraction based methods overapproximate the maximum and minimum probability, we are interested in computing exact maximum/minimum probability of reachability.

## 2 Preliminaries

*Notations.* We use  $\mathbb{R}$ ,  $\mathbb{R}_{\geq 0}$  and  $\mathbb{N}$  to denote the real, non-negative real and natural numbers, respectively. We use  $[n]$  to denote the set  $\{1, 2, \dots, n\}$ . We use  $Id$  to represent the identity function, namely,  $Id : X \rightarrow X$  such that  $Id(x) = x$ . Symbol  $\circ$  represents concatenation of two sequences, that is, given two sequences  $\sigma_1 = s_0, s_1, \dots, s_n$  and  $\sigma_2 = s'_1, s'_2, \dots, s'_m$ ,  $\sigma_1 \circ \sigma_2 = s_0, s_1, \dots, s_n, s'_1, s'_2, \dots, s'_m$ . Given a countable set  $\mathcal{S}$  of real numbers,  $\sum \mathcal{S}$  denotes the sum of all elements in the set, that is,  $\sum \mathcal{S} = \sum_{s \in \mathcal{S}} s$ .

*Probability Distributions.* A probability distribution over a set  $\mathcal{S}$  is a function  $\rho : \mathcal{S} \rightarrow [0, 1]$  such that  $\sum_{s \in \mathcal{S}} \rho(s) = 1$ . We will assume that  $\rho$  has finite support, that is,  $\rho(s) \neq 0$  for only finitely many elements, hence, the summation over  $\mathcal{S}$  for the probability distribution  $\rho$  is valid. We use  $Dist(\mathcal{S})$  to denote the set of all probability distributions over the set  $\mathcal{S}$ .

*Polyhedra.* A set  $\mathcal{X} \subseteq \mathbb{R}^n$  is called an  $n$ -dimensional polyhedron if there exist matrices  $A, B$  such that  $\mathcal{X} = \{x \mid Ax \leq B\}$ .  $Poly(n)$  denotes the set of all polyhedra which are subsets of  $\mathbb{R}^n$ .

## 3 Markov Decision Processes

In this section, we define the class of timed Markov decision processes and a special subclass of the same called the timed Markov chains. The timed Markov decision processes are a rich class of Markov decision processes, where non-deterministic transitions appear not only from probabilistic distributions but also from the time variable.

**Definition 1 (Timed Markov Decision Process (TMDP)).** A TMDP is a structure  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ , where

- $\mathcal{S}$  is a set of states;
- $\longrightarrow \subseteq \mathcal{S} \times \mathbb{R}_{\geq 0} \times \text{Dist}(\mathcal{S})$  is a transition relation capturing a set of timed probabilistic edges.

We denote a timed probabilistic edge  $(s, t, \rho) \in \longrightarrow$  by  $s \xrightarrow{t} \rho$ . Note that a state may have multiple timed probabilistic edges associated with it, that is, distinct edges  $(s, t_1, \rho_1), (s, t_2, \rho_2) \in \longrightarrow$ , where  $t_1 \neq t_2$  or  $\rho_1 \neq \rho_2$ . Thus, the transition relation allows non-determinism.

Next, a path of TMDP  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$  is a finite sequence of states and times  $\sigma = s_0 t_1 s_1 t_2 s_2 \dots t_n s_n$  such that there exists a sequence of probability distributions  $\rho_1 \rho_2 \rho_3 \dots \rho_n$  which satisfy  $s_i \xrightarrow{t_{i+1}} \rho_{i+1}$  and  $\rho_{i+1}(s_{i+1}) > 0$  for  $0 \leq i \leq n-1$ .  $\sigma[i]$  denotes the  $i^{\text{th}}$  state of the path  $\sigma$ , that is,  $\sigma[i] = s_i$ ;  $\text{len}(\sigma)$  represents length of the path  $\sigma$ , that is,  $\text{len}(\sigma) = n$ , and  $L(\sigma)$  represents the last state of the path  $\sigma$ , that is,  $L(\sigma) = \sigma[\text{len}(\sigma)]$ . Also,  $\Delta(\sigma)$  denotes the set containing all prefixes of the path  $\sigma$ , that is,  $\Delta(\sigma) = \{\sigma' \mid \exists \sigma'', \sigma = \sigma' \circ \sigma''\}$ .  $\mathcal{D}(\sigma)$  denotes the duration of the path  $\sigma$ , that is,  $\mathcal{D}(\sigma) = \sum_{i=1}^{\text{len}(\sigma)} t_i$ .  $\text{Paths}(\mathcal{T})$  denotes the set of all paths of  $\mathcal{T}$ , and  $\text{Paths}_k(\mathcal{T}, s, \mathbb{F}) = \{\sigma \in \text{Paths}(\mathcal{T}) \mid \text{len}(\sigma) = k, \sigma[0] = s, \sigma[k] \in \mathbb{F}, \text{ for } 0 \leq i < k, \sigma[i] \notin \mathbb{F}\}$ .  $\text{Paths}(\mathcal{T}, s, \mathbb{F})$  represents the set of all paths that start at state  $s$  and end at some state  $t$  in  $\mathbb{F}$  and no states in between  $s$  and  $t$  are in  $\mathbb{F}$ , that is,  $\text{Paths}(\mathcal{T}, s, \mathbb{F}) = \bigcup_k \text{Paths}_k(\mathcal{T}, s, \mathbb{F})$ .

Furthermore, we define a special subclass of TMDP, where there will be a unique probability distribution associated with each state.

**Definition 2 (Timed Markov Chain (TMC)).** A TMC is a TMDP  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$  where the following condition holds:

- For each state  $s \in \mathcal{S}$  if there exist  $s \xrightarrow{t_1} \rho_1, s \xrightarrow{t_2} \rho_2$ , then  $t_1 = t_2$  and  $\rho_1 = \rho_2$ .

Since TMC allows only one timed probabilistic edge for each state, we can define a probabilistic transition function  $Pr : \mathcal{S} \times \mathcal{S} \rightarrow [0, 1]$ , where  $Pr(s, s') = \rho(s')$  if there exists  $s \xrightarrow{t} \rho$ .

Next, we explain how to resolve the non-determinism in a TMDP  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$  and obtain a TMC. First, we define a *scheduling function* to be a partial function  $\gamma : \text{Paths}(\mathcal{T}) \rightarrow \mathbb{R}_{\geq 0} \times \text{Dist}(\mathcal{S})$  such that if  $\gamma(\sigma) = (t, \pi)$ , then  $L(\sigma) \xrightarrow{t} \pi$ . Let  $\Gamma(\mathcal{T})$  denotes the set of all scheduling functions for a TMDP  $\mathcal{T}$ .

**Definition 3.** Given a TMDP  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$  and a scheduling function  $\gamma$ , we obtain a TMC  $\mathcal{T}_\gamma = (\mathcal{S}_\gamma, \longrightarrow_\gamma)$ , where  $\mathcal{S}_\gamma = \text{Paths}(\mathcal{T})$ ; for each path  $s_\gamma \in \text{Paths}(\mathcal{T})$ ,  $s_\gamma \xrightarrow{t}_\gamma \pi_\gamma$  if  $\exists \pi$  such that  $L(s_\gamma) \xrightarrow{t} \pi$  and for all  $s \in \mathcal{S}$ ,  $\pi_\gamma(s_\gamma t s) = \pi(s)$ .

Let us consider a TMC  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ . Let  $\mathcal{P}_{\mathcal{T}}(\sigma)$  denote the probability associated with a path  $\sigma$  in  $\mathcal{T}$ ; we will drop the subscript  $\mathcal{T}$  when it is clear from the context. We define  $\mathcal{P}(\sigma)$  inductively as follows. If  $len(\sigma) = 0$ , then  $\mathcal{P}(\sigma) = 1$ .

$$\mathcal{P}(\sigma) = \prod_{i=1}^{len(\sigma)} Pr(\sigma[i-1], \sigma[i]).$$

The probability of reaching a target set  $\mathbb{F}$  from a state  $s$  with path length exactly  $k$  and at most  $k$  within time  $T$ , respectively, are defined as,

$$\mathcal{P}_{(T,=k,\mathbb{F})}(\mathcal{T}, s) = \sum \{ \mathcal{P}(\sigma) \mid \sigma \in Paths_k(\mathcal{T}, s, \mathbb{F}), \mathcal{D}(\sigma) \leq T \},$$

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}(\mathcal{T}, s) = \sum_{i=0}^k \mathcal{P}_{(T,=i,\mathbb{F})}(\mathcal{T}, s).$$

For TMDP  $\mathcal{T} = (\mathcal{S}, \longrightarrow)$ , we define the maximum and minimum probability of reaching a target set  $\mathbb{F}$  from a state  $s$  with path length at most  $k$  within time  $T$ , respectively, to be

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}^{sup}(\mathcal{T}, s) = \sup_{\gamma \in \Gamma(\mathcal{T})} \mathcal{P}_{(T,\leq k,\mathbb{F})}(\mathcal{T}_{\gamma}, s),$$

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}^{inf}(\mathcal{T}, s) = \inf_{\gamma \in \Gamma(\mathcal{T})} \mathcal{P}_{(T,\leq k,\mathbb{F})}(\mathcal{T}_{\gamma}, s).$$

### 3.1 Inductive Definition of Probabilistic Reachability

In this section, we provide an alternate inductive definition for both  $\mathcal{P}_{(T,\leq k,\mathbb{F})}^{inf}(\mathcal{T}, s)$  and  $\mathcal{P}_{(T,\leq k,\mathbb{F})}^{sup}(\mathcal{T}, s)$ .

– Base case:

$$\mathcal{P}_{(T,\leq 0,\mathbb{F})}^{inf}(\mathcal{T}, s) = \mathcal{P}_{(T,\leq 0,\mathbb{F})}^{sup}(\mathcal{T}, s) = \begin{cases} 1, & \text{if } s \in \mathbb{F} \\ 0, & \text{otherwise} \end{cases}$$

– Induction step:

If  $s \in \mathbb{F}$ , then

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}^{inf}(\mathcal{T}, s) = \mathcal{P}_{(T,\leq k,\mathbb{F})}^{sup}(\mathcal{T}, s) = 1.$$

Otherwise,

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}^{inf}(\mathcal{T}, s) = \inf_{s \xrightarrow{t} \rho} \left( \sum_{s' \in \mathcal{S}} \rho(s') \mathcal{P}_{(T,\leq k-1,\mathbb{F})}^{inf}(\mathcal{T}, s') \right);$$

$$\mathcal{P}_{(T,\leq k,\mathbb{F})}^{sup}(\mathcal{T}, s) = \sup_{s \xrightarrow{t} \rho} \left( \sum_{s' \in \mathcal{S}} \rho(s') \mathcal{P}_{(T,\leq k-1,\mathbb{F})}^{sup}(\mathcal{T}, s') \right).$$

## 4 Probabilistic Hybrid Systems

In this section, we introduce the class of probabilistic hybrid systems and provide their formal definition and semantics. In addition, we introduce a certain subclass of probabilistic hybrid systems that we study in this paper.

### 4.1 Syntax

Probabilistic hybrid systems capture the discrete, continuous and probabilistic behaviors. The continuous behaviors are captured using differential equations, while the discrete and probabilistic behaviors are captured using probabilistic edges, guards, and resets. Next, we introduce the syntax of probabilistic hybrid systems.

**Definition 4 (Probabilistic Hybrid Systems).** *A probabilistic hybrid system (PHS) is a tuple  $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathcal{Q}_0, \mathcal{X}_0, Inv, Flow, Edges, Guard, Reset, Init)$ , where*

- $\mathcal{Q}$  is a set of locations;
- $\mathcal{X} \subseteq \mathbb{R}^n$  is a continuous state space;
- $\mathcal{Q}_0 \subseteq \mathcal{Q}$  is a set of initial locations;
- $\mathcal{X}_0 \subseteq \mathcal{X}$  is a countable set of initial continuous states;
- $Inv : \mathcal{Q} \rightarrow 2^{\mathcal{X}}$  is an invariant function;
- $Flow : \mathcal{Q} \times \mathcal{X} \rightarrow 2^{\mathcal{X}}$  is a flow function which assigns a vector to each state  $(q, x) \in \mathcal{Q} \times \mathcal{X}$ ;
- $Edges \subseteq \mathcal{Q} \times Dist(\mathcal{Q})$  is a finite set of probabilistic edges;
- $Guard : Edges \rightarrow 2^{\mathcal{X}}$  is a guard function;
- $Reset : Edges \times \mathcal{Q} \times \mathcal{X} \rightarrow \mathcal{X}$  is a reset function;
- $Init : \mathcal{Q}_0 \times \mathcal{X}_0 \rightarrow [0, 1]$  is an initial probability distribution over  $\mathcal{Q}_0, \mathcal{X}_0$ ,  
 $\sum_{q_0 \in \mathcal{Q}_0} \sum_{x_0 \in \mathcal{X}_0} Init(q_0, x_0) = 1$ .

*Notation:* Given a PHS  $\mathcal{H}$ , we will represent its elements using  $\mathcal{H}$  as a subscript. For instance, the invariant and flow functions of  $\mathcal{H}$ , are represented as  $Inv_{\mathcal{H}}$  and  $Flow_{\mathcal{H}}$ , respectively.

### 4.2 Semantics

Next, we describe the semantics of PHS in terms of an infinite state timed markov decision process. A state of the PHS is a pair  $(q, x)$ , where  $q \in \mathcal{Q}$  is a discrete location, and  $x \in \mathcal{X}$  is a continuous state. A timed probabilistic edge associated with a state  $(q, x)$  consists of a time  $T$  elapse in which the state  $x$  evolves according to the dynamics to some state  $x'$  and then  $x'$  transitions instantaneously to other states governed by guards and resets. More precisely, a continuous transition from a state  $(q, x)$  to a state  $(q, x')$  is possible in time  $T$  if there exists a function  $\phi : [0, T] \rightarrow \mathcal{X}$  such that  $\phi(0) = x$ ,  $\phi(T) = x'$ ,

$\frac{d\phi(t)}{dt} \in \text{Flow}((q, \phi(t)))$  and  $\phi(t) \in \text{Inv}(q)$  for  $0 \leq t \leq T$ . A probabilistic transition from a state  $(q, x)$  to a state  $(q', x')$  with probability  $p$  is possible if there exists an edge  $(q, \rho) \in \text{Edges}$  such that  $x \in \text{Guard}((q, \rho))$ ,  $x' = \text{Reset}((q, \rho), q', x)$  and  $\rho(q') = p$ .

**Definition 5.** Given PHS  $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathcal{Q}_0, \mathcal{X}_0, \text{Inv}, \text{Flow}, \text{Edges}, \text{Guard}, \text{Reset}, \text{Init})$ , the semantics of  $\mathcal{H}$  is defined as TMDP  $\llbracket \mathcal{H} \rrbracket = (\mathcal{S}, \longrightarrow_{\llbracket \mathcal{H} \rrbracket})$ , where

1.  $\mathcal{S} = \mathcal{Q} \times \mathcal{X}$ ;
2.  $((q, x), T, \pi) \in \longrightarrow_{\llbracket \mathcal{H} \rrbracket}$  if  $\exists (q, \rho) \in \text{Edges}$  and  $\exists \phi : [0, T] \rightarrow \mathcal{X}$  such that
  - (a)  $\phi(0) = x$  and  $\phi(T) \in \text{Guard}((q, \rho))$ ;
  - (b)  $\phi(t) \in \text{Inv}(q)$  and  $\frac{d\phi(t)}{dt} \in \text{Flow}((q, \phi(t)))$  for all  $t \in [0, T]$ ;
  - (c) For each  $(q', x') \in \mathcal{Q} \times \mathcal{X}$ ,  $\pi((q', x')) = \rho(q')$  if  $x' = \text{Reset}((q, \rho), q', \phi(T))$  else  $\pi((q', x')) = 0$ .

The TMDP has an infinite number of states because each state is a pair of discrete location and a continuous value, where the number of values of the continuous variables is infinite. Note that although we have an infinite number of states, for every timed probabilistic edge  $(q, x) \xrightarrow{T}_{\llbracket \mathcal{H} \rrbracket} \pi$ ,  $\pi$  has finite support.

### 4.3 Subclasses of Probabilistic Hybrid Systems

Next, we define a subclass of probabilistic hybrid systems, called polyhedral probabilistic hybrid systems. In this subclass, invariant, guard and reset functions are provided by certain linear constraints. More precisely, we have

- C1  $\text{Inv} : \mathcal{Q} \rightarrow \text{Poly}(n)$ ;  
 C2  $\text{Guard} : \text{Edges} \rightarrow \text{Poly}(n)$ ;  
 C3  $\text{Reset}(e, (q, x)) = A_e x + B_e$  for some square matrix  $A_e$  and constant vector  $B_e$ ;

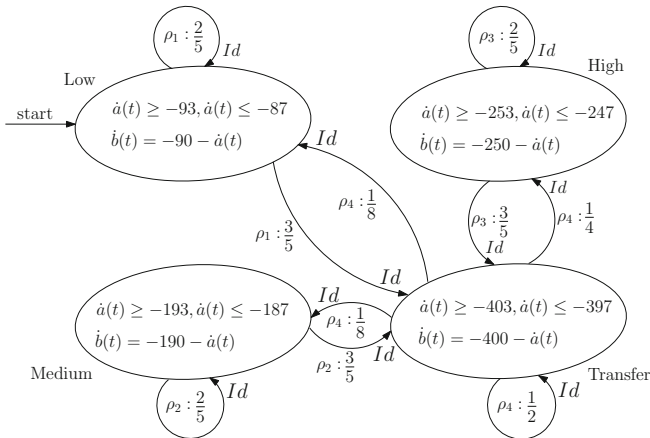
**Definition 6 (Polyhedral Probabilistic Hybrid Systems).** A polyhedral probabilistic hybrid system is a PHS  $\mathcal{H}$ , where the invariant, guard and reset function are as in [C1-C3] and the flow function  $\text{Flow}_{\mathcal{H}}$  is given by  $\text{Flow}_{\mathcal{H}}((q, x)) = P_q$ , where  $P_q$  is a polyhedron in  $\text{Poly}(n)$ , that depends only on  $q$ .

*Example 1.* Consider the PHS with four discrete locations corresponding to the kinetic battery model in a nano satellite given in the paper [18], which is shown in Fig. 1. It has two continuous variables represented by  $a$  and  $b$ . Formally, it can be modeled as a polyhedral PHS  $\mathcal{H} = (\mathcal{Q}, \mathcal{X}, \mathcal{Q}_0, \mathcal{X}_0, \text{Inv}, \text{Flow}, \text{Edges}, \text{Guard}, \text{Reset}, \text{Init})$ , where

- $\mathcal{Q} = \{\text{Low}, \text{Medium}, \text{High}, \text{Transfer}\}$ ;
- $\mathcal{X} = [-100, 2500] \times [-100, 2500]$ ;  $\mathcal{Q}_0 = \{\text{Low}\}$ ;
- $\mathcal{X}_0 = \{(2500, 2500)\}$ ;  $\text{Inv}(q) = \mathcal{X}$  for all  $q$ ;

- The flow function which is given by  $Flow(q, x) = P_q$ , where:  $P_{\text{Low}} = \{(\dot{a}, \dot{b}) \mid -93 \leq \dot{a} \leq -87, \dot{b} = -9 - \dot{a}\}$ ;  $P_{\text{Medium}} = \{(\dot{a}, \dot{b}) \mid -193 \leq \dot{a} \leq -187, \dot{b} = -190 - \dot{a}\}$ ;  $P_{\text{High}} = \{(\dot{a}, \dot{b}) \mid -253 \leq \dot{a} \leq -247, \dot{b} = -250 - \dot{a}\}$ ;  $P_{\text{Transfer}} = \{(\dot{a}, \dot{b}) \mid -403 \leq \dot{a} \leq -397, \dot{b} = -400 - \dot{a}\}$ .
- $Edges = \{\text{Low}, \rho_1\}, (\text{Medium}, \rho_2), (\text{High}, \rho_1), (\text{Transfer}, \rho_1)\}$ , where  $\rho_1, \rho_2, \rho_3, \rho_4 \in \text{Dist}(\mathcal{Q})$ , and  $\rho_1(\text{Low}) = \rho_2(\text{Medium}) = \rho_3(\text{High}) = \frac{2}{5}$ ;  $\rho_1(\text{Transfer}) = \rho_2(\text{Transfer}) = \rho_3(\text{Transfer}) = \frac{3}{5}$ ;  $\rho_4(\text{Low}) = \rho_4(\text{Medium}) = \frac{1}{8}, \rho_4(\text{High}) = \frac{1}{4}$ ;  $\rho_4(\text{Transfer}) = \frac{1}{2}$ ;
- $Guard(e) = \mathbb{R}^2$  for all  $e \in Edges$ ;
- All resets are identity functions, that is,  $Reset(\rho, q) = Id$  for all  $\rho$  and  $q$ ;
- $Init((\text{Low}, (2500, 2500))) = 1$ .

Next, we construct a formula that computes the exact maximum and minimum probability of reachability in a polyhedral PHS.

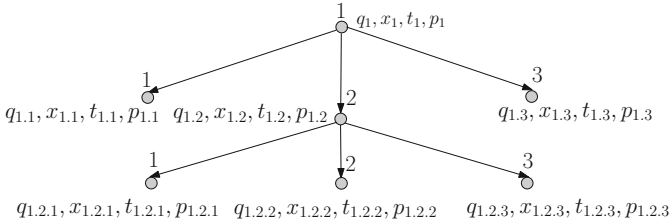


**Fig. 1.** Discharging of Kinetic Battery in Satellite GOMX-1

## 5 Computing Probability of Reachability

Our main problem is to compute the probability of reaching a target set  $\mathbb{F}$  within  $k$  discrete transitions and time  $T$  in a polyhedral probabilistic hybrid system. Our broad approach consists of reducing the problem of computing the minimum and maximum probability of reachability in a polyhedral PHS into two optimization problems with constraints expressed using a satisfiability modulo theory formula, which encodes the computation trees of the polyhedral PHS. From the inductive definition of the probability of reachability, we are required to unroll the polyhedral PHS for  $k$  steps to construct a tree with  $k$  levels (height  $k$ ). The minimum/maximum probability of reachability at each node is expressed iteratively as a solution of an optimization problem over constraints which themselves recursively contain other optimization problems. Note that all the entities

of a polyhedral PHS such as invariants, dynamics, guards, and resets, can be expressed as linear constraints. However, we have non-deterministic probabilistic edges, hence, each recursive call to the optimization problems are in the form of a linear optimization problem subject to constraints consisting of conjunctions and disjunctions of linear constraints. Though theoretically this problem can be solved by encoding it in first-order logic, we do not know of any tool that efficiently solves optimization problems of this kind. Our broad idea is to lift the recursive optimization problem at each node of the computation tree up to the root level, that is, develop an encoding that solves a single optimization problem at the root of the computation tree. Linear optimization problems over conjunctions and disjunctions of linear constraints can be efficiently solved using the tools Z3opt SMT- solver [5] and SYMBA [20], which add optimization capabilities to SMT solving.



**Fig. 2.** Illustration of computation tree for  $k = 2$

Next, we explain the construction of the encoding of the computation, which will be the important part of the optimization problem we formulate. We fix the following ordering on the locations of  $\mathcal{H}$ , namely,  $q_1, \dots, q_n$  and we assume that  $q_1$  is the initial state we are interested. Let us fix a polyhedral PHS  $\mathcal{H}$ . Let  $n$  be the number of locations in  $\mathcal{H}$ . A computation tree of level  $k$  is essentially an  $n$ -ary tree of height  $k$  as shown in Fig. 2. We define the names of the node in the tree as follows. The name of the root node is 1. Inductively, the names of the  $i$ -th child of a node named  $\alpha$  are  $\alpha.i$ , where  $i$  ranges over 1 to  $n$ . Hence, the node 1.2.1 refers to the first child of the second child of the root node. We annotate the tree with states reached along an unrolling of  $k$  steps of  $\llbracket \mathcal{H} \rrbracket$ . We annotate the root with an initial state of  $\llbracket \mathcal{H} \rrbracket$ . The children of a node  $\alpha$  are annotated by the states reached by taking a timed probabilistic edge of  $\llbracket \mathcal{H} \rrbracket$ . Note that in each timed probabilistic edge  $((q, x), t, \pi)$  of  $\llbracket \mathcal{H} \rrbracket$ ,  $\pi$  has finite support; more importantly, for each  $q'$ ,  $\pi(q', x') \neq 0$  for at most one  $x'$ . Hence, we will fix an ordering of the locations  $\mathcal{Q}_{\mathcal{H}}$  and assume that the  $i$ -th child is annotated by a state whose location is the  $i$ -location in the ordering, and the  $i$ -th continuous state is given by the reset function corresponding to the edge taken and the  $i$ -th location (target). (Recall  $Reset : Edges \times \mathcal{Q} \times \mathcal{X} \rightarrow \mathcal{X}$ , where  $\mathcal{Q}$  captures the target location.) Next, we describe the SMT formula that encodes the computation tree.

First, we fix some notations that will be used in the rest of the section. Let  $\mathcal{I}_q(x)$  be a predicate corresponding to  $Inv(q)$ , that is,  $\mathcal{I}_q(x)$  evaluates to



true if and only if  $x \in \text{Inv}(q)$ . Similarly,  $\mathcal{F}_q(x, r)$ ,  $\mathcal{G}_{(q, \rho)}(x)$ , and  $\mathcal{R}_{(q, \rho, q')}(x, x')$  be the predicates that capture  $r \in \text{Flow}(q, x)$ ,  $x \in \text{Guard}(q, \rho)$ , and  $x' = \text{Reset}((q, \rho), q', x')$ , respectively. Also, let  $\mathbb{F}_q(x)$  be a predicate for  $(q, x) \in \mathbb{F}$ . First, we explain how to capture the discrete, continuous and timed probabilistic edges of  $\llbracket \mathcal{H} \rrbracket$  using first-order formulas with only existential quantification, conjunctions and disjunctions, that is, as a satisfiability modulo theory (SMT) formula.

*Continuous transitions.* We construct a formula  $\text{Cont}_q(x, t, x')$  that encodes whether there exists an execution that starts from the state  $(q, x)$  and reaches the state  $(q, x')$  at time  $t$ .

$$\text{Cont}_q(x, t, x') = \exists r, \mathcal{F}_q(x, r) \wedge x' = x + rt \wedge \mathcal{I}_q(x) \wedge \mathcal{I}_q(x')$$

Note that if  $\mathcal{H}$  is a polyhedral PHS, then all the constraints in  $\text{Cont}_q(x, t, x')$  will be linear expressions except for the multiplication of *rate* and *time*, that is,  $rt$ . We can eliminate the nonlinear expression  $rt$  by converting it into an equivalent linear expression. From the definition of polyhedral PHS, the predicates  $\mathcal{F}_q(x, r)$  and  $\mathcal{I}_q(x)$  can be expressed as the conjunctions of linear constraints of the form  $a \cdot r \leq b$ , where  $a$  is a constant  $n$  dimensional vector and  $b$  is a constant number. We introduce a new variable (vector)  $y = rt$  (note  $r$  is a vector and  $t$  is a scalar), and then replace all linear constraints  $a \cdot r \leq b$  in  $\mathcal{F}_q(x, r)$  by the linear constraints  $a \cdot y \leq b \cdot t$ , and the constraint  $x' = x + rt$  by  $x' = x + y$ . The two constraints are equivalent, since we will have assumed that the domain of  $t$  is the non-negative real numbers.

*Discrete probabilistic transitions.* We construct a formula  $\text{Disc}_q(x, \bar{x}, \bar{p})$  that encodes the distribution over the states reached by taking some probabilistic edge  $\rho$  from the state  $(q, x)$ . If no such edge exists, we allow a dummy transition with 0 as the probabilities. The  $(q, \rho)$ 's range over  $\text{Edges}$ .

$$\begin{aligned} \text{Disc}_q(x, \bar{x}, \bar{p}) = & \left[ \left( \bigwedge_{(q, \rho)} \neg \mathcal{G}_{(q, \rho)}(x) \wedge \bigwedge_{j=1}^n p_j = 0 \right) \vee \right. \\ & \left. \bigvee_{(q, \rho)} [\mathcal{G}_{(q, \rho)}(x) \wedge \bigwedge_{j=1}^n (\mathcal{R}_{(q, \rho, q_j)}(x, x_j) \wedge p_j = \rho(q_j)) \right] \end{aligned}$$

*Transition relation.* The formula  $\text{Trans}_q(x, t, x', \bar{x}, \bar{p})$  encodes a timed probabilistic edge, that is, a combination of a continuous transition followed by a discrete probabilistic transition.

$$\text{Trans}_q(x, t, x', \bar{x}, \bar{p}) = [\exists x', \text{Cont}_q(x, t, x') \wedge \text{Disc}_q(x', \bar{x}, \bar{p})]$$

We will use  $\text{Trans}$  as a primitive to encode computation tree of  $\mathcal{H}$ . We need variables to capture different entities at each of the nodes of the computation

tree. Hence, we introduce some notation. Let  $I_k = \{\alpha \mid \alpha = 1.i_1.i_2.\dots.i_m, 0 \leq m \leq k, i_j \in [n] \text{ for } 1 \leq j \leq m\}$  denote the set of all paths starting from root node in the computation tree. Note that the location at node  $\alpha$  is  $L(\alpha)$ . Given a variable  $z$ ,  $\mathcal{V}_z^k$  will denote a set of variables for each node in the tree corresponding to  $z$ , that is,  $\mathcal{V}_z^k = \{z_\alpha \mid \alpha \in I_k\}$ . The free variables of our formula will include  $\mathcal{V}_x^k$ ,  $\mathcal{V}_t^k$ , and  $\mathcal{V}_p^k$ , where  $x_\alpha$  denotes the continuous state in the computation tree at  $\alpha$ ; similarly,  $t_\alpha$  denotes the time spent at location  $q_{L(\alpha)}$ , and  $p_{\alpha.i}$  denotes the probability of the transition from  $(q_{L(\alpha)}, x_\alpha)$  to  $(q_i, x_{\alpha.i})$  in the computation tree. We will assume that  $p_1 = \text{Init}(q_1, x_1)$ .

We need to ensure that the total time spent on the executions is bounded by some value  $T$ . Hence, we track the total time spent along any path from the root in the computation tree using the variables in  $\mathcal{V}_T$ . Our goal is to optimize the probability of reaching a final state. Hence, we use the variables in  $\mathcal{V}_P$  to capture the probabilities along the paths of the computation tree, that is,  $P_\alpha$  captures the probability associated with execution corresponding to  $\alpha$ . Finally, we need to add the probabilities of all those paths that end in a final state and don't reach a final state before that. Hence, we have boolean variable sets  $\mathcal{V}_F$  and  $\mathcal{V}_B$ , where  $F_\alpha$  is a boolean variable that is true when  $\alpha$  corresponds to an execution that ends in a final state without having visited a final state before, and  $B_\alpha$  is a boolean variable that is true if a final state has been visited along  $\alpha$ .

Next, we construct a formula  $\text{Exec}^{\leq k, T, \mathbb{F}}$  that captures the sum of the probabilities of the executions of computation tree with level  $k$  that reaches the target set  $\mathbb{F}$  (for the first time) within time  $T$ .

*Validation of tree edges.* The formula  $\text{Tree}$  validates all the edges in the computation tree for given values of  $\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p$ . It also checks that the initial state and probability (at the root) are valid. Here  $\mathbb{I}_q(x, p)$  is a predicate such that  $\text{Init}(q, x) = p$ . We use  $z_{\bar{\alpha}}$  to denote  $(z_{\alpha.1}, \dots, z_{\alpha.n})$ .

$$\text{Tree}(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p) = \mathbb{I}_{q_1}(x_1, p_1) \wedge \bigwedge_{\alpha \in I_{k-1}} [\text{Trans}_{q_{L(\alpha)}}(x_\alpha, t_\alpha, \bar{x}_\alpha)]$$

*Timing constraints.* The formula  $\text{Time}_T$  checks the relation between global times in  $\mathcal{V}_T$  and local times in  $\mathcal{V}_t$ , and ensures that the total times are less than  $T$  along any executions.

$$\text{Time}_T(\mathcal{V}_T, \mathcal{V}_t) = [(T_1 = 0) \wedge \bigwedge_{\alpha \in I_{k-1}} \left( \bigwedge_{j=1}^n T_{\alpha.j} = T_\alpha + t_\alpha \right) \wedge \bigwedge_{\alpha \in I_k} (0 \leq T_\alpha \leq T)]$$

*Probability checking.* The formula  $\text{Prob}$  checks the relation between the total probability along a path captured using  $\mathcal{V}_P$  and the local probabilities along the individual transitions captured using  $\mathcal{V}_p$ .

$$\text{Prob}(\mathcal{V}_P, \mathcal{V}_p) = [P_1 = p_1 \wedge \bigwedge_{\alpha \in I_{k-1}} \left( \bigwedge_{j=1}^n P_{\alpha.j} = P_\alpha * p_{\alpha.j} \right)]$$

*Checking if final state has been reached.* The formula *Before* captures using  $\mathcal{V}_B$  whether a state from the target set has been seen at any previous node in the path from root node to that node in the execution of computation tree using the values in  $\mathcal{V}_x$ .

$$Before(\mathcal{V}_B, \mathcal{V}_x) = [(B_1 = \mathbb{F}_{q_1}(x_1)) \wedge \bigwedge_{\alpha \in I_{k-1}} \left( \bigwedge_{j=1}^n B_{\alpha,j} = (B_\alpha \vee \mathbb{F}_{q_{L(\alpha)}}(x_\alpha)) \right)]$$

*Final target state checking.* The formula *Final* captures using  $\mathcal{V}_F$  whether a target set is visited for the first time at a node in a path from the root node to that node in the execution of the computation tree using the values in  $\mathcal{V}_B$ .

$$Final(\mathcal{V}_F, \mathcal{V}_B) = [(F_1 = B_1) \wedge \bigwedge_{\alpha \in I_{k-1}} \left( \bigwedge_{j=1}^n (F_{\alpha,j} = (\neg B_\alpha \wedge B_{\alpha,j})) \right)]$$

*Sum of probabilities.* The formula *Sum* checks whether given  $\mathcal{V}_P, \mathcal{V}_F$  and  $p_s$ , if  $p_s$  is the sum of all the probabilities associated with nodes in the computation tree which correspond to executions that reach the final state only in the last state.

$$Sum(\mathcal{V}_P, \mathcal{V}_F, p_s) = [p_s = \left( \sum_{\alpha \in I_k} F_\alpha P_\alpha \right)]$$

Finally, we can put all the formulas defined above to construct the formula  $Exec^{\leq k, T, \mathbb{F}}(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p, \mathcal{V}_T, \mathcal{V}_P, \mathcal{V}_B, \mathcal{V}_F, p_s)$  that captures the values of the variable  $\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p, \mathcal{V}_T, \mathcal{V}_P, \mathcal{V}_B$  and  $\mathcal{V}_F$  for a computation tree, along with the total probability of reaching the target set  $\mathbb{F}$  for this computation tree in  $p_s$ .

$$Exec^{\leq k, T, \mathbb{F}}(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p, \mathcal{V}_T, \mathcal{V}_P, \mathcal{V}_B, \mathcal{V}_F, p_s) = Tree(\mathcal{V}_x, \mathcal{V}_t, \mathcal{V}_p) \wedge Time(\mathcal{V}_T, \mathcal{V}_t) \wedge Prob(\mathcal{V}_P, \mathcal{V}_p) \wedge Before(\mathcal{V}_B, \mathcal{V}_x) \wedge Final(\mathcal{V}_F, \mathcal{V}_B) \wedge Sum(\mathcal{V}_P, \mathcal{V}_F, p_s)$$

Note that the formula  $Exec^{\leq k, T, \mathbb{F}}$  does not check for the minimum/maximum probabilistic reachability. Therefore, we need to solve an optimization problem with  $p_s$  as the objective function over  $Exec^{\leq k, T, \mathbb{F}}$  as the constraints. Note that the formula  $Exec^{\leq k, T, \mathbb{F}}$  has conjunctions and disjunctions of linear expressions which can be solved by existing SMT optimization tools Z3opt SMT-solver [5] and SYMBA [20].

## 6 Experiments

In this section, we present an implementation details of the SMT based approach for computing the bound on the minimum/maximum probability of reachability in a polyhedral probabilistic hybrid system. Our implementation has probabilistic reachability analysis module that takes as input a polyhedral PHS, number of discrete transitions  $k$ , total time  $T$ , and a target set  $\mathbb{F}$ , and outputs an SMT

formula that captures all the computation trees corresponding to the polyhedral PHS. The latter module calls either Z3opt or SYMBA to solve the optimization problems over the SMT formula that returns the minimum/maximum probability of reachability. Next, We present the analysis of a kinetic battery in a nano satellite introduced in [18]. All the computation times are measured in seconds, Unknown shows that either Z3opt or SYMBA do not terminate in 5 min. The evaluation of the experiment has been conducted on Ubuntu 14.04 OS, Intel ® Pentium(R) CPU B960 2.20 GHz  $\times$  2 Processor, 4 GB RAM.

Next, we show the results of the analysis of the case study using our approach. We compute the maximum probability of reachability for different values of  $T, k, \mathbb{F}$ . The results are summarized in Tables 1, 2 and 3. In Tables 1 and 2, we do not restrict the time spend at a location, however, in Table 3, we restrict the time that should be spent at a location. In the tables, Prob denotes the maximum probability of reachability, and  $Z_T$ , and  $S_T$  denote the time to solve the optimization problem over SMT formulas by the tools Z3opt and SYMBA, respectively. First, we consider the maximum probability of reaching the location Transfer starting from the location Low within time  $T = 7$  for different number of discrete transitions  $k$ . The results are tabulated in Table 1.

**Table 1.**  $T = 7$ , Initial state = (Low, (2500,2500))

Row	$k$	Prob	$Z_T$ (sec.)	$S_T$ (sec.)
1	1	3/5	0.0104	0.0846
2	2	21/25	0.0192	0.2395
3	3	117/125	0.0446	2.9072
4	4	609/625	0.1731	34.725
5	5	3093/3125	0.7303	Unknown

Note that the maximum probability of reachability increases as we increase the number of discrete transitions, as expected. Next, Z3opt solves the optimization on the SMT formula in much less time than SYMBA on all the examples. For instance, Z3opt solves the SMT formula in 0.7303 s for  $k = 5$ , however, SYMBA does not terminate within 5 min for the same SMT formula. Hence, we use Z3opt for further experiments.

In Table 2, we observe that the maximum probability of reaching a state where the charge is depleted is 0 within time  $T = 5$  and discrete transitions  $k = 2, 3, 4$ . Even in the location Transfer, which has a large discharge rate, namely, between 397 and 403, 5 units of time is not enough to drive the available charge to less than 0. Hence, the amount of charge never becomes less than 0 in any location within 5 units of time. However, 7 units of time are enough to deplete the available charge with different probabilities at different locations. In addition, the maximum probability increases as we increase the number of discrete transitions  $k$ . In rows 10, 11, 12, we get the same probability values as in

**Table 2.** Initial state = (Low, (2500, 2500))

Row	$k$	$Prob (T = 5)$	$Z_T$	$Prob (T = 7)$	$Z_T$
1	2	0	0.0142	21/25	0.0157
2	3	0	0.0758	117/125	0.1035
3	4	0	0.2302	609/625	0.6223
$\mathbb{F} = \{(Transfer, (a, b)) \mid a < 0\}$					
4	2	0	0.0160	3/20	0.0209
5	3	0	0.0904	57/200	0.1011
6	4	0	0.3508	399/1000	0.3735
$\mathbb{F} = \{(High, (a, b)) \mid a < 0\}$					
7	2	0	0.0298	3/40	0.0153
8	3	0	0.0859	57/400	0.1008
9	4	0	0.3547	1641/8000	0.3493
$\mathbb{F} = \{(Medium, (a, b)) \mid a < 0\}$					
10	2	0	0.0371	21/25	0.0415
11	3	0	0.1426	117/125	0.1479
12	4	0	0.9703	609/625	49.243
$\mathbb{F} = \{(loc, (a, b)) \mid a < 0, loc \in \{Low, Medium, Transfer, High\}\}$					

rows 1, 2, 3 because the maximum probability of depletion of the charge occurs at the location Transfer. Note that the experiments suggest that Z3opt efficiently solves the SMT formula constructed for the experiments.

Finally, in Table 3, the maximum probability increases as we increase time  $T$  because larger number of discrete transitions are enabled as we increase time  $T$ . Note that Z3opt solves the optimization over the SMT formula in this model with time constraint on the locations, faster than SMT formula constructed for the same value of  $k$  and target set in a model where we do not restrict the duration for which the system should be in a location.

**Table 3.**  $k = 4$ , Initial state = (Low, (25000, 25000)),  $\mathbb{F} = \{(loc, (a, b)) \mid a < 0, loc \in \{Low, Medium, Transfer, High\}\}$ 

Row	$T$	$Prob$	$Z_T$
1	100	0	10.432
3	300	117/125	7.8198
4	400	609/625	6.2379

## 7 Conclusions

In this paper, we have developed a method for computing the bound on the minimum/maximum probability of reachability in a polyhedral probabilistic hybrid system, where non-deterministic probabilistic transitions are allowed. This is not only a probabilistic reachability problem, but also an optimization problem. Our method exploits the recent advances of existing tool Z3opt to solve the bounded reachability problem. We have experimented with the method for the analysis of the depletion of the charge of a kinetic battery in a nano-satellite. The experimental results show that the method is efficient for solving the bounded reachability problem. Our future work will consist of extending the bounded reachability analysis to systems with complex non-linear dynamics, wherein in addition to non-determinism, we consider constraints involving rewards and gains.

**Acknowledgements.** Pavithra Prabhakar was partially supported by NSF CAREER Award No. 1552668 and ONR YIP Award No. N000141712577.

## References

1. Abate, A., Amin, S., Prandini, M., Lygeros, J., Sastry, S.: Computational approaches to reachability analysis of stochastic hybrid systems. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 4–17. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-71493-4\\_4](https://doi.org/10.1007/978-3-540-71493-4_4)
2. Abate, A., Prandini, M., Lygeros, J., Sastry, S.: An approximate dynamic programming approach to probabilistic reachability for stochastic hybrid systems. In: 47th IEEE Conference on Decision and Control, 2008. CDC 2008 (2008)
3. Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Probabilistic reachability and safety for controlled discrete time stochastic hybrid systems. *Automatica* **44**, 2724–2734 (2008)
4. Amin, S., Abate, A., Prandini, M., Lygeros, J., Sastry, S.: Reachability analysis for controlled discrete time stochastic hybrid systems. In: Hespanha, J.P., Tiwari, A. (eds.) HSCC 2006. LNCS, vol. 3927, pp. 49–63. Springer, Heidelberg (2006). [https://doi.org/10.1007/11730637\\_7](https://doi.org/10.1007/11730637_7)
5. Bjørner, N., Phan, A.-D., Fleckenstein, L.:  $\nu Z$  - an optimizing SMT solver. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 194–199. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46681-0\\_14](https://doi.org/10.1007/978-3-662-46681-0_14)
6. Blom, H.A.P., Bakker, G.J., Krystul, J.: Probabilistic reachability analysis for large scale stochastic hybrid systems. In: 2007 46th IEEE Conference on Decision and Control (2007)
7. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Reachability in recursive Markov decision processes. *Inf. Comput.* **206**, 520–537 (2008)
8. Bujorianu, M.L.: Extended stochastic hybrid systems and their reachability problem. In: Alur, R., Pappas, G.J. (eds.) HSCC 2004. LNCS, vol. 2993, pp. 234–249. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-24743-2\\_16](https://doi.org/10.1007/978-3-540-24743-2_16)
9. Bujorianu, M.L., Lygeros, J.: Toward a general theory of stochastic hybrid systems. In: Blom H.A.P., Lygeros J. (eds.) Stochastic Hybrid Systems. LNCS, vol. 337. Springer, Heidelberg (2006). [https://doi.org/10.1007/11587392\\_1](https://doi.org/10.1007/11587392_1)

10. D'Argenio, P.R., Jeannet, B., Jensen, H.E., Larsen, K.G.: Reachability analysis of probabilistic systems by successive refinements. In: de Alfaro, L., Gilmore, S. (eds.) PAPM-PROBMIV 2001. LNCS, vol. 2165, pp. 39–56. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-44804-7\\_3](https://doi.org/10.1007/3-540-44804-7_3)
11. Fränzle, M., Hermanns, H., Teige, T.: Stochastic satisfiability modulo theory: a novel technique for the analysis of probabilistic hybrid systems. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 172–186. Springer, Heidelberg (2008). [https://doi.org/10.1007/978-3-540-78929-1\\_13](https://doi.org/10.1007/978-3-540-78929-1_13)
12. Frehse, G.: PHAVer: algorithmic verification of hybrid systems past HyTech. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 258–273. Springer, Heidelberg (2005). [https://doi.org/10.1007/978-3-540-31954-2\\_17](https://doi.org/10.1007/978-3-540-31954-2_17)
13. Fränzle, M., Herde, C.: Efficient proof engines for bounded model checking of hybrid systems. *Electron. Notes Theor. Comput. Sci.* **133**, 119–137 (2005)
14. Gilles, A., Marco, B., Alessandro, C., Roberto, S.: Verifying industrial hybrid systems with mathSAT. *Electron. Notes Theor. Comput. Sci.* **119**, 17–32 (2005)
15. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-11439-2\\_10](https://doi.org/10.1007/978-3-319-11439-2_10)
16. Henzinger, T.A.: The theory of hybrid automata. In: *Proceedings of the Symposium on Logic in Computer Science* (1996)
17. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? *J. Comput. Syst. Sci.* **57**, 94–124 (1998)
18. Hermanns, H., Krčál, J., Nies, G.: Recharging probably keeps batteries alive. In: Berger, C., Mousavi, M.R. (eds.) CyPhy 2015. LNCS, vol. 9361, pp. 83–98. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-25141-7\\_7](https://doi.org/10.1007/978-3-319-25141-7_7)
19. Lafferriere, G., Pappas, G.J., Yovine, S.: A new class of decidable hybrid systems. In: Vaandrager, F.W., van Schuppen, J.H. (eds.) HSCC 1999. LNCS, vol. 1569, pp. 137–151. Springer, Heidelberg (1999). [https://doi.org/10.1007/3-540-48983-5\\_15](https://doi.org/10.1007/3-540-48983-5_15)
20. Li, Y., Albarghouthi, A., Kincaid, Z., Gurfinkel, A., Chechik, M.: Symbolic optimization with SMT solvers. In: *Symposium on Principles of Programming Languages*, POPL (2014)
21. Neuhauser, M.R., Zhang, L.: Time-bounded reachability probabilities in continuous-time Markov decision processes. In: *2010 Seventh International Conference on the Quantitative Evaluation of Systems (QEST)* (2010)
22. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation based abstraction refinement. In: Morari, M., Thiele, L. (eds.) HSCC 2005. LNCS, vol. 3414, pp. 573–589. Springer, Heidelberg (2005). [https://doi.org/10.1007/978-3-540-31954-2\\_37](https://doi.org/10.1007/978-3-540-31954-2_37)
23. Rutten, J.J.M.M., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical techniques for analyzing concurrent and probabilistic systems. *American Mathematical Society* (2004)
24. Shmarov, F., Zuliani, P.: Probreach: verified probabilistic delta-reachability for stochastic hybrid systems. In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control* (2015)
25. Sproston, J.: Decidable model checking of probabilistic hybrid automata. In: Joseph, M. (ed.) FTRTFT 2000. LNCS, vol. 1926, pp. 31–45. Springer, Heidelberg (2000). [https://doi.org/10.1007/3-540-45352-0\\_5](https://doi.org/10.1007/3-540-45352-0_5)
26. Summers, S., Lygeros, J.: Verification of discrete time stochastic hybrid systems: a stochastic reach-avoid decision problem. *Automatica* **46**, 1951–1961 (2010)

27. Wang, Q., Zuliani, P., Kong, S., Gao, S., Clarke, E.M.: Sreach: a probabilistic bounded delta-reachability analyzer for stochastic hybrid systems. In: Proceeding of the Computational Methods in Systems Biology (2015)
28. Wu, D., Koutsoukos, X.: Reachability analysis of uncertain systems using bounded-parameter Markov decision processes. *Artif. Intell.* **172**, 945–354 (2008)
29. Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.M.: Safety verification for probabilistic hybrid systems. *Eur. J. Control* **18**, 572–587 (2012)
30. Zhang, W., Prabhakar, P., Natarajan, B.: Abstraction based reachability analysis for finite branching stochastic hybrid systems. In: ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS) (2017)





# Control and Optimization of the SRPT Service Policy by Frequency Scaling

Andrea Marin<sup>1</sup>(✉), Isi Mitrani<sup>2</sup>, Maryam Elahi<sup>3</sup>, and Carey Williamson<sup>3</sup>

<sup>1</sup> DAIS, Università Ca' Foscari Venezia, Venice, Italy  
marin@unive.it

<sup>2</sup> School of Computing, Newcastle University, Newcastle, UK  
isi.mitrani@newcastle.ac.uk

<sup>3</sup> Department of Computer Science, University of Calgary, Calgary, Canada  
{bmelahi, carey}@ucalgary.ca

**Abstract.** In this paper, we study a system where the speed of a processor depends on the current number of jobs. We propose a queueing model in which jobs consist of a variable number of tasks, and priority is given to the job with the fewest remaining tasks. The number of processor frequency levels determines the dimensionality of the queueing process. The objective is to evaluate the trade-offs between holding cost and energy cost when setting the processor frequency. We obtain exact results for two and three frequency levels, and accurate approximations that can be further generalized. Numerical and simulation results show the high accuracy of the approximate solutions that we propose. Our experiments suggest that a parsimonious model with only two frequency levels is sufficient, since more elaborate models provide negligible improvements when optimizing the system.

## 1 Introduction

In dynamic speed scaling systems, the speed at which the processor executes jobs is adjusted dynamically based on the workload experienced by the system. Modern processors typically support over a dozen discrete operating speeds, often with a factor of two (or more) between the slowest and the fastest speeds available.

Multiple tradeoffs exist in such speed scaling systems. The most obvious is the tradeoff between response time and energy consumption (see. e.g., [4, 13, 17]). To minimize response time, one would use the highest system speed available, while to minimize energy consumption, one would use the lowest system speed. For this reason, most speed scaling research uses a cost function that combines response time (i.e., job delay, or holding cost) and energy consumption when doing system optimization as in [16]. Another interesting tradeoff arises from the interaction between the job scheduling policy and the speed scaling function. In job-count-based speed scaling, for instance, the CPU speed is set dynamically based on the current number of jobs in the system. As a result, different scheduling policies produce different costs, since the average number of jobs in the system varies.

For example, Shortest Remaining Processing Time (SRPT) minimizes system occupancy, and thus tends to run at lower speeds and for longer times than other scheduling policies, such as Processor Sharing (PS) [1].

Another consequence of SRPT-based scheduling is extreme unfairness to large jobs. There are two underlying reasons for this unfairness. First, large jobs tend to wait much longer to receive service, because of SRPT's bias toward short jobs. Second, even when they do receive service, large jobs tend to be served at low(er) speeds, since there are usually very few jobs (perhaps only one) in the system at that point [14].

Size based scheduling disciplines have been considered of primary importance in queueing theory (see, e.g., [9, Chap. 3]) and for practical applications in computer networking (see, e.g., [11] and the references therein). Despite the problems concerning the fairness [1, 3], the optimality of SRPT makes it practically appealing for the situations where job sizes can be predicted accurately. This is the case, for example, of TCP flows whose size is known in advance, e.g., in transferring static resources from a web server as shown in [8].

In this paper, we further investigate performance tradeoffs in dynamic speed scaling systems. One basic dilemma in these systems is what speed to use when there is only a single job in the system. Should it run at the highest speed, to minimize delay, or at the lowest speed, to conserve energy? To answer this question, we investigate a model in which we can determine the optimal speeds to use within a finite set of available speeds.

The main contributions in this paper are the following:

1. we propose a novel analytical model for dynamic speed scaling systems that use the SRPT scheduling policy, with  $K$  available speeds;
2. we derive exact analytic results for  $K = 2$  to optimize the system speeds and minimize the system cost function. The approach can be extended to deal with the case  $K = 3$ ;
3. we derive approximate analytic results for  $K = 2$  that can be generalized for larger  $K$ ;
4. we conduct numerical and simulation experiments to verify the accuracy of our analytical models and we provide new insights on the importance of the available speeds in dynamic speed scaling systems with SRPT scheduling.

The rest of this paper is organized as follows. Section 2 provides a brief summary of prior related work on dynamic speed scaling systems. Section 3 presents our system model. Section 4 derives our exact and approximate models for  $K = 2$  which may be extended to more general cases. Section 5 presents numerical results to evaluate the models and to quantify the benefits of frequency scaling. Finally, Sect. 6 concludes the paper.

## 2 Related Work

SRPT is a preemptive policy that always selects for service the pending job in the system with the least remaining work. In single-speed systems, SRPT is optimal

for mean response time [12]. Although SRPT minimizes the mean response time, it is rarely used in practice, since it can be unfair. In particular, large jobs may starve if small jobs have precedence.

Prior literature on speed scaling systems appears in both the theory and systems communities. The theoretical work typically focuses on formal mathematical proofs of the properties of speed scaling systems, such as optimality and fairness. Systems research typically focuses on robust solutions, rather than optimal ones. In this literature review, we focus primarily on the theoretical work, as relevant background context for our paper.

In speed scaling systems, there are many tradeoffs between service rate, response time, energy consumption, and fairness. Yao et al. [17] conducted one of the first analytical studies of dynamic speed scaling systems in which jobs have explicit deadlines, and the service rate is unbounded. Bansal et al. [4] considered an alternative approach that minimizes system response time, within a fixed energy budget. Other work has focused on finding the optimal fixed rate at which to serve jobs in a system with dynamically-settable speeds [7, 15, 16].

Energy-proportional speed scaling is a prevalent approach, which is nearly optimal [1, 2]. In this model, the power consumption  $P(s)$  of the system depends on the speed  $s$ , which in turn depends on the number of jobs in the system. Bansal, Chan, and Pruhs [2] showed that SRPT with the speed scaling function  $P^{-1}(n+1)$  is 3-competitive for an arbitrary power function  $P$ . Andrew et al. [1] showed that the optimal policy is SRPT with a job-count-based speed scaling function of the form  $s = P^{-1}(n\beta)$ .

Fairness in dynamic speed scaling systems is also an important consideration. In particular, speed scaling systems face inherent tradeoffs between fairness, robustness, and optimality [1]. Processor Sharing (PS) is always fair, even under speed scaling. However, the unfairness of SRPT is magnified under speed scaling, since large jobs tend to run at lower speeds (i.e., when the system is mostly empty). Although PS is fair, it is suboptimal for response time and energy [1].

In this paper, we focus on SRPT scheduling with job-count-based speed scaling. Our model builds upon ideas from Andrew *et al.* [1], as well as recent work by Elahi et al. on the autoscaling properties of dynamic speed scaling systems [6]. We derive exact and approximate models to facilitate optimization of the system cost, by determining the optimal service rates to use.

### 3 Description of the Model

In this section we introduce the queuing model that we consider in this paper together with the associated notation (a summary is given in Table 1). Jobs arrive into the system in a Poisson stream with rate  $\lambda$ , and are served by a single server. Each job consists of a random non-empty batch of i.i.d. service phases which will be referred to as *tasks*. The duration of each task, if served at speed 1 instruction per second, is distributed exponentially with mean 1. The number of tasks in a job's batch will be referred to as the *size* of the job. Those sizes are i.i.d. random variables with an arbitrary distribution: a job has size

**Table 1.** Summary of the notation used in the paper.

$\lambda$	Intensity of the arrival process
$q_i$	Prob. distribution of the number of tasks in a job
$Q$	Finite average job size
$K$	Number of frequency levels
$\mu_k$	Service rate at frequency level $k$
$u_k$	Stationary probability that frequency level $k$ is operating
$L$	Expected number of tasks in the system
$\mathbf{n} = (n_1, \dots, n_K)$	State of the system
$p_k(n)$	Stationary probability of observing $n$ tasks at level (queue) $k$
$w_k(z)$	Generating function of $p_k(n)$
$a(z)$	Generating function of $q_i$
$r_i$	Probability of a job size strictly larger than $i$
$b(z)$	Generating function of $r_i$
$\pi_k(n)$	Marginal stationary probability of observing $n$ tasks in queue $k$
$\pi(n_1, \dots, n_K)$	Stationary probability of state $(n_1, \dots, n_K)$

$i$  with probability  $q_i$  ( $i = 1, 2, \dots$ ). The average job size,  $Q$ , is assumed to be finite.

This job composition means that the possible distributions of job *lengths* (i.e. the sums of their constituent tasks), belong to a large sub-class of the Coxian distributions (see [5]), which are known to be quite general for practical purposes.

The job scheduling policy is a version of SRPT based on remaining sizes, rather than lengths. That is, at any moment, the job with the smallest number of remaining tasks is served. That policy is combined with a control mechanism whereby the frequency of the processor, i.e. the speed at which it works, is scaled according to the current load. There are  $K$  possible frequency levels. If there is only 1 job present, it is served at rate  $\mu_1$  tasks per unit time; if there are 2 jobs, then the one with fewer remaining tasks is served at rate  $\mu_2$  tasks per unit time, with  $\mu_2 > \mu_1$ ;  $\dots$ ; if there are  $K$  or more jobs present, then the one with the fewest remaining tasks is served at rate  $\mu_K$  tasks per unit time, with  $\mu_K > \mu_{K-1}$ .

One is faced with the question of how best to choose the frequency levels. Clearly there are trade-offs between the costs of increasing the processor speed and the costs of holding tasks in the system. We address that question by introducing a cost function which has two components: a cost proportional to the average number of tasks remaining across jobs present,  $L$ , and a cost proportional to the average square (see, e.g., [16]) of the service rate:

$$C = c_1 L + c_2 \sum_{k=1}^K u_k \mu_k^2, \quad (1)$$

where  $c_1$  and  $c_2$  are given coefficients, and  $u_k$  is the probability that frequency level  $k$  is in operation. We assume that when the system is empty, its power consumption is that corresponding to the lowest operating speed  $\mu_1$ .

The purpose of the subsequent analysis is to provide algorithms for computing  $L$  and  $u_k$ , and hence evaluate the cost function for a given set of parameters. The optimal values of  $\mu_k$  can then be found by applying an appropriate numerical search.

The operation of the scheduling policy can be modeled by using  $K$  task queues, numbered  $1, 2, \dots, K$ . Sort the jobs present in the system in decreasing order of the numbers of their remaining tasks. Then queue 1 contains the remaining tasks of job 1 (the largest in the system), queue 2 contains the remaining tasks of job 2, if any,  $\dots$ , queue  $K - 1$  contains the remaining tasks of job  $K - 1$ , if any, and queue  $K$  contains the remaining tasks of all other jobs, if any, whose sizes are smaller than that in queue  $K - 1$ . Queue  $K$  is the only one where there may be tasks from more than one job. Moreover, the number of tasks in queue  $i$  is always larger or equal to those in queue  $j$  if  $i < j$  and  $1 \leq i, j \leq K - 1$ , while queue  $K$  can contain an arbitrary number of tasks. At any epoch, only the tasks from the non-empty queue with the largest index are served with the speed associated with that queue. Therefore, the operating frequency of the processor depends on the number of jobs in the system and corresponds to speed  $\mu_i$  when there are  $i$  jobs in the system, for  $i = 1, \dots, K - 1$  and is  $\mu_K$  if there are  $K$  or more jobs, as required.

The state of the system at a given moment in time is a vector  $(n_1, n_2, \dots, n_K)$ , specifying the contents of the  $K$  queues. That vector satisfies  $(n_1 \geq n_2 \geq \dots \geq n_{K-1})$ , but it is possible that  $n_K > n_{K-1}$ . The server always serves the non-empty queue with the largest index, and if that index is  $i$ , it works at the rate of  $\mu_i$  tasks per unit time.

An incoming job of size  $s$  tasks which finds the system in state  $(n_1, n_2, \dots, n_K)$  may cause a reassignment of tasks to queues in order to preserve the shortest-remaining order. If  $s \leq n_{K-1}$ , then no such reassignment is necessary and the resulting state is  $(n_1, \dots, n_{K-1}, n_K + s)$ . Otherwise, the incoming  $s$  tasks replace the content of queue  $i$ , where  $i$  is the lowest index such that  $s > n_i$ . Queue  $K$  receives the tasks from queue  $(K - 1)$ , so that the new state is  $(n_1, \dots, n_{i-1}, s, n_i, \dots, n_K + n_{K-1})$ . If  $s > n_1$ , then the part of the vector preceding  $s$  is empty.

Consider now the steady-state probability,  $p_k(n)$ , that the total number of tasks in queues  $1, 2, \dots, k$  is  $n$ , while queues  $k + 1, \dots, K$  are empty ( $k = 1, 2, \dots, K$ ). For every  $k$ ,  $p_k(0)$  is the probability of an empty system, so we may sometimes omit the index and just write  $p(0)$ . The difference  $p_k(n) - p_{k-1}(n)$ , for  $k = 1, 2, \dots, K$  and  $n > 0$ , with  $p_0(n) = 0$  by definition, is the probability that there are  $n$  tasks present and the non-empty queue with the highest index is queue  $k$ . In other words, that is the probability that there are  $n$  tasks present and the service rate is  $\mu_k$ . These probabilities are 0 when  $n < k$ .

Let  $w_k(z)$  be the generating function of  $p_k(n)$ :

$$w_k(z) = \sum_{n=0}^{\infty} z^n p_k(n); \quad k = 1, \dots, K .$$

The last of these functions,  $w_K(z)$ , corresponds to the distribution of the total number of tasks in the system. It satisfies the normalizing condition  $w_K(1) = 1$ .

The marginal probabilities,  $u_k$ , that the server works at rate  $\mu_k$  (they appear in the cost function (1)), are given by

$$u_1 = w_1(1); \quad u_k = w_k(1) - w_{k-1}(1); \quad k = 2, \dots, K.$$

Let  $a(z)$  be the generating function of the job size distribution:

$$a(z) = \sum_{n=1}^{\infty} z^n q_n.$$

We shall also need the excess probabilities,  $r_n$ , that the size of a job is strictly greater than  $n$ , for  $n = 0, 1, \dots$  ( $r_0 = 1$ ). The generating function of  $r_n$ ,  $b(z)$ , is given by

$$b(z) = \sum_{n=0}^{\infty} z^n r_n = 1 + \sum_{n=1}^{\infty} z^n [1 - \sum_{j=1}^n q_j].$$

The following relation exists between  $b(z)$  and  $a(z)$ :

$$b(z) = \frac{1 - a(z)}{1 - z}. \tag{2}$$

This is established by expanding  $b(z) - zb(z)$  and performing cancellations. Note that the value of  $b(z)$  at  $z = 1$  is the average job size:  $b(1) = a'(1) = Q$ .

We have the following result.

**Lemma 1.** *The generating functions  $w_1(z)$ ,  $w_2(z)$ , ...,  $w_K(z)$  satisfy the relation*

$$w_K(z)[\mu_K - \lambda zb(z)] = \mu_1 p(0) + \sum_{k=1}^{K-1} (\mu_{k+1} - \mu_k) w_k(z). \tag{3}$$

Setting  $z = 1$  in (3) yields the normalization condition:

$$w_K(1) = \frac{\mu_1 p(0) + \sum_{i=1}^{K-1} (\mu_{i+1} - \mu_i) w_i(1)}{\mu_K - \lambda Q} = 1. \tag{4}$$

The following proposition gives the necessary and sufficient conditions for the stability of the system.

**Proposition 1.** *The queueing system is stable if and only if*

$$\lambda Q < \mu_K. \tag{5}$$

The proof uses the Pakes’ lemma [10] which gives sufficient conditions for stability of Markov chains.

The steady-state average number of tasks in the system,  $L$ , is given by  $w'_K(1)$ . First, by differentiating (2) and applying De L’Hôpital’s rule at  $z = 1$ , we find that  $b'(1) = -a''(1)/2$ . Now, taking the derivative of (3) at  $z = 1$  and rearranging terms, we obtain

$$L = \frac{\sum_{i=1}^{K-1} (\mu_{i+1} - \mu_i) w'_i(1) + \lambda(Q + \frac{1}{2} a''(1))}{\mu_K - \lambda Q}. \tag{6}$$

Thus the quantities that are needed in order to evaluate the cost function (1) are expressed in terms of the functions  $w_i(z)$  for  $i = 1, 2, \dots, K - 1$ , i.e. in terms of the probabilities of all states in which queue  $K$  is empty. In the following sections we provide exact and approximate solutions for those probabilities, in the cases  $K = 2$  and  $K = 3$ . The methodology employed can be extended to deal with higher values of  $K$ , at the price of increased complexity.

The following general result will be useful. Denote by  $\pi_1(i)$  the marginal probability that there are  $i$  tasks in queue 1 (and any numbers in the other queues). Then

$$\lambda r_n \sum_{i=0}^n \pi_1(i) = \mu_1 \pi(n + 1, 0, \dots, 0), \tag{7}$$

where  $\pi(n + 1, 0, \dots, 0)$  is the probability that queue 1 contains  $n + 1$  tasks and all other queues are empty. This equation is obtained by balancing the flows across a cut that separates the set of states with no more than  $n$  tasks in queue 1, from all other states.

The probabilities  $\pi_1(i)$  can also provide an expression for the expected residence time,  $T_1$ , of a job in queue 1. Indeed, if the system is in state  $(i, \cdot)$ , then jobs arrive into queue 1 at rate  $\lambda r_i$ . On the other hand, the average number of jobs in queue 1 (not tasks!) is equal to the probability that queue 1 is not empty, i.e.,  $1 - p(0)$ . Therefore, according to Little’s result:

$$T_1 = \frac{1 - p(0)}{\lambda \sum_{i=0}^{\infty} r_i \pi_1(i)}. \tag{8}$$

### 4 The Model with $K = 2$ Frequency Levels

In this section we consider our model with two frequency levels, i.e., the server works at speed  $\mu_1$  when there is only one job in the system, and  $\mu_2$  when there are at least two jobs, with  $\mu_1 < \mu_2$ . In Sect. 4.1 we provide the exact solution of the model, whereas in Sect. 4.2 we give an approximate, yet a more efficient approach to the computation of the stationary performance indices. The accuracy of the approximation will be assessed in Sect. 5 and will be shown to be very high.

### 4.1 Exact Analysis

The detailed system state is now a pair  $(i, j)$ , where  $i$  is the number of tasks in queue 1 and  $j$  is the number of tasks in queue 2. A service completion causes a transition from state  $(i, 0)$  to state  $(i - 1, 0)$  at rate  $\mu_1$  ( $i > 0$ ), and from state  $(i, j)$  to state  $(i, j - 1)$  at rate  $\mu_2$  ( $j > 0$ ). An arrival of a job of size  $k$  causes a transition from state  $(i, j)$  to state  $(i, j + k)$  if  $k \leq i$ , and to state  $(k, j + i)$  if  $k > i$ . The states  $(0, j)$ , for  $j > 0$ , are unreachable and their probabilities are 0.

Denote the probability of state  $(i, j)$  by  $\pi(i, j)$ . These probabilities satisfy the following global balance equations:

$$\lambda\pi(0, 0) = \mu_1\pi(1, 0). \tag{9}$$

$$(\lambda + \mu_1)\pi(i, 0) = \lambda q_i\pi(0, 0) + \mu_1\pi(i + 1, 0) + \mu_2\pi(i, 1). \tag{10}$$

$$(\lambda + \mu_2)\pi(i, j) = \lambda \sum_{k=1}^m q_k\pi(i, j - k) + \lambda q_i \sum_{k=1}^{m_1} \pi(k, j - k) + \mu_2\pi(i, j + 1), \tag{11}$$

where  $m = \min(i, j)$  and  $m_1 = \min(i - 1, j)$ .

Since all solutions of these equations are proportional to each other, it is enough to find one solution. The values  $\pi(i, j)$  can then be normalized by dividing each of them by their sum.

Start by setting  $\pi(0, 0) = 1$ . Equation (9) then gives  $\pi(1, 0) = \rho_1$ , where  $\rho_1 = \lambda/\mu_1$ . Note that this quantity does not represent offered load, since  $\lambda$  is the arrival rate of jobs, while  $\mu_1$  is a service rate of tasks.

Consider the probabilities  $\pi(1, j)$ , for  $j = 0, 1, \dots$ , and define the generating function

$$g_1(z) = \sum_{j=0}^{\infty} \pi(1, j)z^j. \tag{12}$$

When  $i = 1$  and  $j = 0$ , Eq. (10) can be rewritten as

$$(\lambda + \mu_2)\pi(1, 0) = \lambda q_1\pi(0, 0) + (\mu_2 - \mu_1)\pi(1, 0) + \mu_1\pi(2, 0) + \mu_2\pi(1, 1).$$

For  $i = 1$  and  $j \geq 1$ , Eq. (11) are

$$(\lambda + \mu_2)\pi(1, j) = \lambda q_1\pi(1, j - 1) + \mu_2\pi(1, j + 1).$$

Multiplying the above by  $z^j$  and summing, we get after a little manipulation (and remembering that  $\pi(0, 0) = 1$ ),

$$d_1(z)g_1(z) = \lambda q_1 z - [\mu_1 z + \mu_2(1 - z)]\pi(1, 0) + \mu_1 z\pi(2, 0), \tag{13}$$

where

$$d_1(z) = \lambda z(1 - q_1 z) + \mu_2(z - 1).$$

This expression for  $g_1(z)$  contains an unknown constant,  $\pi(2, 0)$ . However, note that the coefficient  $d_1(z)$  is negative for  $z = 0$  and positive for  $z = 1$ . Therefore,



there is a value,  $z_1$ , such that  $d_1(z_1) = 0$ . Since  $g_1(z)$  is finite on the whole interval  $z \in [0, 1]$ , the right-hand side of (13) must vanish at  $z = z_1$ . This gives

$$\mu_1 z_1 \pi(2, 0) = [\mu_1 z_1 + \mu_2(1 - z_1)]\pi(1, 0) - \lambda q_1 z_1 \pi(0, 0).$$

The next step is to consider the probabilities  $\pi_{2,j}$ , for  $j \geq 0$ , and their corresponding generating function

$$g_2(z) = \sum_{j=0}^{\infty} \pi(2, j)z^j.$$

Repeating the manipulations that led to (13), we obtain

$$d_2(z)g_2(z) = \lambda q_2 z - [\mu_1 z + \mu_2(1 - z)]\pi(2, 0) + \mu_1 z \pi(3, 0) + \lambda q_2 z^2 g_1(z), \tag{14}$$

where

$$d_2(z) = \lambda z(1 - q_1 z - q_2 z^2) + \mu_2(z - 1).$$

Again, the coefficient of  $g_2(z)$  is negative at  $z = 0$  and positive at  $z = 1$ . Therefore, there is a value  $z_2$  in the interval  $(0, 1)$ , such that  $d_2(z_2) = 0$ . Equating the right-hand side of (14) to 0 at  $z = z_2$ , determines the single unknown constant in that equation,  $\pi(3, 0)$ :

$$\mu_1 z_2 \pi(3, 0) = [\mu_1 z_2 + \mu_2(1 - z_2)]\pi(2, 0) - \lambda q_2 z_2 \pi(0, 0) - \lambda q_2 z_2^2 g_1(z_2).$$

The  $i$ 'th step in this process evaluates the generating function of the probabilities  $\pi(i, j)$ ,  $g_i(z)$ , in terms of the already known functions  $g_1(z)$ ,  $g_2(z)$ , ...,  $g_{i-1}(z)$ , and constants  $\pi(1, 0)$ ,  $\pi(2, 0)$ , ...,  $\pi(i, 0)$ :

$$d_i(z)g_i(z) = \lambda q_i z - [\mu_1 z + \mu_2(1 - z)]\pi(i, 0) + \mu_1 z \pi(i+1, 0) + \lambda q_i z \sum_{k=1}^{i-1} z^k g_k(z), \tag{15}$$

where

$$d_i(z) = \lambda z(1 - \sum_{k=1}^i q_k z^k) + \mu_2(z - 1)$$

The coefficient of  $d_i(z)$  has a zero,  $z_i$ , in the interval  $(0, 1)$ , which determines the new unknown constant  $\pi(i + 1, 0)$ .

These iterations continue until  $g_i(1) < \epsilon$ , for some sufficiently small  $\epsilon$ , or until the largest possible value of  $i$ , if the job sizes are bounded. Eventual termination is guaranteed if the queuing process is stable. At that point, all (unnormalized) probabilities  $\pi(i, 0)$ , and hence the function  $w_1(z)$ , have been determined.

The normalization constant,  $G$ , is given by (4):

$$G = \frac{\mu_1 \pi(0, 0) + (\mu_2 - \mu_1)w_1(1)}{\mu_2 - \lambda Q}. \tag{16}$$

Dividing all  $\pi(i, j)$  values, and hence  $w_1(1)$ , by  $G$ , completes the computation of the joint probability distribution of the states  $(i, j)$ . Lemma 1 now provides  $w_2(z)$ .

The total average number of tasks in the system,  $L$ , is given by (6), which now has the form

$$L = \frac{(\mu_2 - \mu_1)w_1'(1) + \lambda[Q + \frac{1}{2}a''(1)]}{\mu_2 - \lambda Q}.$$

The probabilities that the processor speed is  $\mu_1$ , and  $\mu_2$ , are  $w_1(1)$  and  $1 - w_1(1)$ , respectively.

### 4.2 Approximate Solution

In order to derive the approximation, consider the marginal probabilities,  $\pi(i, \cdot) = g_i(1)$ , that there are  $i$  tasks in queue 1, with  $\pi(0, \cdot) = \pi(0, 0)$ . These probabilities appear in the left-hand side of (7).

The fraction of time that the system spends in state  $(i, \cdot)$ , such that queue 2 is not empty, consists of the services of all tasks that join queue 2 when queue 1 reaches size  $i$ . This can happen when (a) the system is in state  $(k, \cdot)$ , for  $1 \leq k < i$ , and a job of size  $i$  arrives (in which case  $k$  tasks are transferred to queue 2), or (b) the system is in state  $(i, \cdot)$  and a job of size  $k$  arrives, for  $k \leq i$ ; all of its tasks join queue 2. Hence we can write

$$\pi(i, \cdot) - \pi(i, 0) = \lambda \left[ q_i \sum_{k=1}^{i-1} k\pi(k, \cdot) + \pi(i, \cdot) \sum_{k=1}^i kq_k \right] \frac{1}{\mu_2}; \quad i = 1, 2, \dots \quad (17)$$

The first sum in the right-hand side is absent when  $i = 1$ .

Introducing the notation

$$s_i = \sum_{k=1}^i k\pi(k, \cdot); \quad a_i = \sum_{k=1}^i kq_k, \quad (18)$$

we can rewrite (17) as

$$\pi(i, \cdot) = \frac{\pi(i, 0) + q_i \rho_2 s_{i-1}}{1 - \rho_2 a_i}; \quad i = 1, 2, \dots, \quad (19)$$

where  $\rho_2 = \lambda/\mu_2$  and  $s_0 = 0$  by definition.

Start with  $\pi(0, 0) = 1$  and  $\pi(1, 0) = \rho_1$ . Compute  $\pi(1, \cdot)$  from (19), then  $\pi(2, 0)$  from (7),  $\pi(2, \cdot)$  from (19),  $\pi(3, 0)$  from (7) and so on, up to the desired accuracy. Normalize, using (16).

This procedure is more economical and more stable than the exact solution.

It is worth of notice that both the exact and the approximate approach can be extended to handle  $K = 3$  levels of frequency. However, the complexity of the analysis and its computational cost increase. The approximate approach, instead, can easily be further generalized to models with more than three queues.

The accuracy of the approximation will be examined in Sect. 5.

## 5 Numerical and Simulation Results

In this section we describe several experiments aimed at evaluating the accuracy of the approximate solutions that have been proposed, and also observing the behaviour of the cost function (1). Systems with two and three frequency levels are examined. A remarkable observation emerging from these experiments is that, for the purposes of optimization, the models with  $K > 2$  may be neglected.

**$K = 2$  Frequency Levels.** We consider the model studied in Sect. 4. In Fig. 1, the cost function  $C$  is computed exactly and approximately, as described in Sect. 4, and is plotted against the queue 1 service rate,  $\mu_1$ . The two cost coefficients are  $c_1 = 1$  and  $c_2 = 2$ . The job arrival rate and the queue 2 service rate are fixed at  $\lambda = 1$  and  $\mu_2 = 7$ . A geometric distribution of job sizes is assumed, with mean 5, truncated at a maximum job size of 50. Thus the offered load,  $\lambda Q / \mu_2$  represents about 71% utilization. The value of  $\mu_1$  is varied between 1 and 6, in increments of 1. The cost function is convex in  $\mu_1$ . We have no formal proof of this, but it is invariably observed to be the case. Intuitively, at low values of  $\mu_1$  the holding costs dominate, while at large values the service rate costs dominate. Moreover, if a point is reached such that an increase in  $\mu_1$  leads to an increase in  $C$ , then clearly any further increase in  $\mu_1$  would make matters worse. The two plots are very close. The approximate solution underestimates  $C$  slightly, but the relative errors never exceed 1%. In particular, both solutions agree that the optimal value of  $\mu_1$  is 3 (subject to the granularity of the increments). We next examine the effect of increasing the variance of the job size distribution. Consider a rather extreme case where jobs have size 1, with probability 3/4, or size 17, with probability 1/4. The average job size is the same as in Fig. 1,  $Q = 5$ , but the variance has gone up from 20 to 337. All other parameters are the same, and again the cost function is plotted against the service rate  $\mu_1$ . Figure 2 shows that the increase in variance has led to an increase in costs of between 7% and 10%. The approximate solution is still within less than 1% of the exact one.

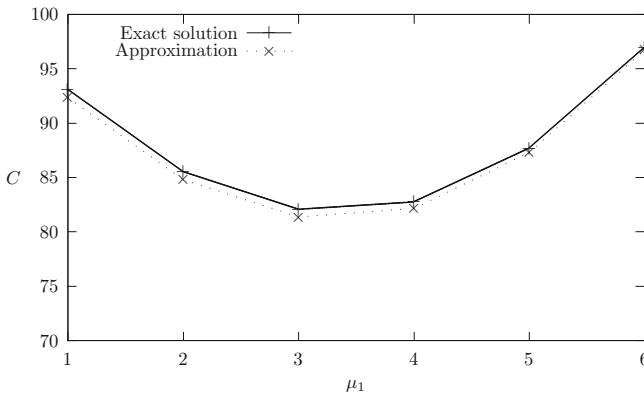


Fig. 1.  $K = 2$ : cost against  $\mu_1$

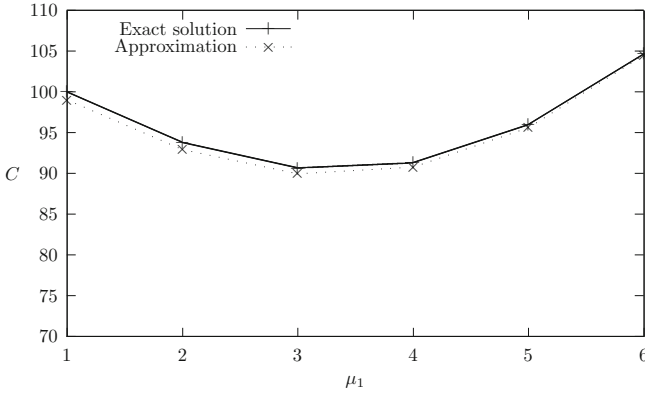


Fig. 2.  $K = 2$ : Large variance of job sizes

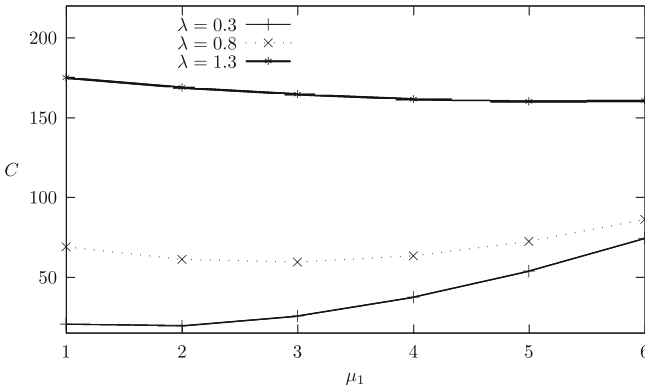
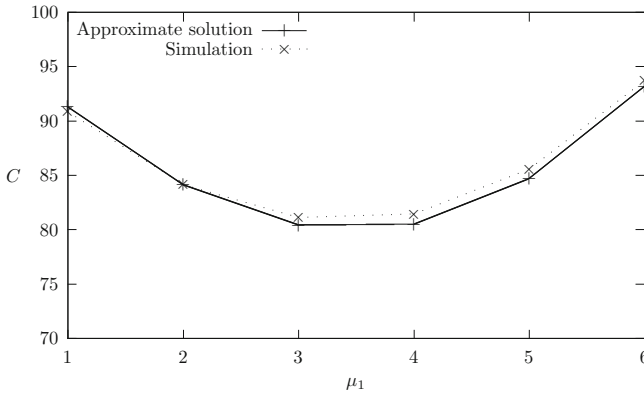


Fig. 3.  $K = 2$ : Impact of the workload intensity on the optimal frequency

The optimal value of  $\mu_1$  has not changed. The effect of the offered load on the shape of the cost function is illustrated in Fig. 3. Three loading regimes are considered, light, moderate and heavy. These are represented by the arrival rates  $\lambda = 0.3$ ,  $\lambda = 0.8$  and  $\lambda = 1.3$ ; they correspond to utilizations of about 21%, 57% and 93%, respectively. The other parameters are the same as in Fig. 1. Costs are evaluated exactly and are plotted against the queue 1 service rate,  $\mu_1$ . The figure suggests the following observations, all of which are quite intuitive. As the offered load increases, (a) costs increase; (b) the relative difference between the optimal and the pessimal costs decreases; (c) the optimal value of  $\mu_1$  increases.

**$K = 3$  Frequency Levels.** The next example evaluates the accuracy of the approximation for a system with three frequency levels. This time  $\mu_2$  and  $\mu_3$  are fixed at 6 and 7 tasks per second respectively, while  $\mu_1$  is varied between 1 and 6, at increments of 1. The job size distribution is geometric with mean 5, and the other parameters are the same as before. Rather than implementing the exact solution, Fig. 4 compares the approximated costs with simulated ones.



**Fig. 4.**  $K = 3$ : cost against  $\mu_1$

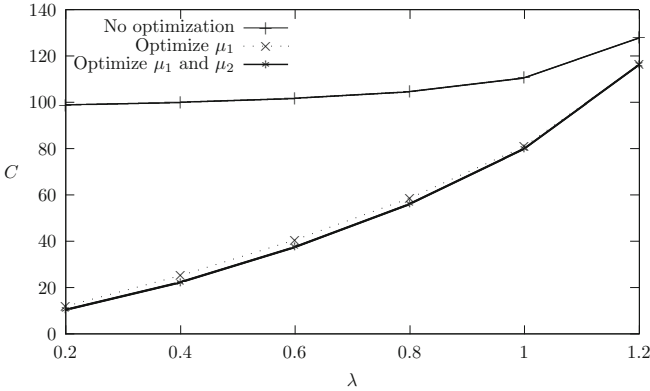
Each simulated point represents a run where 500,000 jobs arrive into the system (i.e., an average of 2.5 million tasks are served). The approximation is again very accurate, with relative errors not exceeding 1%. Moreover, the two plots agree on the optimal point of  $\mu_1 = 3$  (subject to the granularity of the increments). However, the difference in costs between  $\mu_1 = 3$  and  $\mu_1 = 4$  is very slight.

**The Benefits of Optimization.** The last experiment attempts to quantify the benefits of optimization, in the context of a 3-queue system under different loading conditions. Three policies are compared. The ‘default’ policy, or policy 0, does not optimize; it serves all three queues at rate  $\mu_3$ . Under policy 0, the system is equivalent to a single queue with batch arrivals. Policy 1 serves queues 2 and 3 at rate  $\mu_3$ , but uses the optimal value for  $\mu_1$  (found by a one-dimensional search). This amounts to an optimized  $K = 2$  system. Policy 2 serves queue 3 at rate  $\mu_3$ , but uses the optimal values for  $\mu_1$  and  $\mu_2$  (found by a two-dimensional search).

For consistency, all costs in this experiment were evaluated by applying the 3-queue approximation. We have relied on the established accuracy of that approximation. A feasible alternative would have been to evaluate policy 0 and policy 1 exactly (using the exact solutions for the cases  $K = 1$  and  $K = 2$ ), and resort to approximation only for policy 2.

In Fig. 5, the costs incurred by the above three policies are plotted against the job arrival rate  $\lambda$ . It varies between 0.2 and 1.2, while the top service rate remains fixed at  $\mu_3 = 7$ . Job sizes are distributed geometrically with mean 5, which means that the system loading varies between 14% and 86%. The cost coefficients are  $c_1 = 1$  and  $c_2 = 2$ . When searching for the best  $\mu_1$  and  $\mu_2$  values, the latter were incremented in steps of 0.5.

The results displayed in Fig. 5 are quite instructive. First, we observe that there is less to be gained by optimizing a heavily loaded system, than a lightly loaded one. Of course this was to be expected, since the holding costs become dominant under heavy loads, and minimizing those costs requires large service rates.

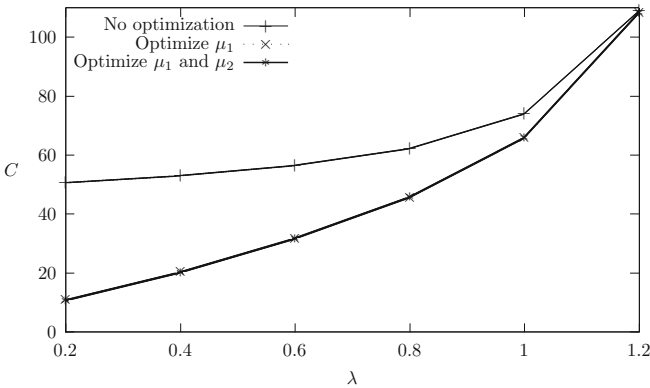


**Fig. 5.** The benefits of optimization for  $c_1 = 1$  and  $c_2 = 2$

The second observation is not so predictable: it seems that the big gains are obtained by optimizing just with respect to  $\mu_1$  (policy 1). The additional improvement achieved by optimizing with respect to  $\mu_2$  as well (policy 2), is quite minor. It is even debatable whether the expense of searching for policy 2 is justified by the benefits that it brings.

This last observation has practical importance. It suggests that the 2-queue model, rather than being just the simplest special case, is in fact a really significant model from the point of view of control and optimization. One may restrict the search for an optimal policy to the case  $K = 2$ , and be reasonably confident that the resulting policy would not be bettered by much.

To check whether the above conclusion remains valid under different cost structures, we have repeated the last experiment with several pairs of coefficients  $c_1$  and  $c_2$ . Figure 6 shows one such example, where  $c_1 = 2$  and  $c_2 = 1$  (i.e., holding tasks in the system incurs higher penalties than speeding up the server).



**Fig. 6.** The benefits of optimization for  $c_1 = 2$  and  $c_2 = 1$

The other parameters remain unchanged, and the costs of policies 0, 1 and 2 are plotted against the arrival rate,  $\lambda$ . The conclusion that the most important model is  $K = 2$ , is confirmed.

## 6 Conclusion

In this paper we studied systems employing the Shortest Remaining Processing Time scheduling discipline with frequency scaling. We have devised a model in which jobs consist of tasks whose service time are exponentially distributed. The distribution of the number of tasks in a job is arbitrary and this allows for a great flexibility in modelling the jobs' service time distribution and the accuracy of its estimation done by the system. The operating frequency is decided on the basis of the number of jobs in the system. The model characteristics allow us to study some important performance indices such as the expected number of tasks in the system, the probability of observing the system operating at a certain frequency level, the expected time that a job remains the largest job in the system and its expected size. We have introduced a cost function that takes into account the system's power consumption and the expected number of tasks in the system (and hence the expected response time). We focused the attention on the systems with 2 and 3 frequency levels and showed that the gain in the cost function of the latter with respect to the former is very small, whereas it is high between the system with 2 frequency levels and the one that does not apply any frequency scaling. This suggests that a simple system with 2 frequency levels, opportunely parametrised, can approximately give the benefits in terms of energy saving and quality of service of more complex systems.

**Acknowledgments.** The work described in this paper has been partially supported by the Università Ca' Foscari Venezia - DAIS within the IRIDE program.

## References

1. Andrew, L., Lin, M., Wierman, A.: Optimality, fairness, and robustness in speed scaling designs. In: Proceedings of ACM SIGMETRICS, pp. 37–48 (2010)
2. Bansal, N., Chan, H., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (2009)
3. Bansal, N., Harchol-Balter, M.: Analysis of SRPT scheduling: investigating unfairness. In: Proceedings of ACM SIGMETRICS, pp. 279–290 (2001)
4. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. *J. ACM* **54**, 3 (2007)
5. Cox, D.R.: A use of complex probabilities in the theory of stochastic processes. *Math. Proc. Cambridge Philos. Soc.* **51**(2), 313–319 (1955)
6. Elahi, M., Williamson, C.: Autoscaling effects in speed scaling systems. In: Proceedings of IEEE MASCOTS, pp. 307–312 (2016)
7. George, J., Harrison, J.: Dynamic control of a queue with adjustable service rate. *Oper. Res.* **49**(5), 720–731 (2001)

8. Harchol-Balter, M., Bansal, N., Shroeder, B., Agrawal, M.: Implementation of SRPT scheduling in web servers. In: Proceedings of IEEE International Parallel & Distributed Processing Symposium (IPDS), pp. 1–24 (2001)
9. Kleinrock, L.: *Queueing Systems: Computer Applications*, vol. 2. Wiley, New York (1976)
10. Pakes, A.G.: Some conditions for ergodicity and recurrence of Markov chains. *Oper. Res.* **17**(6), 1058–1061 (1969)
11. Rai, I.A., Urvoy-Keller, G., Vernon, M.K., Biersack, E.W.: Performance analysis of LAS-based scheduling disciplines in a packet switched network. In: Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS 2004/Performance 2004), pp. 106–117 (2004)
12. Schrage, L.: A proof of the optimality of the shortest remaining processing time discipline. *Oper. Res.* **16**, 678–690 (1968)
13. Skrenes, A., Williamson, C.: Experimental calibration and validation of a speed scaling simulator. In: Proceedings of IEEE MASCOTS, pp. 105–114 (2016)
14. Wierman, A.: Fairness and scheduling in single server queues. *Surv. Oper. Res. Manage. Sci.* **6**(1), 39–48 (2011)
15. Wierman, A., Andrew, L., Tang, A.: Power-aware speed scaling in processor sharing systems. In: Proceedings of IEEE INFOCOM (2009)
16. Wierman, A., Andrew, L., Tang, A.: Power-aware speed scaling in processor sharing systems: optimality and robustness. *Perform. Eval.* **69**, 601–622 (2012)
17. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of ACM Foundations of Computer Systems (FOCS), pp. 374–382 (1995)





# Biased Processor Sharing in Fork-Join Queues

Andrea Marin<sup>(✉)</sup>, Sabina Rossi, and Matteo Sottana

DAIS, Università Ca' Foscari, Venezia, Italy  
{marin,sabina.rossi,matteo.sottana}@dais.unive.it

**Abstract.** We consider a fork-join system in which a fixed amount of computational resources has to be distributed among the  $K$  tasks forming the jobs. The queueing disciplines of the fork- and join- queues are First Come First Served. At each epoch, at most  $K$  tasks are in service while the others wait in the fork-queues. We propose an algorithm with a very simple implementation that allocates the computational resources in a way that aims at minimizing the join-queue lengths, and hence at reducing the expected job service time. We study its performance in saturation and under exponential service time and provide a methodology to derive the relevant performance indices. Explicit closed-form expressions for the expected response time and join-queue length are given for the cases of jobs consisting of two, three and four tasks.

## 1 Introduction

In the literature, processor sharing (PS) queues have been widely studied since they are valid approximations of time sharing systems such as the *round robin* scheduling discipline for operating systems [10]. Moreover, PS queues have important theoretical properties that allow for analytical tractability of queueing networks (see, e.g., [4] and the references therein). In contrast with PS queues, where the jobs in the system receive the same amount of computational power, in biased PS (BPS) queues the computational power is distributed according to some policy that favours some classes of customers with respect to others [11, Chap. 11]. In [9], Kleinrock first introduces a form of BPS queue in which multiple classes of customers are statically associated with a certain weight and each job receives a computational power which is proportional to the weight of its class. This idea has been widely exploited to achieve fairness or for the optimisation of the systems' response time [1].

In this paper, we revisit the idea of BPS in the context of fork-join systems: jobs arrive at the system and are forked in  $K$  tasks that are enqueued in  $K$  fork-queues. Fork-join systems have a wide practical interest because they can be used to study parallel/distributed systems with synchronization constraints, similarly to what happens in the MapReduce computations.

In our model, since all the jobs are split into the same amount of tasks, we label the class of the task with the number of the fork- and join-queue associated

with it. The tasks in service are those at the front of the fork-queues. At the service completion, the task is stored in the corresponding join-queue. The join operation takes place once all or  $J$  out of  $K$  of the sibling tasks of a job are stored in the heads of the join-queues. Clearly, all the queues follow a First-Come-First-Served (FCFS) policy and the BPS has to share the total processing capacity among the  $K$  tasks in service.

The easiest way to address the problem is to note that there is no benefit in terms of the minimization of the expected occupancy of the join-queues in starting the service of a task belonging to a new job, if there are still older ones to be completed. An example of such a discipline is a round robin among the forked tasks. However, this policy has some practical drawbacks. First of all, there is a problem of reliability. If one of the task computations does not terminate due to an error, this would halt the whole system. Second, if the BPS queue is a model of a distributed system, it is not so simple to move the whole computational power from a computational unit to another: in general we have some fixed computational resources and some others are available for re-allocation. In that case the round-robin policy of the relocatable resources is not effective any more. Still, we may devise a policy that assigns all the relocatable resources to the classes that have served the smallest amount of tasks: in this case, in the event of errors the policy becomes very inefficient and, in general, it may be difficult to implement especially in a distributed environment since we need to keep track of the length of the join-queues and for each completion fairly allocate the resources to the shortest ones. We will call this discipline *feed the shortest join-queue* (FJQ) that differs from the well-know *join the shortest queue* because we do not move the tasks but the computational units.

In order to overcome these problems, we introduce a BPS discipline inspired by our previous work [13]. We define the neighbour of each class  $k$  as the class  $k + 1$  (we use the symbol  $k^+$  which takes into account the modularity). The weight of a class  $k$  is  $\mu$  if the system has served less tasks of class  $k$  than tasks of class  $k^+$ ,  $\eta$  otherwise. Therefore, differently from previous works, our strategy dynamically changes the class weight at each task completion event. We show that the queueing model has an elegant expression for the steady-state distribution of the join-queue lengths under the following assumptions: (1) the task service times are i.i.d. exponential random variables, (2) the model works in saturation, i.e., there is always a task to be fetched from the fork-queue. In the case of complete immediate join, i.e., the job is served when all its tasks are completed and the join takes place instantaneously, we give a methodology to compute the expected join-queue length, throughput and expected waiting time for arbitrary  $K$  and apply it for  $K = 2, 3, 4$ . More servers can be considered with the cost of an increased complexity in the derivation of the closed-form expressions. Finally, we compare this strategy with the round robin and the FJQ discipline. The numerical analysis shows that for low  $K$ , our discipline shows similar performance indices than the others in the ideal situation where the resources can be arbitrary moved across the system, no faults occur, and complete joins are performed. On the other hand, in the case of limitations on the minimum/maximum

amount of computational resources that can be assigned to each task class, our strategy performs much better than the round robin, and shows the benefits of FJQ with a limited overhead in the scheduling design.

The paper is structured as follows. Section 2 discusses the state of the art and related work. In Sect. 3 we introduce the three policies for BPS that we consider in this paper: RR, FJQ and DBS. The analytical solution of DBS is presented in Sect. 4 and we describe the method for deriving the expected performance indices. The method is applied to obtain the solutions for  $K = 2, 3, 4$ . In Sect. 5 we perform the comparison of the three scheduling disciplines and, finally, Sect. 6 concludes the paper.

## 2 Related Work

Biased processor sharing queues have been firstly introduced by Kleinrock [9] in order to model systems in which a set of computational resources are assigned to the customers in the system proportionally to their class priority. Since then, the idea of BPS have been widely adopted for several purposes including the implementation of size-based scheduling algorithms with partial knowledge of the job sizes [1] or the maximization of profits in the cloud [5]. In this paper we adopt this idea to solve the problems of resource allocation in a fork-join system. In contrast with the approach in [20], we do not collect statistics on the job execution times and base our policy merely on the state of the join-queues.

In [7, 21] the authors introduce the so called Flatto-Han-Wright (FHW) model consisting of only two exponential servers. In [16] Nelson and Tantawi derive the expression of the mean response time for the FHW model, when  $K = 2$  and the service times are i.i.d. exponential random variables. In [2, 3] queueing networks with fork-join nodes are studied, and a set of stability conditions are proposed. In [6], the authors use some results on order statistics to solve a class of fork-join queues. Among the approximate analysis, we mention also [18] where the authors derive numerical bounds for fork-join systems that can be computed efficiently. The same authors, in [19] derive stochastic bounds for fork-join systems and study some scalability laws of the system under renewal and no-renewal arrival of jobs.

The problem of balancing fork-join systems has been addressed also in [15]. However, in contrast with the scheduling proposed in this paper, the goal of the algorithm defined in [15] is that of balancing the join-queues in the case of servers with independent computational resources. Therefore, although we inherit the idea of defining the working speed of a server of class  $k$  based on the difference between the join-queue length of  $k$  and that of its neighbour, the definition of the problem and the resulting model is quite different. In fact, our contribution aims at allocating a fixed amount of resources, rather than devising an algorithm for the control of a processor speed. As consequence, the main focus of [15] was that of evaluating the throughput reduction caused by the speed control algorithm while in this work the throughput is not reduced. Indeed, here, we focus our attention on the computation of the expected number of tasks in the join-queues and hence on the expected waiting time of a served task.

### 3 Policies for Biased Processor Sharing in Fork-Join Queues

In this section we introduce three policies for assigning weights to the task classes and hence to address the problem of the resource allocation in the BPS system applied to fork-join queues.

We consider a system in which jobs are forked into  $K$  tasks that we consider statistically identical. Tasks are enqueued in the fork-queues which are numbered from 1 to  $K$  and all the tasks in the head of the fork-queues can be served. The service rate ranges between  $\gamma_{min}$  and  $\gamma_{max}$  and when all the fork-queues are non-empty, the sum of all the service rates is constant  $\gamma$ . Henceforth, without loss of generality, we consider  $\gamma = 1$ . Each task is labelled with a class that corresponds to the number of the fork-queue that it entered. When a task of class  $k$  has been served, it enters join-queue  $k$  and waits for the join. In this section, we are not interested in the details of the join policy, and the only condition that we ask is that once a job is considered served, the other tasks that it consisted of that may still be present in the fork-queues are not cancelled. The task will be discarded once it reaches the join queue. This clarification is necessary if the system uses  $J$  out of  $K$  tasks to serve the job for reliability purposes. Basically, this corresponds to the idea that the only information that is exchanged between the processing unit and the join unit are the served tasks, while the cancelling of the pending tasks would require a more complicated control system and an overhead in the message exchange.

One way of obtaining the BPS scheduling discipline is that of assigning a weight  $w_k(\mathbf{n})$  to each task class  $k$ , for  $k = 1, \dots, K$  given state  $\mathbf{n} = (n_1, \dots, n_K)$  of the join-queues, where  $n_k$  is the number of tasks waiting in join queue  $k$ . Notice that the weight of the classes dynamically changes during the computation. The speed of service of class  $k$  given  $\mathbf{n}$  is:

$$\gamma_k(\mathbf{n}) = \frac{w_k(\mathbf{n})}{\sum_{i=1}^K w_i(\mathbf{n})} \gamma, \tag{1}$$

where in the case of  $w_k(\mathbf{n}) = 0$  for all  $k$ , we take  $\gamma_k(\mathbf{n}) = \gamma/K$ . Weight function  $\gamma_k$  must satisfy the following conditions for all  $k$ :

$$\min_{\mathbf{n}} \{\gamma_k(\mathbf{n})\} \geq \gamma_{min} \quad \text{and} \quad \max_{\mathbf{n}} \{\gamma_k(\mathbf{n})\} \leq \gamma_{max}. \tag{2}$$

In the following paragraphs, we introduce the three queueing disciplines that we consider. In order to ease their presentation we provide their description under the saturation assumption (i.e., the fork-queues are never empty) that is required by our analytical results. This will also be the scenario under which the disciplines are compared in the following sections and represents a good approximation of the heavy-load case.

*Round Robin (RR).* In the round robin queueing discipline the weights are assigned in such a way that only one class has speed  $\gamma_{max}$  while all the others work at speed  $\gamma_{min}$ , therefore  $\gamma = (K - 1)\gamma_{min} + \gamma_{max}$ . Let  $k$  be the class working

at  $\gamma_{max}$ , then at its task completion event the service speed will change to  $\gamma_{min}$  and that of class  $k^+$  will change to  $\gamma_{max}$ . Recall that  $k^+ = (k \bmod K) + 1$ . A special case is that of  $\gamma_{min} = 0$  and  $\gamma_{max} = \gamma$ , for which we can easily describe the discipline with the weight functions (1).

*Feed the Shortest Join-Queue (FJQ).* Let  $\mathbf{n}$  be the vector of the join-queue lengths, and let  $n_{min} = \min_i \{n_i\}$ , then FJQ assigns the weights as follows:

$$w_k(\mathbf{n}) = \begin{cases} \mu & \text{if } n_k = n_{min} \\ \eta & \text{otherwise,} \end{cases}$$

where  $\eta < \mu$ . Notice that, in the case of immediate join,  $n_{min} = 0$  and hence the implementation of the discipline is rather simplified. FJQ gives the highest weights to the shortest queues. Weights  $\mu$  and  $\eta$  must satisfy:

$$\frac{\mu}{\mu + (K - 1)\eta} \gamma \leq \gamma_{max} \quad \text{and} \quad \frac{\eta}{\eta + (K - 1)\mu} \gamma \geq \gamma_{min} \quad (3)$$

A special case is when  $\gamma_{min} = 0$  and  $\gamma_{max} = \gamma$  with immediate join, where we can set  $\eta = 0$  and  $\mu = 1$  to model a discipline that only serves the task classes which are needed to complete a job and hence vector  $\mathbf{n}$  contains only 0 and 1.

Notice that, according to the definition we have given of FJQ, the weight associated with queue  $k$  depends only on the local state  $n_k$  in case of immediate join. Other definitions could be proposed that take into account the global state of the queues, but they would introduce a higher overhead and a their implementation would be more complicated.

*Difference Based Scheduling (DBS).* In DBS each class  $k$  is associated with a counter  $m_k$  that is increased at each task completion of class  $k$  and is decreased when there is task completion at class  $k^+$ . In other words, we have that  $m_k = n_k - n_{k^+}$ , i.e., it is the difference in the number of completed class  $k$  and class  $k^+$  tasks. The weight is assigned as follows:

$$w_k(\mathbf{n}) = w(m_k) = \begin{cases} \mu & \text{if } m_k < 0 \\ \eta & \text{otherwise,} \end{cases} \quad (4)$$

where  $\eta, \mu$  must satisfy Constraints (3). Notice that with respect to policy FJQ, DBS needs only local information to assign the weights and hence its implementation results very simple and with low overhead. Moreover, DBS is robust with respect to faults. Assume that class  $k$  computation is blocked due to a hardware failure or a software error. Clearly, its state variable  $m_k$  cannot be increased and will be decremented by 1 at each neighbour  $k^+$  task completion. Nevertheless, DBS will assign to class  $k$  only part of the relocatable computational power in contrast with FJQ that, in the long run, will assign all the relocatable computational power to the failed class.

## 4 The Model for the DBS

In this section we propose a model for DBS queueing discipline that works under the following assumptions:

- A1** The model is saturated, i.e., the fork queues have always at least one task to serve;
- A2** The join is immediate and complete;
- A3** The service time distribution is exponential.

Since the purpose of this work is comparing DBS, FJQ and RR, assumptions A1 and A2 simply give a reasonable testing scenario. The assumption of saturation is designed to stress the scheduling policies and it should be noticed that it is possible to have join-queues with expected finite length (in the long run) even in case of saturation thanks to the assumption of immediate join. Assumption A3 is the most restrictive, but allows us to define an analytical model for the system as in [7, 16].

*State Space and Markov Chain.* We consider a system consisting of  $K \geq 2$  task classes. For any  $k \in \{1 \dots, K\}$ , we use  $n_k(t)$  to denote the stochastic process associated with the number of tasks in the join-queue  $k$  at time  $t \in \mathbb{R}$ . Clearly, we have that  $n_k(t) \in \mathbb{N}$  for all  $t \geq 0$ . We are interested in studying the stochastic process  $X_K(t) = (n_1(t), \dots, n_K(t))$  on the state space

$$\mathcal{S} = \{\mathbf{n} = (n_1, \dots, n_K) \mid n_k \in \mathbb{N} \wedge \exists k : n_k = 0\}.$$

where  $k \in \{1 \dots, K\}$ . Notice that the fact that there always exists at least one  $k$  such that  $n_k = 0$  follows from A2.  $X_K(t)$  is a continuous time Markov chain (CTMC) defined as follows: for  $h \rightarrow 0^+$  and  $t > 0$ ,

$$Pr\{X_K(t+h) = \mathbf{n} + \mathbf{e}_k \mid X_K(t) = \mathbf{n} \wedge n_k \neq 0\} = \gamma_k(\mathbf{n})h + o(h) \tag{5}$$

$$Pr\{X_K(t+h) = \mathbf{n} + \mathbf{e}_k \mid X_K(t) = \mathbf{n} \wedge n_k = 0 \wedge \sum_{i=1}^K \delta_{n_i=0} > 1\} = \gamma_k(\mathbf{n})h + o(h) \tag{6}$$

$$Pr\{X_K(t+h) = \mathbf{n} - \sum_{i=1, i \neq k}^K \mathbf{e}_i \mid X_K(t) = \mathbf{n} \wedge n_k = 0 \wedge \sum_{i=1}^K \delta_{n_i=0} = 1\} = \gamma_k(\mathbf{n})h + o(h) \tag{7}$$

$$Pr\{X_K(t+h) = \mathbf{n} \mid X_K(t) = \mathbf{n}\} = 1 - \left(\sum_{k=1}^K \gamma_k(\mathbf{n})\right)h + o(h) \tag{8}$$

where  $\mathbf{e}_k$  is a  $K$ -dimensional vector with all zeros with the exception of component  $k$  which is 1,  $\delta_P$  is 1 if proposition  $P$  is true otherwise it is equal to 0, and  $\gamma_k(\mathbf{n})$  is defined as in Eq. (1), where  $w_k(\mathbf{n}) = w(n_k - n_{k+})$  and  $\bar{w}(\mathbf{n}) = \sum_{i=1}^K w_i(\mathbf{n})$ .

### 4.1 Stationary Analysis of $X_K(t)$ for $K = 2$

In this section, we show that  $X_2(t)$  is stationary if and only if it is reversible. Indeed, the stochastic process corresponds to a random walk on the line. The transition graph of  $X_2(t)$  is depicted in Fig. 1. Notice that if the chain is ergodic, i.e., all the states are positive recurrent, then the CTMC is trivially reversible [8] and vice versa. We also observe that if we use Eq. (4) to define  $\gamma_k(\mathbf{n})$  then the model is stationary if and only if  $\eta < \mu$ .

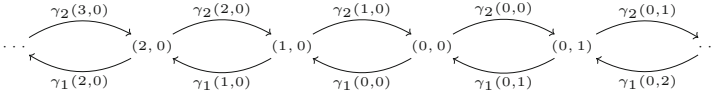


Fig. 1. Transition graph of  $X_2(t)$ .

### 4.2 Stationary Analysis of $X_K(t)$ for $K > 2$

Now, we consider the case  $K > 2$ . Notice that  $X_K(t)$ ,  $K > 2$ , is trivially not reversible, since we can find at least one state  $\mathbf{n}$  which has a transition to  $\mathbf{n}'$  but not the other way around. Theorem 1 gives the expression of the invariant measure of  $X_K(t)$ . For the sake of brevity, we sketch the proof based on the verification of the global balance equations although a constructive approach based on the notion of  $\rho$ -reversibility can be efficiently applied [12,14].

**Theorem 1.**  $X_K(t)$  has a product-form invariant measure given by:

$$\pi_K(\mathbf{n}) = \frac{1}{G_K} \bar{w}(\mathbf{n}) \left( \prod_{i=1}^K \frac{\prod_{m=0}^{n_i - n_{i+} - 1} w(m)}{\prod_{m=n_i - n_{i+}}^{-1} w(m)} \right) \tag{9}$$

which can be normalized on  $G_K$  to give the equilibrium distribution whenever  $X_K(t)$  is ergodic. In Eq. (9) we assume that empty products are equal to 1.

*Proof.* Recall that, we are considering  $\gamma = 1$ , without loss of generality. Assume that we have only one 0 in  $\mathbf{n}$  and that it occurs in the first position so we have that  $\mathbf{n} = \{0, n_2, n_3, \dots, n_K\}$  with  $n_i > 0$  for  $i = 2, 3, \dots, K$  and  $n_1 = 0$ . Then, we can write:

$$\pi_K(\mathbf{n}) \frac{\bar{w}(\mathbf{n})}{\bar{w}(\mathbf{n})} = \sum_{j=2}^K \pi_K(\mathbf{n} - \mathbf{e}_j) \frac{w(n_j - 1 - n_{j+})}{\bar{w}(\mathbf{n})} + \pi_K(\mathbf{n} + \sum_{j=2}^K \mathbf{e}_j) \frac{w(-n_2 - 1)}{\bar{w}(\mathbf{n})}.$$

Let  $k^- = (k \bmod K) + K - 1$ , by replacing Expression (9) in this equation, we have:

$$\begin{aligned} \pi_K(\mathbf{n}) = \sum_{j=2}^K \pi_K(\mathbf{n}) \frac{1}{w(n_j - 1 - n_{j+})} \frac{w(n_j - 1 - n_{j+})}{\bar{w}(\mathbf{n})} w(n_{j^-} - n_j) \\ + \pi_K(\mathbf{n}) \frac{w(n_K)}{w(-n_2 - 1)} \frac{w(-n_2 - 1)}{\bar{w}(\mathbf{n})}, \end{aligned}$$

that can be simplified as follows:

$$\begin{aligned} \pi_K(\mathbf{n}) &= \sum_{j=2}^K \pi_K(\mathbf{n}) \frac{w(n_{j^-} - n_j)}{\bar{w}(\mathbf{n})} + \pi_K(\mathbf{n}) \frac{w(n_K)}{\bar{w}(\mathbf{n})}, \\ \pi_K(\mathbf{n}) &= \pi_K(\mathbf{n}) \sum_{j=1}^K \frac{w(n_{j^-} - n_j)}{\bar{w}(\mathbf{n})}, \end{aligned}$$

$$\bar{w}(\mathbf{n}) = \sum_{j=1}^K w(n_{j-} - n_j) = \sum_{j=1}^K w(n_j - n_{j+}) = \sum_{j=1}^K w_j(\mathbf{n}).$$

The same identity can be verified for an arbitrary number of 0s in the state.  $\square$

Let us define  $w_k(\mathbf{n})$  as in Eq. (4) to implement the DBS scheduler, then Theorem 2 gives us necessary and sufficient conditions for the ergodicity of  $X_K(t)$  as well as the expression for the normalising constant.

**Theorem 2.** *Let  $X_K(t)$  be the CTMC defined according to Eqs. (5)–(8) with  $\gamma_k(\mathbf{n})$  defined according to the weights of Eq. (4). Then, the following properties hold:*

- $X_K(t)$  is ergodic for all finite  $K \geq 2$  if and only if  $\eta < \mu$ ;
- If  $\eta < \mu$ , the normalising constant of Eq. (9) that gives the unique stationary distribution is:

$$G_{K,\eta,\mu} = K\eta + \sum_{j=1}^{K-1} (j\eta + (K-j)\mu) \binom{K}{j} \binom{K-1}{j-1} (K-j)\beta(\eta/\mu, K-j, 1-K), \quad (10)$$

where  $\beta$  denotes the incomplete Euler’s Beta-function:

$$\beta(z, a, b) = \int_0^z u^{a-1}(1-u)^{b-1} du.$$

*Proof.* The proof is similar to that shown in [13, Theorem 2].  $\square$

The evaluation of the incomplete  $\beta$  function is efficient and for small values of  $K$  can be performed symbolically as stated by the following corollary whose proof follows by the properties of hypergeometric functions [17]:

**Corollary 1.** *Given a finite  $K \in \mathbb{N}$ ,  $K \geq 2$ , the expression of  $G_{K,\eta,\mu}$  is a rational function which can be computed as:*

$$G_{K,\eta,\mu} = K\eta + \sum_{j=1}^{K-1} (j\eta + (K-j)\mu) \binom{K}{j} \binom{K-1}{j-1} (K-j) \left(\frac{\eta/\mu}{1-\eta/\mu}\right)^{k-j} \cdot \sum_{v=0}^{j-1} (-1)^v \binom{j-1}{v} \frac{1}{K-j+v} \left(\frac{\eta/\mu}{\eta/\mu-1}\right)^v$$

The stationary probabilities of the model can be expressed as follows. First observe that for each state  $\mathbf{n} = (n_1, \dots, n_K)$  of our model,  $\sum_{i=1}^K (n_i - n_{i+}) = 0$ . Let us denote by  $m_k$  the difference  $n_k - n_{k+}$  for  $k \in \{1, \dots, K\}$ .



**Corollary 2.** *Let  $X_K(t)$  be the CTMC defined according to Eqs. (5)–(8) with  $\gamma_k(\mathbf{n})$  defined according to the weights defined by Eq. (4) and  $\eta < \mu$ . Then, we have:*

- let  $\mathbf{0}$  be the state  $(n_1, \dots, n_K)$  where each  $n_k = 0$  for  $k \in \{1, \dots, K\}$ , then

$$\pi_{K,\eta,\mu}(\mathbf{0}) = \frac{1}{G_{K,\eta,\mu}} K \eta$$

- let  $\mathbf{n} = (n_1, \dots, n_K)$  be a state of  $X_K(t)$ , let  $j$  be the number of non-negative differences  $m_k$  in  $\mathbf{n}$  and  $m$  be the sum of the positive  $m_k$  in  $\mathbf{n}$ , then

$$\pi_{K,\eta,\mu}(\mathbf{n}) = \frac{1}{G_{K,\eta,\mu}} (j\eta + (K - j)\mu) \left(\frac{\eta}{\mu}\right)^m.$$

The proof easily follows from Theorem 2.

### 4.3 Marginal Probabilities and Expected Performance Indices

Clearly, one desires to obtain the marginal stationary probability of finding  $n$  tasks in a join-queue and, consequently, the expected stationary number of tasks in a join-queue. Henceforth, we assume  $\eta < \mu$  or, equivalently, the model to be stable.

The marginal probabilities  $\pi_{K,\eta,\mu}^*$  are defined as follows: for all  $n \in \mathbb{N}$ ,

$$\pi_{K,\eta,\mu}^*(n) = \sum_{\mathbf{n}=(n,n_2,\dots,n_K) \in \mathcal{S}} \pi_{K,\eta,\mu}(\mathbf{n}), \tag{11}$$

and the expected number of tasks that are present in an arbitrary join-queue:

$$N_{K,\eta,\mu} = \sum_{n=1}^{\infty} n \pi_{K,\eta,\mu}^*(n). \tag{12}$$

In order to obtain these important indices of the model, we fix  $K$  and describe a procedure that can be applied to derive them. Then, we show the expressions of the marginal probabilities and the expected join-queue lengths for  $K = 2, 3, 4$ .

For what concerns the throughput, observe that each class of tasks receives in the average a computational power of  $\gamma/K$ . Since the model is stable, this means that the throughput is exactly  $\gamma/K$ . Finally, we can obtain the expected waiting time of task in a join-queue by using Little’s law as  $KN_{K,\eta,\mu}/\gamma$ .

**Computation of the Marginal Probabilities.** We describe a procedure to compute the marginal probabilities of finding  $n$  tasks in a join-queue. The expected stationary number of tasks in a join-queue is computed in a similar manner.

As stated above, the marginal probability for a specific  $n \in \mathbb{N}$  is obtained by Eq. (11), i.e., we have to sum the probabilities of all the configurations of the

form  $\mathbf{n} = (n, n_2 \dots, n_K)$ . Our procedure, for  $n > 1$ , consists in the following steps: (1) first we fix all the possible ways in which the join-queues have 0 tasks waiting for the join; the output of this step is a set of configuration schemata of the form:

$$(n, 0, n_3 \dots, n_K), (n, 0, 0, n_4 \dots, n_K), (n, 0, n_3, 0 \dots, n_K), \dots (n, 0, 0, 0 \dots, 0);$$

(2) for each configuration scheme computed in the previous step, we determine all the possible ways in which a non-empty queue has a number of tasks waiting for join which is greater or equal than the number of tasks of the corresponding neighbour; the output of this step is a set of configuration schemata of the form:

$$(n, 0, +, - \dots, +), (n, 0, -, 0 \dots, -), (n, -, -, -, \dots, -) \dots (n, +, +, +, \dots, +);$$

(3) for each configuration scheme computed in the previous step we count the number of configurations belonging to the given scheme and hence we can compute the sum of their probabilities.

Below we show the application of this algorithm for the case  $K = 3$ . Notice that the complexity of the algorithm grows quickly with  $K$  but once implemented in a symbolic mathematical software, it provides the expression for the join-queue length as function of the model parameters.

**Marginal Probabilities and Average Performance indices for  $K = 2$ .**

Let us analyse the behavior of  $X_K(t)$  with  $K = 2$ .

**Proposition 1.** *The marginal probabilities of  $X_2(t)$  are defined as follows:*

$$\pi_{2,\eta,\mu}^*(n) = \begin{cases} \frac{3}{4} - \frac{\eta}{4\mu} & \text{if } n = 0, \\ \frac{\mu^2 - \eta^2}{4\mu^2} \left(\frac{\eta}{\mu}\right)^{n-1} & \text{if } n \geq 1. \end{cases}$$

*The expected number of tasks in a join-queue is:*

$$N_{2,\eta,\mu} = \frac{\eta + \mu}{4(\mu - \eta)}.$$

In this case the proof is rather simple and one can resort to the fact that  $X_2(t)$  is reversible. Notice that:

$$\lim_{\eta \rightarrow \mu^-} \pi_{2,\eta,\mu}^*(0) = \frac{1}{2} \quad \text{and} \quad \lim_{\eta \rightarrow \mu^-} N_{2,\eta,\mu} = \infty,$$

where the first limit tells us that if we observe a random join-queue in the long run we have 50% of probability to find it empty even if the system is unstable. In fact, this follows from the immediate join assumption that enforces one of the two join-queues to be empty. The second limit is coherent with Theorem 2 and states that when  $\eta \rightarrow \mu$ , the expected join-queue length is infinite. We can also observe that  $\partial N_{2,\eta,\mu} / \partial \eta$  is strictly positive in  $[0, \mu]$  showing that the minimum join-queue length (and hence expected waiting time) corresponds to  $\eta = 0$ . In other words, the minimal expected join-queue length is 0.25 for which the probability of the empty queue is 3/4 and that of a single task is 1/4 (notice that we took  $0^0 = 1$  in order to avoid the extra case of  $n = 1$ ).

**Marginal Probabilities and Average Performance indices for  $K = 3$ .**

We now consider  $X_3(t)$  and obtain the following results.

**Proposition 2.** *The marginal probabilities of  $X_3(t)$  are defined as follows:*

$$\pi_{3,\eta,\mu}^*(n) = \begin{cases} \frac{2\mu}{3(\eta+\mu)} & \text{if } n = 0, \\ \frac{(\eta-\mu)(\eta^2 n - 2\eta\mu - n\mu^2)}{3\mu^2(\eta+\mu)} \left(\frac{\eta}{\mu}\right)^{n-1} & \text{if } n \geq 1. \end{cases}$$

The expected number of jobs is:

$$N_{3,\eta,\mu} = \eta \left( \frac{1}{3(\eta+\mu)} + \frac{1}{\mu-\eta} \right) + \frac{1}{3}.$$

*Proof.* In order to prove the proposition, we resort to the procedure described above. Let us consider the case  $m > 1$  and determine all the possible ways in which the join-queues have 0 tasks waiting for the join. We obtain the following set of configuration schemata:  $\{(n, 0, 0), (n, y, 0), (n, 0, y)\}$ . Then, we determine all the possible ways in which a non-empty queue has a number of tasks waiting for join which is greater or equal than the number of tasks of the corresponding neighbour on its left and obtain the set  $\{(n, 0, 0), (n, 0, +), (n, -, 0), (n, +, 0)\}$ . The probability of  $(n, 0, 0)$  is  $\frac{1}{G_{3,\eta,\mu}}(2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n$ .

Consider now the scheme  $(n, 0, +)$  representing all the configurations of the form  $(n, 0, y)$  where either  $y < n$  or  $y \geq n$ . The sum of the probabilities for  $y < n$  is  $\frac{1}{G_{3,\eta,\mu}} \sum_{y=1}^{n-1} (\eta + 2\mu) \left(\frac{\eta}{\mu}\right)^n$ , while for  $y \geq n$  the sum is  $\frac{1}{G_{3,\eta,\mu}} \sum_{y=n}^{\infty} (2\eta + \mu) \left(\frac{\eta}{\mu}\right)^y$ .

The scheme  $(n, -, 0)$  represents all the configurations  $(n, y, 0)$  where  $y \leq n$  and the sum of the probabilities associated with such configurations is  $\frac{1}{G_{3,\eta,\mu}} \sum_{y=1}^n (2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n$ . Finally, the scheme  $(n, +, 0)$  represents all the configurations  $(n, y, 0)$  where  $y \geq n$  and the sum of the probabilities of such configurations is  $\frac{1}{G_{3,\eta,\mu}} \sum_{y=n+1}^{\infty} (\eta + 2\mu) \left(\frac{\eta}{\mu}\right)^y$ .

Summing up, we obtain that

$$\begin{aligned} \pi_{3,\eta,\mu}^*(n) = & \frac{1}{G_{3,\eta,\mu}} \left( (2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n + \sum_{y=1}^{n-1} (\eta + 2\mu) \left(\frac{\eta}{\mu}\right)^n + \sum_{y=n}^{\infty} (2\eta + \mu) \left(\frac{\eta}{\mu}\right)^y \right. \\ & \left. + \sum_{y=1}^n (2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n + \sum_{y=n+1}^{\infty} (\eta + 2\mu) \left(\frac{\eta}{\mu}\right)^y \right). \end{aligned}$$

We can prove that all the above series are convergent and in particular:

$$\begin{aligned} \pi_{3,\eta,\mu}^*(n) = & \frac{1}{G_{3,\eta,\mu}} \left( n(2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n + (n-1)(\eta + 2\mu) \left(\frac{\eta}{\mu}\right)^n \right. \\ & \left. + (2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n - \frac{\mu(2\eta + \mu) \left(\frac{\eta}{\mu}\right)^n}{\eta - \mu} - \frac{\eta(\eta + 2\mu) \left(\frac{\eta}{\mu}\right)^n}{\eta - \mu} \right) \end{aligned}$$

$$\begin{aligned}
 &= \frac{1}{G_{3,\eta,\mu}} \left( \frac{3 \left(\frac{\eta}{\mu}\right)^n (\eta^2 n - 2\eta\mu - n\mu^2)}{\eta - \mu} \right) \\
 &= \frac{(\eta - \mu) \left(\frac{\eta}{\mu}\right)^{n-1} (\eta^2 n - 2\eta\mu - n\mu^2)}{3\mu^2(\eta + \mu)}.
 \end{aligned}$$

The case  $n = 0$  follows by:

$$\begin{aligned}
 \pi_{3,\eta,\mu}^*(0) &= 1 - \sum_{n=1}^{\infty} \pi_{3,\eta,\mu}^*(n) \\
 &= 1 - \sum_{n=1}^{\infty} \frac{(\eta - \mu) \left(\frac{\eta}{\mu}\right)^{n-1} (\eta^2 n - 2\eta\mu - n\mu^2)}{3\mu^2(\eta + \mu)} = \frac{2\mu}{3(\eta + \mu)}.
 \end{aligned}$$

The computation of the expected number of tasks in a join-queue is simply obtained from the marginal probabilities as:

$$N_{3,\eta,\mu} = \sum_{n=1}^{\infty} n \pi_{3,\eta,\mu}^*(n) = \eta \left( \frac{1}{3(\eta + \mu)} + \frac{1}{\mu - \eta} \right) + \frac{1}{3}. \quad \square$$

In this case, we have that for  $\eta \rightarrow \mu^-$   $\pi_{3,\eta,\mu}^* \rightarrow 1/3$ , as expected, and the minimum average number of tasks in a join-queue is  $1/3$  when  $\eta = 0$ .

**Marginal Probabilities and Average Performance indices for  $K = 4$ .**  
 The last example that we propose is the solution of the model for  $X_4(t)$ .

**Proposition 3.** *The marginal probabilities of  $X_4(t)$  are defined as follows: for every  $m \in \mathbb{N}$ ,*

- if  $n = 0$  then  $\pi_{4,\eta,\mu}^*(0) = \frac{5\mu(\eta + \mu)}{8(\eta^2 + 3\eta\mu + \mu^2)}$
- if  $n \geq 1$  then

$$\begin{aligned}
 \pi_{4,\eta,\mu}^*(n) &= (\mu - \eta) \left(\frac{\eta}{\mu}\right)^{n-1} \cdot \\
 &\frac{(\eta^3 n(2n - 1) - \eta^2(n + 5)(2n - 1)\mu + \eta((9 - 2n)n + 5)\mu^2 + n(2n + 1)\mu^3)}{8\mu^2(\eta^2 + 3\eta\mu + \mu^2)}
 \end{aligned}$$

The expected number of jobs is:

$$N_{4,\eta,\mu} = \frac{(\eta + \mu) (3\eta^2 + 23\eta\mu + 3\mu^2)}{8(\mu - \eta) (\eta^2 + 3\eta\mu + \mu^2)}.$$

We observe that also in this case  $\partial N_{4,\eta,\mu} / \partial \eta$  is positive for  $0 \leq \eta \leq \mu$  and hence we have the shortest join-queue length when  $\eta = 0$ , where  $N_{4,0,\mu} = 3/8$ .

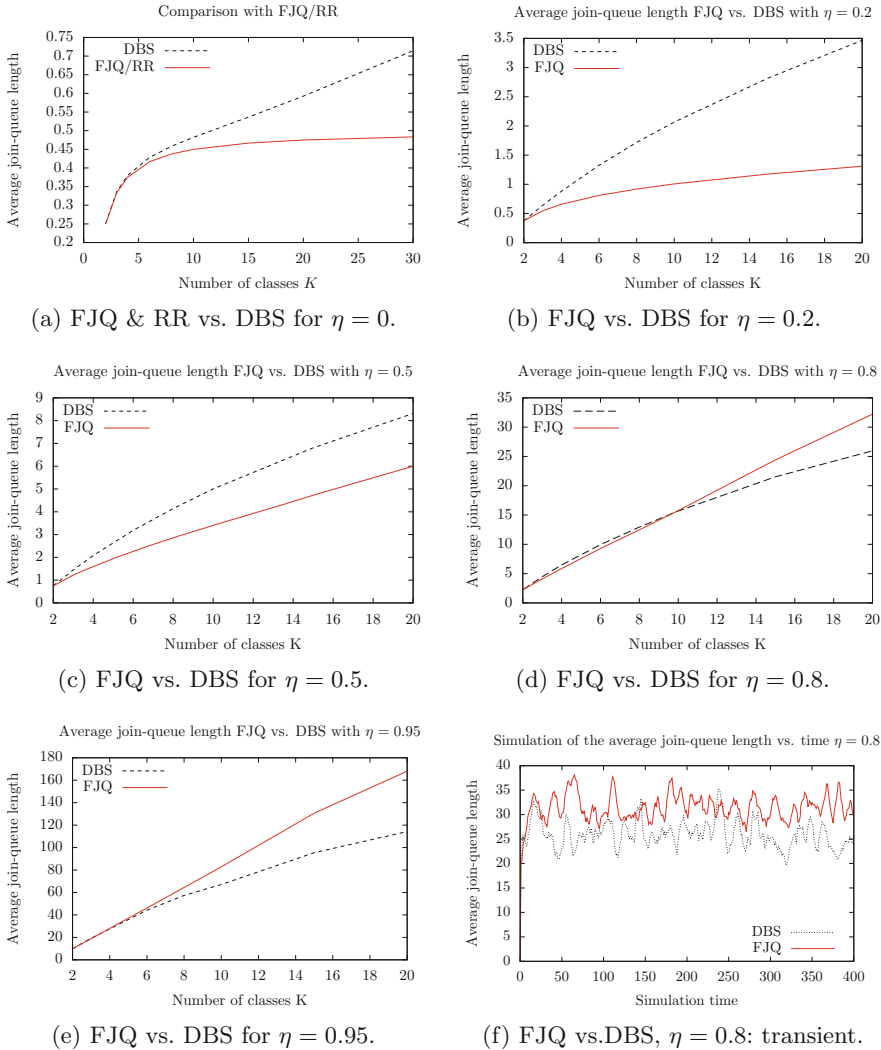
## 5 Comparison of the Scheduling Disciplines

In this section, we compare the three disciplines (RR, FJQ, DBS) under the assumptions A1–A3 described in Sect. 4. Let us consider the RR discipline. It can be proved that RR is stable if and only if  $\gamma_{min} = 0$ . Informally, one may see the case in which  $\gamma_{min} > 0$  as the superposition of two processes: one that serves all the classes with rate  $\gamma_{min}$  independently of the state of the join-queues, and the other which is RR with the minimum service rate set to 0 and the fastest is  $\gamma_{max} - \gamma_{min}$ . Since the former of these two processes is not positive recurrent, we have that the superposition cannot be positive recurrent. A formal proof can be provided by using the Lyupanov's conditions for the instability of Markov chain. If we consider  $\gamma_{min} = 0$ , we can derive the expected join-queue length of class  $k$ . In fact, for each job, queue  $k$  remains empty for an expected duration of  $k/\gamma$  units of time, and then it will contain one job for  $(K - k)/\gamma$ . Each join occurs after the service of class  $K$  and can be seen as a renewal point. Hence, the expected number of tasks in queue  $k$  is  $K - k$ . Thus, the expected number of tasks per queue is  $1/2 - 1/(2K)$ .

Let us consider the FJQ discipline. If  $\gamma_{min} = 0$  we can resort again to the renewal theory to easily derive the expected join-queue length. Let the renewal point be the join. Immediately after the join, we serve all the  $K$  classes with rate  $\gamma/K$  and the first task completion will occur after an exponential time with mean  $1/\gamma$ . For all this time, the join-queues are all empty. We can iterate this argument for the remaining  $K - 1$  tasks in service, and we clearly obtain the same results we have for RR. Therefore, if  $\gamma_{min} = 0$ , RR and FJQ are equivalent in terms of expected number of tasks in the join-queues (and hence also for what concerns the expected waiting time of a task). The analysis of FJQ when  $\gamma_{min} > 0$  is outside the scope of this paper and in order to perform the comparison we resort to stochastic simulations. All the simulations we show are based on 15 independent experiments, with a confidence interval at 95% whose relative error is below 2%. The warm up period has been determined by using the Welch's graphical method.

The first scenario we consider is when  $\gamma_{min} = 0$ , i.e., we can relocate all the computational resources. Figure 2a shows the plot of the average join-queue lengths as function of the number of classes  $K$ . We observe that for small  $K$  the performance of DBS is quite similar to that of the other two disciplines, but for large  $K$ , RR and FJQ perform better. However, it is important to notice that for large  $K$  the overhead and the complexity of the scheduling operations required by FJQ is high, therefore we can conclude that RR is the best policy if we are able to relocate the whole computational power.

The second scenario that we consider is when  $\gamma_{min} > 0$ . We consider  $\mu = 1$  and change the value of  $\eta$ . Recall that in this case the process underlying RR is not positive recurrent and hence we can compare only FJQ and DBS. Figures 2b–e compare the two disciplines for increasing values of  $\eta$ . Recall that larger values of  $\eta$  imply less relocatable resources and hence both the disciplines tend to perform worse. However, the plots of Figs. 2d–e are quite instructive. In fact, while one may think that FJQ is unconditionally better than DBS since it uses



**Fig. 2.** Numerical analysis of DBS scheduling policy.

information about the global state of the system, our simulations show that this is untrue, i.e., for values of  $\eta$  approaching 1 and large  $K$ , the expected join-queue length in DBS is smaller than that of FJQ. At the moment, we can support this statement only by resorting to the stochastic simulation of FJQ, since we have analytical results only for DBS. In Fig. 2f we show the Welch’s plot of the two disciplines on  $6 \cdot 10^6$  events. Intuitively, this happens because on the one hand DBS may assign less computational power to the empty join-queues than FJQ, but it also assigns less computational power to the longest join-queues. In the

case of stressed systems (when  $\eta$  is large), DBS' strategy results more convenient than the greedy approach of FJQ.

## 6 Conclusion

In this paper, we have presented a scheduling strategy, namely DBS, for biased processor sharing working in fork-join systems. The goal of DBS is that of assigning the computational resources to the tasks that are delaying the join operations. We have compared the performance of DBS with two other strategies: the 'round robin' (RR) and the 'feed the shortest join-queue' (FJQ). To this aim, we developed an analytical model whose steady-state distribution has a closed-form expression under some assumptions. Among these, the most relevant is the saturation of the model which defines a comparison scenario in heavy-load. We have shown that starting from the expression of the steady-state distribution we can algorithmically derive important average performance indices such as the expected number of tasks in a join-queue and the expected waiting time for a task in a join-queue. In conclusion, we showed that if a job consists of a small amount tasks  $K$ , then the performance of DBS and FJQ are very similar, while RR is applicable only in the case we can move among the tasks the whole available computational power. For large values of  $K$  and few relocatable resources, stochastic simulations show that DBS performs better than FJQ, while for systems with large amount of relocatable resources the opposite holds.

## References

1. Altman, E., Avrachenkov, K., Ayesta, U., Brown, P., Nunez-Queija, R.: A survey on discriminatory processor sharing. *Queueing Syst.* **53**(1–2), 53–63 (2006)
2. Baccelli, F., Liu, Z.: On the execution of parallel programs on multiprocessor systems: a queuing theory approach. *J. ACM* **37**(2), 373–414 (1990)
3. Baccelli, F., Massey, W.A., Towsley, D.: Acyclic fork-join queuing networks. *J. ACM* **36**(3), 615–642 (1989)
4. Bondald, T., Proutière, A.: Insensitivity in processor-sharing networks. *Perform. Eval.* **49**(1–4), 193–209 (2002)
5. Doncel, J., Ayesta, U., Brun, O., Prabhu, B.J.: A resource-sharing game with relative priorities. *Perform. Eval.* **79**, 287–305 (2014)
6. Fiorini, P.M., Lipsky, L.: Exact analysis of some split-merge queues. *SIGMETRICS Perform. Eval. Rev.* **43**(2), 51–53 (2015)
7. Flatto, L., Hahn, S.: Two parallel queues created by arrivals with two demands I. *SIAM J. Appl. Math.* **44**(5), 1041–1053 (1984)
8. Kelly, F.: *Reversibility and Stochastic Networks*. Wiley, New York (1979)
9. Kleinrock, L.: Time-shared systems: a theoretical treatment. *J. ACM* **14**(2), 242–261 (1967)
10. Kleinrock, L.: *Queueing Systems, Volume 1: Theory*. Wiley, New York (1975)
11. Lazowska, E.D., Zahorjan, J.L., Graham, G.S., Sevcick, K.C.: *Quantitative system performance: computer system analysis using queueing network models*. Prentice Hall, Englewood Cliffs (1984)

12. Marin, A., Rossi, S.: On the relations between lumpability and reversibility. In: Proceedings of the IEEE 22nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2014), pp. 427–432 (2014)
13. Marin, A., Rossi, S.: Fair workload distribution for multi-server systems with pulling strategies. *Perform. Eval.* **113**, 26–41 (2017)
14. Marin, A., Rossi, S.: On the relations between Markov chain lumpability and reversibility. *Acta Informatica* **54**(5), 447–485 (2017)
15. Marin, A., Rossi, S.: Power control in saturated fork-join queueing systems. *Perform. Eval.* **116**, 101–118 (2017)
16. Nelson, R., Tantawi, A.N.: Approximate analysis of fork/join synchronization in parallel queues. *IEEE Trans. Comput.* **37**(6), 739 (1986)
17. Olver, F.W., Lozier, D.W., Boisvert, R.F., Clark, C.W.: *NIST Handbook of Mathematical Functions*, 1st edn. Cambridge University Press, New York (2010)
18. Rizk, A., Poloczek, F., Ciucu, F.: Computable bounds in fork-join queueing systems. In: Proceedings of ACM SIGMETRICS, pp. 335–346 (2015)
19. Rizk, A., Poloczek, F., Ciucu, F.: Stochastic bounds in fork-join queueing systems under full and partial mapping. *Queueing Syst.* **83**(3–4), 261–291 (2016)
20. Tsimashenka, I., Knottenbelt, W.J., Harrison, P.G.: Controlling variability in split-merge systems and its impact on performance. *Ann. Oper. Res.* **239**, 569 (2014)
21. Wright, P.E.: Two parallel processors with coupled inputs. *Adv. Appl. Prob.* **24**, 986–1007 (1992)





# Probabilistic Model Checking for Continuous-Time Markov Chains via Sequential Bayesian Inference

Dimitrios Milios<sup>1,2(✉)</sup>, Guido Sanguinetti<sup>2,3</sup>, and David Schnoerr<sup>2,4</sup>

<sup>1</sup> Department of Data Science, EURECOM, Biot, France  
[milios@eurecom.fr](mailto:milios@eurecom.fr)

<sup>2</sup> School of Informatics, University of Edinburgh, Edinburgh, UK

<sup>3</sup> SynthSys, Centre for Synthetic and Systems Biology,  
University of Edinburgh, Edinburgh, UK

<sup>4</sup> Centre for Integrative Systems Biology and Bioinformatics,  
Department of Life Sciences, Imperial College London, London, UK

**Abstract.** Probabilistic model checking for systems with large or unbounded state space is a challenging computational problem in formal modelling and its applications. Numerical algorithms require an explicit representation of the state space, while statistical approaches require a large number of samples to estimate the desired properties with high confidence. Here, we show how model checking of time-bounded path properties can be recast exactly as a Bayesian inference problem. In this novel formulation the problem can be efficiently approximated using techniques from machine learning. Our approach is inspired by a recent result in statistical physics which derived closed-form differential equations for the first-passage time distribution of stochastic processes. We show on a number of non-trivial case studies that our method achieves both high accuracy and significant computational gains compared to statistical model checking.

**Keywords:** Bayesian inference · Model checking · Moment closure

## 1 Introduction

Probabilistic model checking of temporal logic formulae is a central problem in formal modelling, both from a theoretical and an applicative perspective [1, 2, 4–6, 22]. Classical algorithms based on matrix exponentiation and uniformisation are well-understood, and form the core routines of mature software tools such as PRISM [28], MRMC [26] and UPPAAL [7]. Nevertheless, the need to explicitly represent the state space makes their application to large systems problematic, or, indeed, theoretically impossible in the case of systems with unbounded state spaces, which appear frequently in biological applications.

Statistical model checking (SMC) approaches [37, 38] have emerged in recent years as a powerful alternative to exact techniques. Such methods provide a

Monte Carlo estimate of the desired probability by repeatedly sampling trajectories from the model. SMC can also provide probabilistic guarantees on the estimated probabilities, and, by choosing the number of simulations to be suitably large, one can reduce the uncertainty over the estimates arbitrarily.

While SMC offers a practical and flexible solution in many scenarios, its reliance on repeated simulation of the system makes it naturally computationally intensive. Although SMC can be trivially parallelized, the approach can still be computationally onerous for systems which are intrinsically expensive to simulate, such as systems with large agent counts or exhibiting stiff dynamics.

In this paper, we propose an alternative approach to solving the probabilistic model checking problem which draws on a recently proposed technique from statistical physics [33]. We show that the model checking problem is equivalent to a sequential Bayesian computation of the marginal likelihood of an auxiliary observation process. This marginal likelihood yields the desired time-bounded reachability probability, which is closely related to the eventually and globally temporal operators. We also expand the methodology to the case of the time-bounded until operator, thus covering a wide range of properties for temporal logics such as CSL [1, 2, 4–6]. The formulation of the model checking problem as a Bayesian inference method allows us to utilise efficient and accurate approximation methodologies from the machine learning community. In particular, we combine Assumed Density Filtering (ADF) [29, 31] with a moment-closure approximation scheme, which enables us to approximate the entire cumulative distribution function (CDF) of the *first* time that a time-bounded until property is satisfied by solving a small set of closed ordinary differential equations (ODEs) and low-dimensional integrals.

The rest of the paper is organised as follows. We discuss the related work in Sect. 2 and we provide some background material in Sect. 3. We then describe our new approach, highlighting both the links and differences to the recently proposed statistical physics method of [33] in Sect. 4. We consider four non-linear example systems of varying size and stiffness in Sect. 5.

## 2 Related Work

In recent years, the computational challenges of probabilistic model checking have motivated the development of approaches that rely on stochastic approximations as an alternative to both classical methods and SMC. In one of the earliest attempts, passage-time distributions were approximated by means of *fluid analysis* [24]. This framework was later extended to more general properties expressed as *stochastic probes* [15]. Fluid approximation has also been used to verify CSL properties for individual agents for large population models [9, 10]. In [11], a Linear Noise Approximation (LNA) was employed to verify not only local properties of individuals, but also global ones, which are given as the fraction of agents that satisfy a certain local specification. The verification of such local and global properties has been recently generalised for a wider class of stochastic approximations, including moment closure [13].

Regarding our work, one key difference with respect to these earlier approaches is that we consider global time-bounded until properties that characterise the behaviour of the system at the population level. In that sense, our approach is mostly related to [8, 12], which rely on the LNA to approximate the probability of global reachability properties. In particular, the LNA is used to obtain a Gaussian approximation for the distribution of the hitting time to the absorbing set [12]. The methodology is different in [8], where it is shown that the LNA can be abstracted as a time-inhomogeneous discrete-time Markov chain which can be used to estimate time-bounded reachability properties. However, this method approximates the unconstrained process, and needs to subsequently resort to space and time discretisation to approximate the desired probability.

### 3 Background

A Continuous-Time Markov Chain (CTMC) is a Markovian (i.e. memoryless) stochastic process that takes values on a countable state space  $\mathcal{S}$  and evolves in continuous time [18]. More formally:

**Definition 1.** *A stochastic process  $\{X(t) : t \geq 0\}$  is a Continuous-Time Markov Chain if it satisfies the Markov property, i.e. for any  $h \geq 0$ :*

$$p(X_{t+h} = j \mid X_t = i, \{X_\tau : 0 \leq \tau \leq t\}) = p(X_{t+h} = j \mid X_t = i) \quad (1)$$

A CTMC is fully characterised by its generator matrix  $Q$ , whose entries  $Q_{ij}$  denote the transition rate from state  $i$  to state  $j$ , for any  $i, j \in \mathcal{S}$  [32]. The dynamics of a CTMC are defined by the *master equation*, which is a system of coupled ODEs that describe how the probability mass changes over time for all states. For a CTMC with generator matrix  $Q$ , the master equation will be:

$$\frac{dP(t)}{dt} = P(t)Q \quad (2)$$

where  $P(t)$  is the transition probability matrix at time  $t$ ; the quantity  $P_{ij}(t) = p(X_t = j \mid X_{t_0} = i)$  denotes the probability to transition from state  $i$  at time  $t_0$  to state  $j$  at time  $t \geq t_0$ . The master equation is solved subject to initial conditions  $P(0)$ . Throughout this work, we shall consider CTMCs that admit a *population* structure, so that we can represent the state of a CTMC as a vector of non-negative integer-valued variables  $\mathbf{x} = \{X_1, \dots, X_N\}$ , that represent population counts for  $N$  different interacting entities.

#### 3.1 Moment Closure Approximation

For most systems, no analytic solutions to the master equation in (2) are known. If the state space  $\mathcal{S}$  is finite, (2) constitutes a finite system of ordinary differential equations and can be solved by matrix exponentiation. For many systems of practical interest however,  $\mathcal{S}$  is either infinite, or so large that the computational costs of matrix exponentiation become prohibitive.

*Moment closure methods* constitute an efficient class of approximation methods for certain types of master equations, namely if the elements  $Q_{ij}$  of the generator matrix are polynomials in the state  $\mathbf{x}$ . This is for example the case for population CTMC of mass action type which are frequently used to model chemical reaction networks [20]. In this case, one can derive ordinary differential equations for the moments of the distribution of the process. Unless the  $Q_{ij}$  are all polynomials in  $\mathbf{x}$  of order one or smaller, the equation for a moment of a certain order will depend on higher order moments, which means one has to deal with an infinite system of coupled equations. *Moment closure methods* close this infinite hierarchy of equations by truncating to a certain order. A popular class of moment closure methods does so by assuming  $P(t)$  to have a certain parametric form [36]. This then allows to express all moments above a certain order in terms of lower order moments and thus to close the equations for these lower order moments.

In this paper, we utilise the so-called *normal moment closure* which approximates the solution of the master equation by a multi-variate normal distribution by setting all cumulants of order greater than two to zero [21, 34, 35]. This class of approximations was recently used within a formal modelling context in [19].

### 3.2 Probabilistic Model Checking

The problem of probabilistic model checking of CTMCs is defined in the literature as the verification of a CTMC against Continuous Stochastic Logic (CSL) [1, 2, 4–6]. A CSL expression is evaluated over the states of a CTMC. In the original specification [1], the syntax of a CSL formula is described by the grammar:

$$\phi ::= \text{tt} \mid \alpha \mid \neg\phi \mid \phi_1 \wedge \phi_2 \mid \mathcal{P}_{\bowtie p}(\Phi)$$

where  $\phi$  is a state-formula, and  $\Phi$  is a path-formula, i.e. it is evaluated over a random trajectory of the Markov chain. An atomic proposition  $\alpha$  identifies a subset of the state space; in this paper, we consider atomic propositions to be linear inequalities on population variables. The probabilistic operator  $\mathcal{P}_{\bowtie p}(\Phi)$  allows reasoning about the probabilities of a path-formula  $\Phi$ :

$$\Phi ::= \mathbf{X}\phi \mid \phi_1 \mathbf{U} \phi_2 \mid \phi_1 \mathbf{U}^{[t_1, t_2]} \phi_2$$

$\mathcal{P}_{\bowtie p}(\Phi)$  asserts whether the probability that  $\Phi$  is satisfied meets a certain bound expressed as  $\bowtie p$ , where  $\bowtie \in \{\leq, \geq\}$  and  $p \in [0, 1]$ . In order to evaluate the probabilistic operator, we need to calculate the satisfaction probability for a path-formula  $\Phi$ , which involves one of three temporal operators: next  $\mathbf{X}$ , unbounded until  $\mathbf{U}$ , and time-bounded until  $\mathbf{U}^{[t_1, t_2]}$ .

For a finite CTMC, it is well-known that evaluating the probability of  $\mathbf{X}\phi$  is reduced to matrix/vector multiplication, while evaluating the unbounded until  $\phi_1 \mathbf{U} \phi_2$  requires solving a system of linear equations [4]. The time-bounded until operator can also be evaluated numerically via an iterative method that relies on uniformisation [4]. This process may have a prohibitive computational cost if the size of the state space is too large. For systems with unbounded state space,

the only option to estimate the time-bounded until probabilities is by the means of stochastic simulation [37, 38], which also has a high computational cost.

Other temporal operators can be expressed as special cases of the until operator. For the time-bounded eventually operator we have:  $\mathbf{F}^{[t_1, t_2]} \phi = \mathbf{tt} \mathbf{U}^{[t_1, t_2]} \phi$ , while for the globally operator we have:  $\mathbf{G}^{[t_1, t_2]} \phi = \neg \mathbf{F}^{[t_1, t_2]} \neg \phi$ . The latter two operators formally describe the problem of time-bounded reachability.

## 4 Methodology

Given a property of the form  $\Phi = \phi_1 \mathbf{U}^{[0, t]} \phi_2$ , our goal is to approximate the cumulative probability of reaching  $\phi_2$  at  $\tau \leq t$ , while satisfying  $\phi_1$  until then.

### 4.1 Time-Bounded Reachability as Bayesian Inference

We first consider the problem of reachability, which is closely related to the eventually temporal operator  $\mathbf{F}^{[0, t]} \phi$ . If  $S_\phi$  denotes the set of states that satisfy the formula  $\phi$ , then we are interested in the probability that  $S_\phi$  is reached for the first time; this quantity is also known in the literature as *first-passage time*.

Building upon [16, 17] Schnoerr et al. [33] have recently formulated time-bounded reachability as a Bayesian inference problem. Using this formulation, they proposed a method where the entire distribution of first-passage times can be approximated by taking advantage of some well-established methodologies in the Bayesian inference and statistical physics literature. In the current section, we revise the approach of Schnoerr et al. [33] for reachability, while in Sect. 4.2 we expand the method to the more general case of time-bounded until.

In the Markov chain literature [32], the states in the set  $S_\phi$  are often called the *absorbing* states. Let  $C = \mathcal{S} \setminus S_\phi$  denote the set of *non-absorbing* states. The cumulative probability for the system to reach an absorbing state at or before time  $t$  is equal to 1 minus the probability of the system having remained in  $C$  until  $t$ . Schnoerr et al's insight was to formulate this probability in terms of a Bayesian computation problem. Consider an auxiliary binary observation  $c(t)$  process which evaluates to 1 whenever the system is in the non-absorbing set  $C$  and 0 otherwise. The pair  $\{c(t), \mathbf{x}_t\}$  constitutes a *hidden Markov model* (HMM) in continuous time; the required cumulative probability would then correspond to the marginal likelihood of observing a string of all 1s as output of the HMM. Computing this marginal likelihood is a central and well studied problem in machine learning and statistics.

Even in this novel formulation, the problem is generally still intractable. To make progress, we first discretise the time interval  $[0, t]$  into time points  $\mathcal{T} = \{t_0 = 0, \dots, t_N = t\}$  with spacing  $t/N$ . For the process  $\mathbf{x}_{t_i}$  at time  $t_i$  being in  $C$  we thus have the observation model  $p(C_{t_i} | \mathbf{x}_{t_i}) = 1$  if  $\mathbf{x}_{t_i} \in C$  and zero otherwise. Note that  $p(C_{t_i} | x_{t_i})$  is the distribution of the observation process  $c(t)$ , i.e.  $c(t_i) \sim p(C_{t_i} | x_{t_i})$ . The marginal likelihood  $Z_{[0, t]}$  of having remained in  $C$  for all  $t_i \in \mathcal{T}$  factorises as

$$Z_{[0,t]} = p(C_{t_0}) \prod_{i=1}^N p(C_{t_i} | C_{<t_i}) \tag{3}$$

where we introduced the notation  $C_{<t_i} \equiv C_{t_{i-1}, \dots, t_0}$ . The factors of the rhs in (3) can be computed iteratively as follows. Let  $\mathbf{x}_0$  be the initial condition of the process. Suppose that the system did not transition into the absorbing set until time  $t_{i-1}$  (that is, the process remained in  $C$ ), and that the state distribution conditioned on this observations is  $p(\mathbf{x}_{t_{i-1}} | C_{<t_i}, \mathbf{x}_0)$ . We can solve the system forward in time up to time  $t_i$  to obtain the predictive distribution  $p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)$ , which will serve as a prior, and combine it with the likelihood term  $p(C_{t_i} | \mathbf{x}_{t_i})$  that the process has remained in  $C$  at time  $t_i$ .

We can then define a posterior over the state space by applying the Bayes rule as follows:

$$p(\mathbf{x}_{t_i} | C_{\leq t_i}, \mathbf{x}_0) = \frac{p(C_{t_i} | \mathbf{x}_{t_i}) p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)}{p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)} \tag{4}$$

The likelihood  $p(C_{t_i} | \mathbf{x}_{t_i})$  represents the probability that the process does not leave  $C$  at time  $t_i$ . The prior denotes the state space probability considering that the process had remained in  $C$  for time  $< t_i$ . The posterior then will be the state space distribution after observing that the Markov process has remained in  $C$  at the current step.

The evidence  $p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)$  in (4) is a factor in the rhs of (3). It can be obtained by marginalising the joint probability  $p(C_{t_i}, \mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)$  over  $\mathbf{x}_{t_i}$ :

$$p(C_{t_i} | C_{<t_i}, \mathbf{x}_0) = \int_S p(C_{t_i} | \mathbf{x}_{t_i}) p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0) d\mathbf{x}_{t_i} \tag{5}$$

The process described above is a Bayesian formulation for the introduction of absorbing states. By multiplying by the likelihood, we essentially remove the probability mass of transitioning to a state in  $S_\phi$ ; the remaining probability mass (the evidence) is simply the probability of remaining in  $C$ . Therefore, the probability of transitioning to  $S_\phi$  for the first time at time  $t_i$  is the complement of the evidence:

$$p(S_{t_i}^\phi | C_{<t_i}, \mathbf{x}_0) = 1 - p(C_{t_i} | C_{<t_i}, \mathbf{x}_0) \tag{6}$$

Thus, Eq. (6) calculates the first-passage time probability for any  $t_i \in \mathcal{T}$ . Note that this approach neglects the possibility of the process leaving from and returning to region  $C$  within on time step. The time spacing thus needs to be chosen small enough for this to be a good approximation.

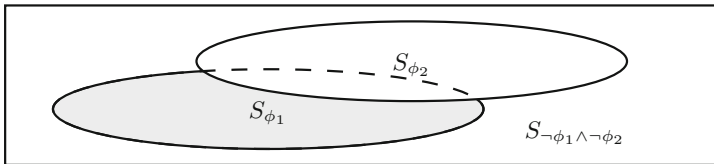
Schnoerr et al. [33] further approximated the binary observation likelihood  $p(C_{t_i} | \mathbf{x}_{t_i})$  by a soft, continuous loss function. This allowed them to take the continuum limit of vanishing time steps which in turn allows to approximate the evidence  $p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)$  by solving a set of ODEs. In this work, we keep the binary, discontinuous observation process and keep time discrete, which allows us to extend the framework from [33] to the time-bounded until operator.

## 4.2 The Time-Bounded Until Operator

Consider the time-bounded property  $\phi_1 \mathbf{U}^{[0,t]}\phi_2$  which will be satisfied if a state in  $S_{\phi_2}$  is reached up to time  $t$  and the stochastic process has remained in  $S_{\phi_1}$  until then. Assuming that  $\phi_1$  is satisfied up to some  $t_i \leq t$ , there are three distinct possibilities regarding the satisfaction of the until property:

- it evaluated as false if we have  $\mathbf{x}_{t_i} \notin S_{\phi_1}$  and  $\mathbf{x}_{t_i} \notin S_{\phi_2}$  simultaneously,
- the property is evaluated as true if  $\mathbf{x}_{t_i} \in S_{\phi_2}$ ,
- otherwise the satisfaction of the property is undetermined up to time  $t_i$ .

These possibilities correspond to three non-overlapping sets of states:  $S_{\neg\phi_1 \wedge \neg\phi_2}$ ,  $S_{\phi_2}$  and  $S_{\phi_1} \setminus S_{\phi_2}$  accordingly, as seen in Fig. 1.



**Fig. 1.** The until formula  $\phi_1 \mathbf{U}^{[0,t]}\phi_2$  is trivially satisfied for states in  $S_{\phi_2}$ , while it is not satisfied for any state in  $S_{\neg\phi_1 \wedge \neg\phi_2}$ . For the rest of the states  $C = S_{\phi_1} \setminus S_{\phi_2}$  (i.e. the grey area above) the property satisfaction is not determined. Assuming that the CTMC state has remained in  $C$ , we define a reachability problem to the union  $S_{\phi_2} \cup S_{\neg\phi_1 \wedge \neg\phi_2}$ . In contrast with the standard reachability problem, the probability of  $S_{\phi_2}$  is of interest only, which is a subset of the absorbing states.

In order to calculate the satisfaction probabilities for any time  $t_i \leq t$ , we assume that the property has not been determined before  $t_i$ . That means that the Markov process has remained in the set  $C = S_{\phi_1} \setminus S_{\phi_2}$ , which is marked as the grey area in Fig. 1. The Bayesian formulation of reachability discussed in Sect. 4.1 can be naturally applied to the problem of reaching the union  $S_{\phi_2} \cup S_{\neg\phi_1 \wedge \neg\phi_2}$ . The prior term  $p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)$  denotes the state distribution given that the property remained undetermined before  $t_i$ . The likelihood term  $p(C_{t_i} | \mathbf{x}_{t_i})$  indicates whether the Markov process has remained in the non-absorbing set  $C = S_{\phi_1} \setminus S_{\phi_2}$  at  $t_i$ . Finally, the posterior given by (4) will be the state space distribution after observing that the property has remained undetermined at the last step.

In contrast with the reachability problem however, once the absorbing set is reached, we only know that the formula has been determined, but we do not know whether it has been evaluated as true or false. More specifically, the evidence  $p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)$  as given by Eq. (5) represents the probability that the satisfaction has remained undetermined at time  $t_i$ . Although the negation of the evidence was sufficient to resolve the reachability probability as in Eq. (6), we are now interested only in a subset of the absorbing states. At a particular time  $t_i$  we have to calculate the probability of reaching  $S_{\phi_2}$  explicitly. This is given by the overlap mass of the prior process  $p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)$  and probability of

transitioning into  $S_{\phi_2}$ . Given that  $p(S_{t_i}^{\phi_2} | \mathbf{x}_{t_i}) = 1$ , if  $\mathbf{x}_{t_i} \in S_{\phi_2}$  at time  $t_i$  and zero otherwise, we have:

$$p(S_{t_i}^{\phi_2} | C_{<t_i}, \mathbf{x}_0) = \int_S p(S_{t_i}^{\phi_2} | \mathbf{x}_{t_i}) p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0) d\mathbf{x}_{t_i} \tag{7}$$

which is the probability of transitioning to  $S_{\phi_2}$  at time  $t_i$ , while remaining in  $C$  until then. The effect of  $p(S_{t_i}^{\phi_2} | \mathbf{x}_{t_i})$  is essentially a truncation of the state space; the first-passage probability at  $t_i$  is simplified as follows:

$$p(S_{t_i}^{\phi_2} | C_{<t_i}, \mathbf{x}_0) = \int_{\mathbf{x}_{t_i} \in S_{\phi_2}} p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0) d\mathbf{x}_{t_i} \tag{8}$$

Considering a Gaussian approximation for  $p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)$ , as we discuss in the next section, and given that the state formula  $\phi_2$  is a conjunction of linear inequalities, Eq. (8) can be easily calculated by numerical routines.

The Bayesian formulation that we introduce has essentially the same effect as the traditional probabilistic model checking methods [5]. The probability of the until operator is usually evaluated by first introducing the set of absorbing states  $S_{\phi_2} \cup S_{\neg\phi_1 \wedge \neg\phi_2}$ , and then calculating the probability of reaching the set  $S_{\phi_2}$ , which is a subset of the absorbing states. The advantage of the formulation presented here is that it allows us to leverage well-established machine learning methodologies, as we see in the section that follows.

### 4.3 Gaussian Approximation via Assumed Density Filtering

The Bayesian formulation as described so far does not involve any approximations apart from time discretisation. In fact for a discrete-state system, both the prior and the likelihood terms (i.e.  $p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0)$  and  $p(C_{t_i} | \mathbf{x}_{t_i})$  equivalently) will be discrete distributions in (4). Therefore, quantities such as the evidence in (5) and the probability of reaching  $S_{\phi_2}$  in Eq. (8) can be calculated exactly, as the integrals reduce to summations. However, if the size of the state space is too large or unbounded, this process can be computationally prohibitive. The formulation presented above allows us to derive an efficient approximation method that relies on approximating the discrete process by a continuous one.

We adopt a moment closure approximation scheme where all cumulants of order three or larger are set to zero, which corresponds to approximating the single-time distribution of the process by a Gaussian distribution. As described in Sect. 3.1, the moment closure method results in a system of ODEs that describe the evolution of the expected values and the covariances of the population variables in a given CTMC. At any time  $t_i$ , the state distribution is approximated by a Gaussian with mean  $\mu_{t_i}$  and covariance  $\Sigma_{t_i}$ :

$$p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t_i}; \mu_{t_i}, \Sigma_{t_i})$$

The evidence is the probability mass of non-absorbing states; i.e. it is observed that the process has remained within  $C$ . Since  $C$  is identified by linear inequalities on the population variables, both the evidence in Eq. (5) and the probability



mass in the target set in (8) can be estimated by numerically solving the integrals. There are many software routines readily available to calculate the CDF of multivariate Gaussian distributions by numerical means.

Nevertheless, the posterior in Eq. (4) is not Gaussian and we introduce a Gaussian approximation. It is proven that ADF minimises the KL divergence between the true posterior and the approximating distribution, subject to the constraint that the approximating distribution is Gaussian [29, 31]. Considering the prior  $\mathcal{N}(\mathbf{x}_{t_i}; \mu_{t_i}, \Sigma_{t_i})$ , the ADF updates [17] will be:

$$\tilde{\mu}_{t_i} = \mu_{t_i} + \Sigma_{t_i} \partial_{\mu_{t_i}} \log Z_{t_i} \quad (9)$$

$$\tilde{\Sigma}_{t_i} = \Sigma_{t_i} + \Sigma_{t_i} \partial_{\mu_{t_i}^2} \log Z_{t_i} \Sigma_{t_i} \quad (10)$$

where the evidence  $Z_{t_i} = p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)$  is equal to the mass of the truncated Gaussian that corresponds to the non-absorbing states  $C$  at time  $t_i$ . The dimensionality of the Gaussians is equal to the number of distinct populations in the system; this is generally small, meaning that computations of truncated Gaussian integrals can be carried out efficiently. A detailed exposition can be found in the archive version of the paper [30].

#### 4.4 Algorithm

Algorithm 1 is an instantiation of model checking via sequential Bayesian inference (MC-SBI). The algorithm evaluates the probability that a property  $\Phi = \phi_1 \mathbf{U}^{[0,t]} \phi_2$  is satisfied for a sequence of time points  $\mathcal{T} = \{t_0 = 0, t_1, \dots, t_N = t\}$ , thus approximating the CDF of the *first* time that  $\Phi$  is satisfied.

In the beginning of each iteration at line 5, we calculate the probability  $\pi_i$  that  $\Phi$  is satisfied at  $t_i$ . In lines 6–8, we calculate the posterior state distribution, assuming that  $\Phi$  has not been determined at the current step. Finally, the state distribution is propagated by the moment closure ODEs; the new state probabilities  $p(\mathbf{x}_{t_{i+1}} | C_{<t_{i+1}}, \mathbf{x}_0)$  will serve as the prior in the next iteration.

It is useful at this stage to pause and consider the differences from the first-passage time algorithm proposed in [33]: both papers share the same insight that reachability properties can be computed via Bayesian inference. However, the resulting algorithms are quite different. The crucial technical difficulty when considering formulae involving an until operator is the need to evaluate the probability of transitioning into the region identified by the second formula  $S_{\phi_2}$ . It is unclear how to incorporate such a computation within the continuous-time differential equations approach of [33], which dictates the choice of pursuing a time discretisation approach here. The time discretisation however brings the additional benefit that we can evaluate *exactly* the moments of the Bayesian update in step 8 of Algorithm 1, thus removing one of the sources of error in [33] (at a modest computational cost, as the solution of ODEs is generally faster than the iterative approach proposed here).

---

**Algorithm 1.** Model Checking via Sequential Bayesian Inference
 

---

**Require:** CTMC with initial state  $\mathbf{x}_0$ , property  $\Phi = \phi_1 \mathbf{U}^{[0,t]} \phi_2$ , time sequence  $\mathcal{T} = \{0, t_1, \dots, t\}$   
**Ensure:** Probabilities  $\{\pi_0, \dots, \pi_N\}$  that approximate the CDF of the time that  $\Phi$  is satisfied

- 1: Define  $C = S_{\phi_1} \setminus S_{\phi_2}$ , where the satisfaction of  $\Phi$  is not determined
- 2: Set the initial prior:  $p(\mathbf{x}_{t_0} | C_{<t_0}, \mathbf{x}_0) \leftarrow \mathcal{N}(\mathbf{x}_{t_0}; \mu_{t_0}, \Sigma_{t_0})$
- 3: Initialise the probability that  $\Phi$  is not determined:  $p(C_{<t_0}, \mathbf{x}_0) \leftarrow 1$
- 4: **for**  $i \leftarrow 0$  **to**  $N$  **do**
- 5:   Calculate the probability that  $\Phi$  is satisfied for first time at  $t_i$ :

$$\pi_i \leftarrow p(C_{<t_i}, \mathbf{x}_0) \times \int_{\mathbf{x}_{t_i} \in S_{\phi_2}} p(\mathbf{x}_{t_i} | C_{<t_i}, \mathbf{x}_0) d\mathbf{x}_{t_i}$$

- 6:   Calculate the evidence  $p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)$  according to Equation (5)
- 7:   Calculate the probability that  $\Phi$  is not determined in the next step:

$$p(C_{<t_{i+1}}, \mathbf{x}_0) \leftarrow p(C_{<t_{i+1}}, \mathbf{x}_0) \times p(C_{t_i} | C_{<t_i}, \mathbf{x}_0)$$

- 8:   Calculate the posterior mean  $\tilde{\mu}_{t_i}$  and covariance  $\tilde{\Sigma}_{t_i}$  according to (9) and (10) respectively
- 9:   Considering  $\tilde{\mu}_{t_i}$  and  $\tilde{\Sigma}_{t_i}$  as initial conditions, use moment closure ODEs to obtain:  $\mu_{t_{i+1}}$  and  $\Sigma_{t_{i+1}}$
- 10:   Set the prior of the next step:

$$p(\mathbf{x}_{t_{i+1}} | C_{<t_{i+1}}, \mathbf{x}_0) \leftarrow \mathcal{N}(\mathbf{x}_{t_{i+1}}; \mu_{t_{i+1}}, \Sigma_{t_{i+1}})$$

11: **end for**

---

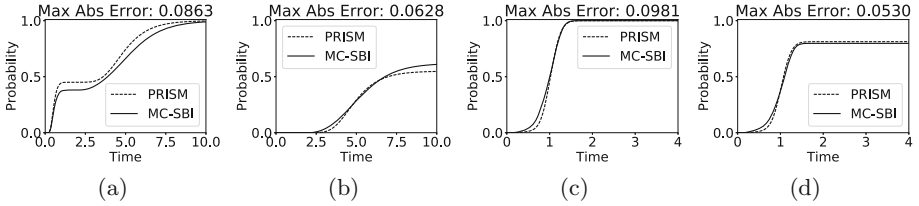
## 5 Examples

In this section, we demonstrate the potential of our approach on a number of examples. More specifically, we report for each example the calculated CDF for the time that a formula  $\Phi = \phi_1 \mathbf{U}^{[0,t]} \phi_2$  is first satisfied. Additionally for each until property, we also report the CDF of the first-passage time to the absorbing set; this corresponds to the eventually formula  $\mathbf{F}^{[0,t]}(\phi_2 \vee \neg \phi_1 \wedge \neg \phi_2)$ , following the discussion of Sect. 4.2.

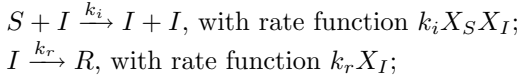
As a baseline reference, we use the PRISM Model Checker [28], which is a well-established tool in the literature. For a time-bounded until property  $\Phi$ , PRISM is capable of estimating its satisfaction probability by considering the following variation of the probabilistic operator  $\mathcal{P}_{=?}(\Phi)$ . The result of  $\mathcal{P}_{=?}(\Phi)$  denotes the probability that  $\Phi$  has been satisfied at any  $\tau \leq t$ , thus it can be directly compared to our approach. In particular, PRISM offers numerical verification of time-bounded until properties that relies on the uniformisation method [4]. We make use of numerical verification when possible, but for more complex models we resort to SMC.

### 5.1 An Epidemiology Model

We consider a SIR model, whose state is described by three variables that represent the number of susceptible ( $X_S$ ), infected ( $X_I$ ), and recovered ( $X_R$ ) individuals in a population of fixed size. The dynamics are described by the reactions:



**Fig. 2.** First-passage time results for the SIR model: (a) the CDF of first-passage times into the absorbing states for  $\varphi_1$ , (b) CDFs of first-passage times for the until formula  $\varphi_1$ , (c) the CDF of first-passage times into the absorbing states for  $\varphi_2$ , (d) CDFs of first-passage times for the until formula  $\varphi_2$ .



Considering initial state  $[X_S = 40, X_I = 10, X_R = 0]$ , the reachable state space as reported by PRISM involves 1271 states and 2451 transitions, which is a number small enough to allow the use of numerical verification.

We consider two properties: the first property states whether the infected population remains under a certain threshold until the extinction of the epidemic:

$$\varphi_1 = X_I < 30 \mathbf{U}^{[0, t_1]} X_I = 0 \quad (11)$$

where  $t_1 = 10$ . Also, we consider a property that involves more than one species:

$$\varphi_2 = X_S > 1 \mathbf{U}^{[0, t_2]} X_I < X_R \quad (12)$$

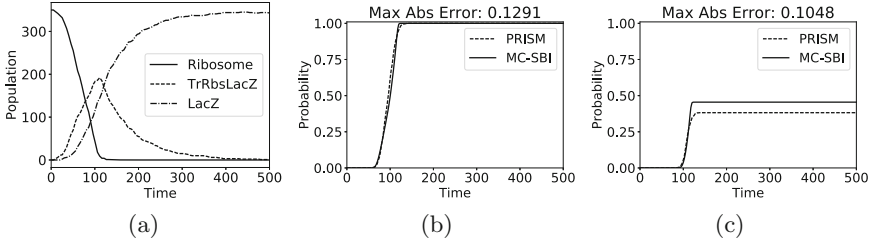
where  $t_2 = 4$ . The random variables  $[X_S, X_I, X_R, X_I - X_R]$  follow a joint Gaussian distribution, which is compatible with the assumptions of our approach.

We have used Algorithm 1 to approximate the CDF of the time that  $\varphi_1$  and  $\varphi_2$  are first satisfied on a sequence  $\mathcal{T}$  of 200 time-points. We have also used the hybrid engine of PRISM in order to produce accurate estimates of the satisfaction probabilities of  $\varphi_1$  and  $\varphi_2$ , for  $t_1 \in [0, 10]$  and  $t_2 \in [0, 4]$  respectively.

The calculated CDFs for  $\varphi_1$  are summarised in Fig. 2b, while in Fig. 2a we report the CDFs of the first-passage time into its absorbing set. Similarly, the CDFs for  $\varphi_2$  are reported in 2d, and the CDFs of the corresponding absorbing set can be found in Fig. 2c. In both cases the CDFs calculated by our approach (MC-SBI) are close to the numerical solutions of PRISM.

## 5.2 LacZ - A Model of Prokaryotic Gene Expression

We consider the model of LacZ protein synthesis in *E. coli* that first appeared in [27] and has been used as a model checking benchmark [14]. The model consists of 12 species and 11 reactions; its full specification can be found in [30]. We are interested in three variables:  $X_{Ribosome}$  for the population of ribosomes,



**Fig. 3.** First-passage time results for the LacZ model: (a) sample trajectory, (b) the CDF of first-passage times into the absorbing states for the property  $\varphi_3$ , (c) CDFs of first-passage times for the until formula  $\varphi_3$ .

$X_{TrRbsLacZ}$  which represents the population of translated sequences, and  $X_{LacZ}$  representing the molecules of protein produced. The following property:

$$\varphi_3 = (X_{Ribosome} > 0 \wedge X_{TrRbsLacZ} < 200) \mathbf{U}^{[0,500]} X_{LacZ} > 150 \quad (13)$$

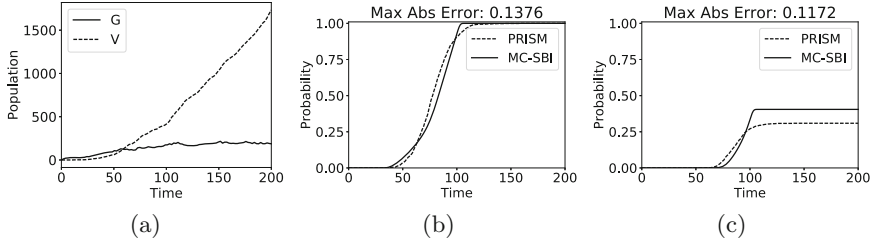
monitors whether both  $X_{Ribosome}$  and  $X_{TrRbsLacZ}$  satisfy certain conditions until the LacZ protein produced reaches a specified threshold (i.e.  $X_{LacZ} > 150$ ). A randomly sampled trajectory can be seen in Fig. 3a.

We have attempted to explore the reachable state space of the model using the hybrid engine of PRISM; that involved more than 26 trillions of states and 217 trillions of transitions. It is fair to state that numerical methodologies can be ruled out for this example. Thus we compare our approach with SMC as implemented in PRISM, where 1000 samples were used; the confidence interval for the results that follow is  $\pm 0.039$ , based on 99.0% confidence level.

Figure 3 summarises the calculated first-passage time CDFs evaluated on a sequence of 200 time-points. In Fig. 3b we see that the moment closure method resulted in a particularly accurate approximation of the first-passage time distribution for the absorbing states. Regarding the distribution of  $\varphi_3$ , the results of MC-SBI and PRISM’s SMC seem to be in agreement (Fig. 3c); however that our method overestimates the final probability of satisfying  $\varphi_3$ .

### 5.3 A Stiff Viral Model

Stiffness is a computational issue in many chemical reaction systems which arises when some reactions occur much more frequently than others. This group of fast reactions dominates the computational time, and thus renders simulation particularly expensive. As an example of a stiff system, we consider the model of viral infection in [23]. The model state is described by four variables: the population of the viral template  $X_T$ , the viral genome  $X_G$ , the viral structural protein  $X_S$ , and  $X_V$  that captures the number of viruses produced. For the initial state we set  $X_T = 10$ , and the rest of the variables to zero. The reactions that determine the dynamics can be found in [30]. The model state space is unbounded, therefore we resort to the SMC capabilities of PRISM to evaluate our



**Fig. 4.** First-passage time results for the Viral model: (a) sample trajectory, (b) the CDF of first-passage times into the absorbing states for the property  $\varphi_4$ , (c) CDFs of first-passage times for the until formula  $\varphi_4$ .

approach. The SMC used 1000 samples, resulting in confidence interval  $\pm 0.038$ , based on 99.0% confidence level.

Figure 4a depicts a random trajectory that shows the evolution of the viral genome  $X_G$  and the virus population  $X_V$  over time. We see that  $X_G$  slowly increases until it apparently reaches a steady-state and fluctuates around the value 200, while  $X_V$  continues to increase at a non-constant rate. In this example, we shall monitor whether the viral genome remains under the value of 200 until the virus population reaches a certain threshold:

$$\varphi_4 = X_G < 200 \mathbf{U}^{[0,200]} X_V > 500 \quad (14)$$

The results of Fig. 4 show that our method did not capture the CDFs as well as in the previous two examples. However, considering that our method is four orders of magnitude faster than SMC (cf. Table 1), it still gives a reasonably good approximation, particularly in the case of the eventually value. Again, we have considered a sequence of length 200.

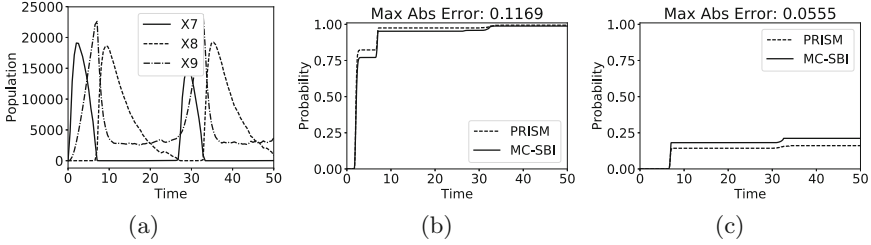
## 5.4 A Genetic Oscillator

Finally, we consider the model of a genetic oscillator in [3] consisting of 9 species and 16 reactions. The original model is defined in terms of concentrations; in order to convert the specification in terms of molecular populations, we consider a volume  $V = 1/6.022 \times 10^{-22}$ . The full model description can be found in [30]. We consider an initial state where  $X_1 = 10$ ,  $X_3 = 10$  and the rest of the variables are equal to 1. As we can see in the random trajectory in Fig. 5a, we have a system too large to apply traditional model checking methods.

We focus on variables  $X_7$  and  $X_9$ ; the following property monitors whether  $X_7$  remains under 19000 until  $X_9$  exceeds the value of 24000:

$$\varphi_5 = X_7 < 19000 \mathbf{U}^{[0,50]} X_9 > 24000 \quad (15)$$

In this example, we evaluated the CDF on a sequence  $\mathcal{T}$  of length 2000. We compare against the SMC algorithm in PRISM using 1000 samples, which resulted



**Fig. 5.** First-passage time results for the genetic oscillator model: (a) sample trajectory, (b) the CDF of first-passage times into the absorbing states for the property  $\varphi_5$ , (c) CDFs of first-passage times for the until formula  $\varphi_5$ .

in confidence interval  $\pm 0.030$ , based on 99.0% confidence level. Figure 5 shows a good approximation of the rather unusual first-passage time CDFs for both the absorbing states (Fig. 5b) and the  $\varphi_5$  property (Fig. 5c).

### 5.5 A Note on the Execution Times

Table 1 summarises the execution times for our method (MC-SBI) and statistical model checking (SMC). We have used numerical verification as implemented in PRISM for the SIR model only. For the other examples, the state space is too large to use an explicit representation, and we report the simulation running times only. The numerical approach is much faster when this is applicable. However, the computational savings for MC-SBI are obvious for the more complicated examples, in particular the viral model and the genetic oscillator.

We have used StochDynTools [25] to derive the moment closure approximations automatically. The CDFs have been evaluated on a sequence of 200 points for all models except for the genetic oscillator, where 2000 points were used.

**Table 1.** Execution times in seconds for model checking via sequential Bayesian inference (MC-SBI) and model checking in PRISM ( $10^4$  samples were used for SMC).

Model	MC-SBI	PRISM (Numerical)	PRISM (SMC)
SIR	8 s	$\sim 1$ s	$\sim 1$ s
LacZ	38 s	N/A	46 s
Viral	8 s	N/A	24875 s
Genetic oscillator	87 s	N/A	20707 s

## 6 Conclusions

We have presented a novel approach to the classical model checking problem based on a reformulation as a sequential Bayesian inference problem. This reformulation is exact up to time-discretisation errors; it was originally suggested in

[33] for reachability problems, and was extended in the present work to general CSL formulae including time-bounded Until operators. Apart from its conceptual appeal, this reformulation is important because it enables us to obtain an approximate solution using efficient and highly accurate tools from machine learning. Our method leverages a class of analytical approximations to CTMCs known as moment closures, which enable an efficient computation of the process marginal statistics.

We have shown on a number of diverse case studies that our method achieves excellent accuracy with significantly reduced computational costs compared to SMC. Nevertheless, our algorithm requires some approximations to the underlying stochastic process. The first approximation is the adoption of a time discretisation; this is a controllable approximation and can be rendered arbitrarily precise by reducing the time step (at a computational cost that grows linearly with the number of steps). The second approximation consists in propagating forward the first two moments of the process via moment closure and ADF. Approximation quality in this case is system dependent. Several studies have examined the problem of convergence of moment closure approximations [34, 35], however, to the best of our knowledge, error bounds for such approximations are an open problem in the mathematics of stochastic processes. Despite such issues, we believe that the reformulation of model checking problems in terms of Bayesian inference has the potential to open the door to a new class of approximate algorithms to attack this classic problem in computer science.

## References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996). [https://doi.org/10.1007/3-540-61474-5\\_75](https://doi.org/10.1007/3-540-61474-5_75)
2. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. ACM Trans. Comput. Logic **1**(1), 162–170 (2000)
3. Azunre, P., Gomez-Urbe, C., Verghese, G.: Mass fluctuation kinetics: analysis and computation of equilibria and local dynamics. IET Syst. Biol. **5**(6), 325–335 (2011)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Model checking continuous-time Markov chains by transient analysis. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 358–372. Springer, Heidelberg (2000). [https://doi.org/10.1007/10722167\\_28](https://doi.org/10.1007/10722167_28)
5. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.-P.: Automated performance and dependability evaluation using model checking. In: Calzarossa, M.C., Tucci, S. (eds.) Performance 2002. LNCS, vol. 2459, pp. 261–289. Springer, Heidelberg (2002). [https://doi.org/10.1007/3-540-45798-4\\_12](https://doi.org/10.1007/3-540-45798-4_12)
6. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)
7. Behrmann, G., David, A., Larsen, K.G., Hakansson, J., Petterson, P., Yi, W., Hendriks, M.: UPPAAL 4.0. In: Proceedings of QEST 2006, pp. 125–126. IEEE Computer Society (2006)

8. Bortolussi, L., Cardelli, L., Kwiatkowska, M., Laurenti, L.: Approximation of probabilistic reachability for chemical reaction networks using the linear noise approximation. In: Agha, G., Van Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 72–88. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-43425-4\\_5](https://doi.org/10.1007/978-3-319-43425-4_5)
9. Bortolussi, L., Hillston, J.: Fluid model checking. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 333–347. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32940-1\\_24](https://doi.org/10.1007/978-3-642-32940-1_24)
10. Bortolussi, L., Hillston, J.: Model checking single agent behaviours by fluid approximation. *Inf. Comput.* **242**(C), 183–226 (2015)
11. Bortolussi, L., Lanciani, R.: Model checking Markov population models by central limit approximation. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 123–138. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40196-1\\_9](https://doi.org/10.1007/978-3-642-40196-1_9)
12. Bortolussi, L., Lanciani, R.: Stochastic approximation of global reachability probabilities of Markov population models. In: Horváth, A., Wolter, K. (eds.) EPEW 2014. LNCS, vol. 8721, pp. 224–239. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10885-8\\_16](https://doi.org/10.1007/978-3-319-10885-8_16)
13. Bortolussi, L., Lanciani, R., Nenzi, L.: Model checking Markov population models by stochastic approximations. *CoRR*, abs/1711.03826 (2017)
14. Bortolussi, L., Milios, D., Sanguinetti, G.: Smoothed model checking for uncertain continuous time Markov chains. *Inform. Comput.* **247**, 235–253 (2016)
15. Bradley, J.T., Hayden, R.A., Clark, A.: Performance specification and evaluation with unified stochastic probes and fluid analysis. *IEEE Trans. Softw. Eng.* **39**, 97–118 (2013)
16. Cseke, B., Opper, M., Sanguinetti, G.: Approximate inference in latent Gaussian-Markov models from continuous time observations. In: Proceedings of NIPS, pp. 971–979. Curran Associates Inc. (2013)
17. Cseke, B., Schnoerr, D., Opper, M., Sanguinetti, G.: Expectation propagation for continuous-time stochastic processes. *J. Phys. A-Math. Theor.* **49**(49), 494002 (2016)
18. Durrett, R.: *Essentials of Stochastic Processes*. Springer, New York (2012)
19. Feng, C., Hillston, J., Galpin, V.: Automatic moment-closure approximation of spatially distributed collective adaptive systems. *ACM Trans. Model. Comput. Simul.* **26**(4), 26:1–26:22 (2016)
20. Gardiner, C.W.: *Handbook of Stochastic Methods*, vol. 3. Springer, Berlin (1985)
21. Goodman, L.A.: Population growth of the sexes. *Biometrics* **9**(2), 212–225 (1953)
22. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Form. Asp. Comput.* **6**(5), 512–535 (1994)
23. Haseltine, E.L., Rawlings, J.B.: Approximate simulation of coupled fast and slow reactions for stochastic chemical kinetics. *J. Chem. Phys.* **117**(15), 6959 (2002)
24. Hayden, R.A., Stefanek, A., Bradley, J.T.: Fluid computation of passage-time distributions in large Markov models. In: Proceedings of QAPL, vol. 413, pp. 106–141 (2012)
25. Hespanha, J.P.: *StochDynTools* – a MATLAB toolbox to compute moment dynamics for stochastic networks of bio-chemical reactions
26. Katoen, J.-P., Khattri, M., Zapreevt, I.S.: A Markov reward model checker. In: Proceedings of QEST, pp. 243–244. IEEE Computer Society (2005)
27. Kierzek, A.M.: STOCKS: STOChastic Kinetic Simulations of biochemical systems with Gillespie algorithm. *Bioinformatics* **18**(3), 470–481 (2002)



28. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) *CAV 2011*. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_47](https://doi.org/10.1007/978-3-642-22110-1_47)
29. Maybeck, P.S.: *Stochastic Models, Estimation, and Control*. Academic Press, New York (1982)
30. Milios, D., Sanguinetti, G., Schnoerr, D.: Probabilistic model checking for continuous-time Markov chains via sequential Bayesian inference. *CoRR ArXiv*, abs/1711.01863v2 (2018)
31. Minka, T.P.: *A family of algorithms for approximate Bayesian inference*. Ph.D. thesis, MIT (2001)
32. Norris, J.R.: *Markov Chains*. Cambridge University Press, Cambridge (1997)
33. Schnoerr, D., Cseke, B., Grima, R., Sanguinetti, G.: Efficient low-order approximation of first-passage time distributions. *Phys. Rev. Lett.* **119**, 210601 (2017)
34. Schnoerr, D., Sanguinetti, G., Grima, R.: Validity conditions for moment closure approximations in stochastic chemical kinetics. *J. Chem. Phys.* **141**(8), 08B616.1 (2014)
35. Schnoerr, D., Sanguinetti, G., Grima, R.: Comparison of different moment-closure approximations for stochastic chemical kinetics. *J. Chem. Phys.* **143**(18), 11B610.1 (2015)
36. Schnoerr, D., Sanguinetti, G., Grima, R.: Approximation and inference methods for stochastic biochemical kinetics - a tutorial review. *J. Phys. A* **50**(9), 093001 (2017)
37. Younes, H.L., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.* **204**(9), 1368–1409 (2006)
38. Zuliani, P., Platzer, A., Clarke, E.M.: Bayesian statistical model checking with application to Simulink/Stateflow verification. In: *Proceedings of HSCC*, pp. 243–252. ACM (2010)



# LIFT: Learning Fault Trees from Observational Data

Meike Nauta<sup>(✉)</sup>, Doina Bucur, and Mariëlle Stoeltinga

University of Twente, Enschede, The Netherlands  
{m.nauta,d.bucur,m.i.a.stoeltinga}@utwente.nl

**Abstract.** Industries with safety-critical systems increasingly collect data on events occurring at the level of system components, thus capturing instances of system failure or malfunction. With data availability, it becomes possible to automatically learn a model describing the failure modes of the system, i.e., how the states of individual components combine to cause a system failure. We present LIFT, a *machine learning method* for *static fault trees* directly out of *observational datasets*. The fault trees model probabilistic causal chains of events ending in a global system failure. Our method makes use of the Mantel-Haenszel statistical test to narrow down possible causal relationships between events. We evaluate LIFT with synthetic case studies, show how its performance varies with the quality of the data, and discuss practical variants of LIFT.

## 1 Introduction

Fault tree (FT) analysis [1] is a widely applied method to analyse the safety of high-tech systems, such as self-driving cars, drones and robots. FTs model how system failures occur as a result of component failures: the leaves of the tree model different failure modes, while the fault tree gates model how failure modes propagate through the system and lead to system failures. A wide number of metrics, such as the system reliability and availability, can then be computed to evaluate whether a system meets its dependability and safety requirements.

A key bottleneck is the construction of the FT. This requires domain knowledge, and the number of potential *failure causes* and contributing factors can be overwhelming: age, system loads, usage patterns and environmental conditions can all influence the failure mechanisms. It is thus appealing to learn FTs automatically from data, to assist reliability engineers in tackling the complexity of today's systems. This paper is a first step in this direction: we *learn* static FTs from *observational records*.

**The Fault-Tree Formalism.** The nodes in an FT are either events or logical gates. Figure 1 shows an example FT and the graphical notation. A part of the system is modelled by an *intermediate event*; a special intermediate event is the root node of the tree, called the *top event* or *outcome*, which models the global system failure. A set of *basic events*, distinct from the intermediate events, marks the most elementary faults in system components, may be annotated

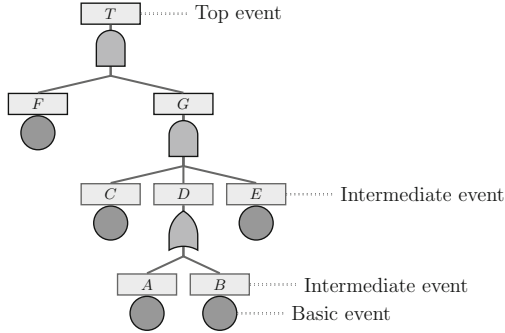


Fig. 1. Example fault tree with annotations

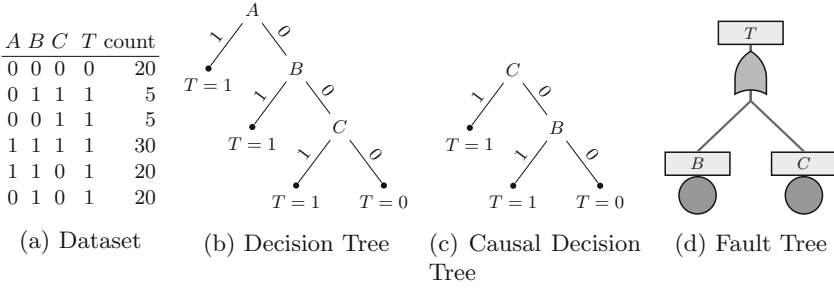
with a probability of occurrence, and form the leaves of the FT. Intermediate events form the inputs and the output of any gate, and are the output of any basic event. The basic gates, AND and OR, denoted by the standard logic-gate symbols, model their standard logic meaning, in terms of causal relationships between the events in the input and the event in the output of any gate.

**Summary of Contribution.** We learn static FTs with Boolean event variables (where an event variable has value True or 1 if that fault occurs in the system),  $n$ -ary AND/OR gates, and annotated with event failure probabilities. The input to the algorithm consists of raw, untimed observational data over the system under study, i.e., a dataset where each row is a single *observation* over the entire system, and each column *variable* records the value of a system *event*. All intermediate events to be included in the FT must be present in the dataset, but not all of those events in the dataset may be needed in the FT. We do not know what the basic events will be, nor which gates form the FT, nor which intermediate events are attached to the gates. We know the top event: the system failure of interest. Our main result is an algorithm that learns a statistically significant FT; we allow for a user-specified amount of noise, assumed uniformly distributed in the data. We evaluate the algorithm on synthetic data: given a “ground truth” FT, we synthesise a random dataset, apply the learning algorithm, and then compare the machine-learnt FT to the ground truth.

An example dataset is shown in Fig. 2a, in compact form: each row is a *count* (e.g., 20) of identical *records*, where each record is an untimed list of Boolean observations of events (denoted  $A, B, C$  and  $T$ , with  $T$  the global system failure, or outcome). The order of the records in a dataset is not significant.

A tree formalism commonly machine-learnt from such observational data is the Binary Decision Tree (BDT), a practical tool for the description, classification and generalisation of data [2]. The BDT learning algorithm appears to be a natural starting point for the design of an FT learning algorithm; however, we argue below why the BDT learning logic is unsatisfactory.

**Detecting Causality from Data.** The construction of a BDT is a greedy algorithm: a series of local optimum decisions determines subsequent branching



**Fig. 2.** An example showing that a BDT does not encode causal relationships.

nodes. Each decision uses a variable test condition (e.g., a classification error) to find the best split in the data records [3]. For example, when creating a BDT for the dataset in Fig. 2a to classify variable  $T$ , a naive approach is to first split the dataset on variable  $A$ , and obtain the BDT in Fig. 2b. However, decision trees model *correlations* (which are symmetric) and not the *causal* relationships (which are asymmetric) required for an FT. As a correlation between variables does not imply a causation, the knowledge represented in a decision tree does not provide root causes for faults, and thus cannot support decision making.

To overcome this problem, the *Causal Decision Tree* (CDT) [4] was recently introduced. A CDT differs from a BDT in that each of its non-leaf nodes has a causal interpretation with respect to the outcome  $T$ . CDTs find causal relationships automatically by using the Mantel-Haenszel statistical test [5]. A causal relationship between a variable and the outcome exists if the causal effect is statistically significant (i.e., is above random chance). A CDT for the dataset in Fig. 2a is in Fig. 2c; it shows that  $C$  has a causal relationship with  $T$  and that  $B$  has a causal relationship with  $T$  under the “context” (i.e., fixed variable assignment)  $C = 0$ .  $A$  is not included in this CDT. The path  $(A = 1) \rightarrow (T = 1)$  in the BDT with probability  $P(T = 1|A = 1) = 1$  correctly classifies half of the records in the dataset. However, the path does not code a causal relationship between  $A$  and  $T$  since, for example, given  $C = 1$ ,  $P(T = 1|A = 1) - P(T = 1|A = 0) = 0$ . When fixing the value of  $C$ , a change in  $A$  does not result in a change in  $T$ . In fact,  $C$  causes  $T$ , and  $B$  causes  $T$  under the context  $C = 0$ .

CDTs on average achieve similar classification accuracy as decision trees, even though this is not a CDT objective; also, the size of CDTs is on average half that of decision trees [4], simplifying their analysis. Some aspects of CDT learning are useful in the automatic construction of an FT. However, while a CDT can only model the causal relationship between a variable and the outcome, the strength of an FT is the additional modelling of (a) multiple independent variables that may cause a failure, and (b) if-then Boolean logic. As shown in Fig. 2d, the CDT of Fig. 2c can be redrawn as an FT with a single OR gate.

In the following, Sect. 2 gives the related work on the automated synthesis of fault trees. Section 3 formally introduces FTs. Section 4 presents the LIFT algorithm and examples. Section 5 evaluates LIFT on datasets with noise or

superfluous variables. Section 6 discusses possible LIFT variants. The conclusions, including future work, are presented in Sect. 7.

## 2 Related Work

System dependability evaluation via fault tree analysis is largely a manual process, performed over informal fault-tree models which will not accurately describe an evolving system [6]. Due to this, the *automatic synthesis of fault trees* has been of recent interest. However, we stress the fact that most of the existing contributions generate the necessary dependability information from existing, formal system models, and are thus Model-Based Dependability Analysis (MBDA) techniques [6–8]. In contrast, there is little research aiming to synthesise causal dependability information for *black-box systems*, for which formal models do not exist, or for which the quantity and quality of the available sensed data surpasses the quality and completeness of existing system models.

**Learning Fault Trees from Data.** Observational data was used for machine-learning fault trees in the Induction of Fault Trees (IFT) algorithm [9], based on decision-tree learning. As in our method, all that is needed are observations of measurable quantities taking certain values. However, IFT completely disregards the matter of causality between events, and essentially learns a syntactically correct FT which encodes exactly the same information as a decision tree – so the FT is essentially a classifier, rather than a means of modelling causal effect.

**Generating Fault Trees from Formal System Models.** A diverse body of techniques is available for this; we refer to recent reviews on MBDA for a complete picture [6–8] and give here a brief overview of the most relevant generation methods. While these approaches cannot directly synthesise FTs from observational data (as in our work), other techniques able to learn the required system models from observational data could (indirectly) bridge this gap.

In the Hierarchically Performed Hazard Origin & Propagation Studies (HiP-HOPS) framework [10], any system model formalising the transactions among the system components, annotated with failure information for components (as Boolean expressions), may be used to synthesise an FT. Using these annotations, the synthesis is straightforward: it proceeds top-down from the top event and creates local FTs based on the component failure annotations; these are then merged into a global FT showing all combinations leading to system failure. If formal models in the AltaRica high-level system description language are available, they include explicit transitions modelling causal relations between state variables and events, which can similarly be used to synthesise classic FTs [11]. The Formal Safety Analysis Platform (FSAP/NuSMV-SA) generates, from NuSMV system models, FTs which show only the relation between top events and basic events, and not how faults propagate among the system components [12]. The Architecture Analysis and Design Language (AADL) includes an Error Model for the specification of fault information, and a number of techniques exist to translate an AADL model into static or dynamic FTs (recently, in [13]).

AADL models have also been translated into models compatible with the HiP-HOPS and AltaRica frameworks, enabling cross-framework FT synthesis [6].

A process of FT generation with explicit reasoning about causality is described in [14]; however, this approach still requires a formal system model to exist. Given such a probabilistic system model, a set of probabilistic counterexamples (i.e., system execution paths of temporally ordered, interleaved events leading to a system fault) is obtained from the process of model-checking. As the system is concurrent, the counterexamples potentially, but not necessarily, model causality. Logical combinations of events are determined as causes of other events using a set of test conditions; the time complexity is cubic in the size of the set of counterexamples.

**Other Approaches.** Causal Bayesian Networks (CBNs) [15] can also be learnt from observational data, as well as Boolean formulas (BFs) [16]; both models may be translated into FTs, and both learning problems are NP-hard or require exponential time [16, 17]. As our algorithm will also be shown to have a worst-case exponential complexity, both CBNs and BFs remain feasible alternatives to FT learning.

### 3 Background: Fault Trees

We define the basic components of an FT formally in Definitions 1–4.

**Definition 1.** A *gate*  $G$  is a tuple  $\langle t, \mathbf{I}, O \rangle$ , where:

- $t$  is the type of  $G$ , with  $t \in \{And, Or\}$ .
- $\mathbf{I}$  is a set of  $n \geq 2$  intermediate events  $\{i_1, \dots, i_n\}$  that are inputs to  $G$ .
- $O$  is the intermediate event that is output for  $G$ .

We denote by  $I(G)$  the set of intermediate events in the input of  $G$ , and by  $O(G)$  the intermediate event in the output of  $G$ .

**Definition 2.** An **AND gate** is a gate  $\langle And, \mathbf{I}, O \rangle$  where output  $O$  occurs (i.e.  $O$  is True) if and only if every  $i \in \mathbf{I}$  occurs.

**Definition 3.** An **OR gate** is a gate  $\langle Or, \mathbf{I}, O \rangle$  where output  $O$  occurs (i.e.  $O$  is True) if and only if at least one  $i \in \mathbf{I}$  occurs.

**Definition 4.** A **basic event**  $B$  is an event with no input and one intermediate event as output. We denote by  $O(B)$  the intermediate event in the output of  $B$ .

Intuitively, a basic event  $B$  models an elementary system fault in the real world; its output  $O(B)$  is True when this elementary system fault occurs. Then, all system components modelled by the events in the input of an AND gate must fail in order for the system modelled by the event in the output to fail.

We then formalise the fault tree in Definition 5.

**Definition 5.** A *fault tree*  $\mathbf{F}$  is a tuple  $\langle \mathbf{BE}, \mathbf{IE}, T, \mathbf{G} \rangle$ , where:

- $\mathbf{BE}$  is the set of basic events;  $\forall B \in \mathbf{BE}, O(B) \in \mathbf{IE}$ . A basic event may be annotated with a probability of occurrence  $p$ .
- $\mathbf{IE}$  is the set of intermediate events, where  $\mathbf{IE} \cap \mathbf{BE} = \emptyset$ .
- $T$  is the top event,  $T \in \mathbf{IE}$ .
- $\mathbf{G}$  is the set of gates;  $\forall G \in \mathbf{G}, I(G) \subset \mathbf{IE}, O(G) \in \mathbf{IE}$ .
- The graph formed by  $\mathbf{G}$  should be connected and acyclic, with the top event  $T$  as unique root.

Given fault tree  $\mathbf{F}$ , we denote by  $IE(\mathbf{F})$  the set of intermediate events in  $\mathbf{F}$ .

The basic LIFT algorithm (Sect. 4) will learn trees rather than directed acyclic graphs (DAGs), i.e. an intermediate event can be the input of only one gate. Section 6 will then discuss a DAG variant of the LIFT algorithm.

**Comparison FT-CDT.** Unlike FTs, CDTs can be learnt from data, and also encode causal relationships between variables; an example CDT was given in Fig. 2c. However, there are major *syntactic* differences between the two formalisms. An FT can be n-ary, while a CDT can only be binary: every branching decision is based on a Boolean variable. Also, an FT is more concise: it models only the positive (failure) outcome, while the CDT must model both outcomes of any variable. Finally, the position of the outcome differs: while in FTs the top event models the system outcome, in a CDT this is modelled by leaf nodes.

## 4 Machine Learning Fault Trees

The dataset from which an FT can be learnt contains untimed, Boolean observations of system events; an FT *event* corresponds to a column *variable* in the dataset. A record and a dataset are formally defined in Definitions 6–7.

**Definition 6.** A *record*  $R$  over the set of variables  $\mathbf{V}$  is a list of length  $|\mathbf{V}|$  containing tuples  $[(V_i, v_i)]$ ,  $1 \leq i \leq |\mathbf{V}|$ , where:

- $V_i$  is a variable name,  $V_i \in \mathbf{V}$ .
- $v_i$  is a Boolean value of  $V_i$ .

**Definition 7.** A *dataset*  $\mathbf{D}$  is a set of  $r$  records, all over the same set of variables  $\mathbf{V}$ . Each variable name in  $\mathbf{V}$  forms a column in  $\mathbf{D}$  and each record forms a row. When  $k$  identical records are present in  $\mathbf{D}$ , a single such record is shown, with a new count column for the value  $k$ .

A synthetic dataset (of 185 records in total, but only 11 unique records) is shown in Table 1. We assume the *sufficiency* of any dataset (i.e., all shared causes are measured [18]) and also its *faithfulness* (i.e., the data accurately represents the real-world dependencies [18]). However, because of either sensor glitches or human error, there may be some noise in the dataset (i.e., flipped bits).

**Table 1.** Example dataset

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>	<i>T</i>	Count
0	0	0	0	0	0	0	0	30
1	0	1	1	0	1	0	0	20
0	1	0	1	1	0	0	0	20
0	1	0	1	1	1	0	0	20
1	0	0	1	0	0	0	0	15
0	0	1	0	0	1	0	0	15
0	0	0	0	1	0	0	0	15
0	0	1	0	1	1	0	0	15
1	1	1	1	1	0	1	0	20
0	1	1	1	1	1	1	1	10
1	0	1	1	1	1	1	1	5

From a dataset, causal relationships between (groups of) variables can be discovered to form an FT. For this, one can use the standard Mantel-Haenszel Partial Association test (PAMH) [5], a test used for the analysis of data that is *stratified*. When stratifying the dataset, the effect of other variables on the outcome variable *T* is eliminated, and hence the difference reflects the causal effect of one variable (say, *E*) on the outcome *T*. By this test, a causal relationship between two given variables is statistically significant if and only if the PAMH-score  $\geq \tilde{\chi}_{\alpha,1}^2$ , where  $\tilde{\chi}_{\alpha,1}^2$  is the standard critical value of the chi-square distribution with 1 degree of freedom [19]. A significance level of  $\alpha = 0.05$  or  $\alpha = 0.01$  is often used in practice.

A stratum is formally defined in Definition 8 (and is a classic concept, as per [19]). A concrete example is given later in this section, in Example 1.

**Definition 8.** *Given a dataset **D** over the set of variables **V**, a **stratum**  $S_{E,T}$  (where *E* and *T* are variables from **V**) is a contingency table which shows the distribution of the values of variables *E* and *T* in the dataset, as counts, in the following format:*

	<i>T</i> = 1	<i>T</i> = 0	Total
<i>E</i> = 1	$n_{11}$	$n_{12}$	$n_{1.}$
<i>E</i> = 0	$n_{21}$	$n_{22}$	$n_{2.}$
Total	$n_{.1}$	$n_{.2}$	$n_{..}$

where *n* denotes the number of records that satisfy a given valuation of *E* and *T*.



The PAMH-score can be calculated over multiple strata (as done in CDTs [4] and first formalised in [19]); here we have a single stratum, as follows:

$$\text{PAMH}(E, T) = \left( \left| \frac{n_{11}n_{22} - n_{21}n_{12}}{n_{..}} \right| - \frac{1}{2} \right)^2 \bigg/ \frac{n_{1.}n_{2.}n_{.1}n_{.2}}{n_{..}^2(n_{..} - 1)}$$

Using these concepts of strata and the PAMH-score, the LIFT algorithm<sup>1</sup>, shown in Algorithm 1, synthesises an FT from a dataset  $\mathbf{D}$ . A variable in  $\mathbf{D}$  corresponds to an intermediate event. All intermediate events to be included in the FT must be present in the dataset, but not all of those events in the dataset may be needed in the FT. We do not know what the basic events will be, nor which gates form the FT, nor which intermediate events are attached to the gates.

*Checking a Proposed Gate.* To create a fault tree  $\mathbf{F}$ , the LIFT algorithm iteratively adds a level to  $\mathbf{F}$ , starting with just the top event (line 28–31 in Algorithm 1). Each time `CreateLevel` is called, the depth of the FT increases with one level. For each intermediate event  $E$  at the lowest level of  $\mathbf{F}$ , sets (of size  $\geq 2$ ) containing intermediate events not yet in  $\mathbf{F}$  are proposed as input of a new (AND or OR) gate whose output is  $E$ . This is done by checking the gate for correctness according to the properties of Definitions 2–3. If both gates are correct, any design choice can be made, as discussed later in Sect. 6.

*Example 1.* Using the data set shown in Table 2a, a significance level  $\alpha$ , the outcome variable  $T$  and the set of variables  $\mathbf{I} = \{A, B\}$ , one can check if gate  $G$  of the form  $\langle \text{And}, \mathbf{I}, T \rangle$  meets the property specified in Definition 2 and is statistically significant using function `CheckANDgate`.

A temporary new variable  $\mathbf{v}$  is added to the dataset (Table 2b).  $\mathbf{v}$  encodes an AND relation between  $A$  and  $B$ ;  $\mathbf{v}$  occurs (is 1) only when both  $A$  and  $B$  occur. This variable  $\mathbf{v}$  can then be compared with top event  $T$  to measure if there is a causal relationship between  $T$  and  $A \text{ AND } B$ . The stratum  $\mathbf{S}_{\mathbf{v}, T}$  is computed by counting the corresponding records in Table 2b, as shown in Table 2c.

The user can specify the ratio of noise allowed by LIFT per stratum. (If the user can assume that the flipped bits are uniformly distributed in the dataset, the expected per-stratum noise ratio is equal to the global noise ratio). For simplicity, we set this noise “allowance” equal to the significance level  $\alpha$ ; the algorithm is easily modified for any other level. In the dataset shown in Table 2a, one can see that one record may be noise. In this example, we will set  $\alpha = 0.05$ , so we allow 5% noise in a stratum. It then follows that the proposed AND gate  $G$  of the form  $\langle \text{And}, \{A, B\}, T \rangle$  meets the property of Definition 2 because in stratum  $\mathbf{S}_{\mathbf{v}, T}$  we have  $n_{12} = 0$ ,  $n_{21} = 1$ , meaning one record where the values of  $\mathbf{v}$  and  $T$  differ. We do allow a ratio  $\alpha$  out of  $n_{..} = 91$  to differ, but  $1 < 0.05 \cdot 91$  holds. However, if we would have selected a significance level  $\alpha = 0.01$ , since  $1 < 0.01 \cdot 91$  does not hold, the FT couldn’t include this gate.

<sup>1</sup> Code can be found at <https://github.com/M-Nauta/LIFT>.

---

**Algorithm 1.** LIFT: Learning a Fault Tree from a dataset
 

---

**Input:**  $\mathbf{D}$ , a data set containing  $r$  records over  $\mathbf{V}$ ;  
 $T$ , the intended top event with  $T \in \mathbf{V}$ ;  
 $\alpha$ , the significance level for the Mantel-Haenszel test

**Result:** Fault Tree  $\mathbf{F}$

```

1 Function CheckANDGate( $E, \mathbf{I}$ ):
2    $result = \text{False}$ ,  $pamh = 0.0$ 
3    $\mathbf{v} = [v_1, \dots, v_r]$  in which  $v_j$  is 1 if every  $i \in \mathbf{I}$  in record  $j$  of  $\mathbf{D}$  is 1
4   if  $n_{12}$  of  $\mathbf{S}_{\mathbf{v},E} < \alpha n_{..}$  &&  $n_{21}$  of  $\mathbf{S}_{\mathbf{v},E} < \alpha n_{..}$  then
5      $pamh = \text{PAMH}(\mathbf{v}, T)$ 
6     if  $pamh \geq \tilde{\chi}_{\alpha,1}^2$  then
7        $result = \text{True}$ 
8   return  $result$ ,  $pamh$ 

9 Function CheckORGate( $E, \mathbf{I}$ ):
10 | Similar to lines 2-8

11 Function CreateLevel( $\mathbf{F}$ , Leaves):
12 | for  $l \in \text{Leaves}$  do
13 |    $k = 2$ ,  $gate = \text{False}$ 
14 |   while not  $gate$  and  $k \leq |\mathbf{V} \setminus IE(\mathbf{F})|$  do
15 |     for  $\mathbf{a}$  in generator of combinations of size  $k$  from  $\mathbf{V} \setminus IE(\mathbf{F})$  do
16 |       compute  $isGate$ ,  $pamh = \text{CheckANDGate}(l, \mathbf{a})$ 
17 |       compute  $isGate$ ,  $pamh = \text{CheckORGate}(l, \mathbf{a})$ 
18 |       if at least one  $\mathbf{a}$  exists where  $isGate$  was True then
19 |         select that  $\mathbf{a}$  and gate type  $t$  where  $pamh$  was maximum
20 |         add gate  $(t, \mathbf{a}, l)$  to  $\mathbf{F}$ 
21 |          $gate = \text{True}$ 
22 |       else
23 |          $k++$ 
24 |   if not  $gate$  then
25 |      $p = \text{ratio of records in } \mathbf{D} \text{ where } l = 1$ 
26 |     create basic event  $B$  as input for  $l$  in  $\mathbf{F}$ , and annotate  $B$  with  $p$ 
27 |   return  $\mathbf{F}$ 

28 let  $\mathbf{F} = \text{Fault Tree } \langle \emptyset, \{T\}, T, \emptyset \rangle$ 
29 while at least one new event is added to  $\mathbf{F}$  do
30 | let Leaves = set of all intermediate events at the lowest level of  $\mathbf{F}$ 
31 |  $\mathbf{F} = \text{CreateLevel}(\mathbf{F}, \text{Leaves})$ 

```

---

If the proposed gate has less noise than allowed (which is in this case true for  $\alpha = 0.05$ ), we can determine if the causal relation between  $T$  and  $\mathbf{v}$  is significant, by calculating the PAMH-score:

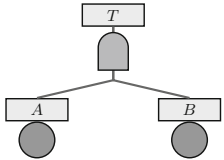
$$\text{PAMH}(\mathbf{v}, T) = \left( \frac{15 \cdot 75 - 1 \cdot 0}{91} - \frac{1}{2} \right)^2 \bigg/ \frac{15 \cdot 76 \cdot 16 \cdot 75}{91^2(91 - 1)} = 76.66 .$$

**Table 2.** Dataset for Example 1, over variables  $A, B$  and outcome variable  $T$ ; stratum and fault tree learnt.

$A$	$B$	$T$	count
0	0	0	30
1	0	0	25
0	1	0	20
0	1	1	1
1	1	1	15

$A$	$B$	$\mathbf{v}$	$T$	count
0	0	0	0	30
1	0	0	0	25
0	1	0	0	20
0	1	0	1	1
1	1	1	1	15

$T=1$ $T=0$ Total			
$\mathbf{v} = 1$	15	0	15
$\mathbf{v} = 0$	1	75	76
Total	16	75	91



(a) Dataset

(b) Dataset incl.  
var.  $\mathbf{v}$  (AND gate)

(c) Stratum  $\mathbf{S}_{\mathbf{v},T}$

(d) FT learnt

For  $\alpha = 0.05$ , the critical value  $\tilde{\chi}_{\alpha,1}^2 = 3.84$ . Since the PAMH-score is higher than  $\tilde{\chi}_{\alpha,1}^2$ ,  $\mathbf{v}$  and  $T$  can be concluded to have a significant causal relationship.

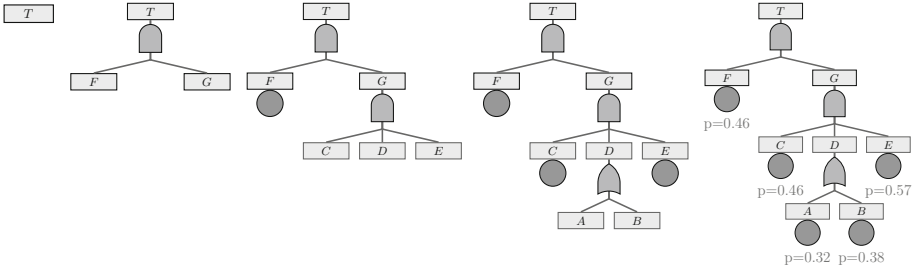
Similarly, a proposed OR gate is checked. By creating a temporary new variable  $\mathbf{v}$  for the OR gate, one can create a stratum to calculate the noise and the PAMH-score in a similar way. This OR gate will have too much noise and is therefore not correct. So,  $\langle \text{And}, \mathbf{I}, T \rangle$  is added to  $\mathbf{F}$ . Table 2d shows the final FT learnt from the original dataset in Table 2 for  $\alpha = 0.05$ .

An FT may have a path containing two subsequent gates of the same type; in this case, the FT solution is not unique, and one may optimise for either *minimal gate sizes*, or *minimal tree depth*. We choose here the former, i.e., select the smallest input sets for all gates. LIFT is easily modified for another aim. Example 2 below clarifies this situation.

*Example 2.* Take the dataset in Table 1 at the beginning of this section. The LIFT algorithm starts with an FT containing only the top event  $T$  (line 28 in Algorithm 1). For this top event, the algorithm generates all combinations (sets) of intermediate events, in order of increasing size (line 15). For each set  $\mathbf{a}$  containing intermediate events, LIFT tests whether a gate  $\langle \text{Or}/\text{And}, \mathbf{a}, T \rangle$  (either AND or OR) meets the property in Definitions 2-3 and does not exceed the noise allowance (line 4). If true, LIFT checks if the PAMH score is higher than the threshold for the Mantel-Haenszel test.

For this dataset, there is no correct OR gate  $\langle \text{Or}, \mathbf{a}, T \rangle$ . However, there are 9 sets of intermediate events that can act as input for a correct and significant AND gate  $\langle \text{And}, \mathbf{a}, T \rangle$ :  $\mathbf{a} = \{F, G\}$ ,  $\mathbf{a} = \{E, F, G\}$ ,  $\mathbf{a} = \{D, F, G\}$ ,  $\mathbf{a} = \{C, F, G\}$ ,  $\mathbf{a} = \{D, E, F, G\}$ ,  $\mathbf{a} = \{C, E, F, G\}$ ,  $\mathbf{a} = \{C, D, F, G\}$ ,  $\mathbf{a} = \{C, D, E, F\}$  and  $\mathbf{a} = \{C, D, E, F, G\}$ . In other words, there are multiple structural solutions for the FT, when the FT has a path with two subsequent gates of the same type. In such cases, LIFT learns the solution with minimum-sized gates. The input sets are generated in increasing size, and LIFT will stop proposing input sets when the minimum correct input set is found. In this example, the smallest set has size two, namely  $\{F, G\}$ . Therefore, gate  $\langle \text{And}, \{F, G\}, T \rangle$  is added to  $\mathbf{F}$  (as shown in Fig. 3). In case of multiple correct sets of the same size (which can arise when

there are more variables in the dataset than needed in the FT) the set with the highest PAMH-score is selected (line 19 in Algorithm 1). The algorithm can be easily modified for another design decision, as argued later in Sect. 6.



**Fig. 3.** Applying LIFT in Example 2 on the dataset shown in Table 1.

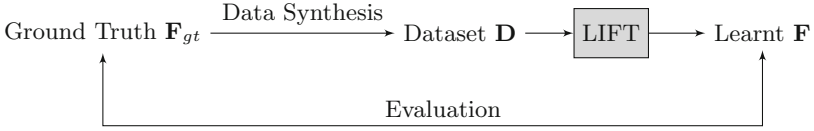
In the next iteration, for both  $F$  and  $G$  again sets of intermediate events are tried. The search space is now  $2^5 - 5 - 1 = 26$ , because  $|\mathbf{V} \setminus IE(\mathbf{F})| = 5$ . There is no correct gate  $\langle Or/And, \mathbf{a}, F \rangle$  for intermediate event  $F$ . Therefore, a basic event is added as input for  $F$  (line 26). For intermediate event  $G$ , one correct and significant AND gate  $\langle And, \{C, D, E\}, G \rangle$  is found and added to  $\mathbf{F}$ . Similar iterations are done for  $C$ ,  $D$  and  $E$  followed by  $A$  and  $B$ , as shown in Fig. 3.

When the dataset contains information on system states which are always measured in a fixed time horizon (i.e. *discrete* time), one can easily derive stochastic measures such as failure probabilities using standard probability laws. The statistical probability that an event  $E \in \mathbf{D}$  occurs is simply  $P(E = 1) = \frac{\# \text{ records where } E=1}{\text{total } \# \text{ records}}$ ; all basic events are annotated with these probabilities.

## 5 Evaluation

The algorithm is evaluated following the approach shown in Fig. 4. A number of fault trees  $\mathbf{F}_{gt}$  are generated as ground truth; from each of these FTs, a dataset is synthesised randomly, including adding noise and superfluous variables (both of these processes of synthesis are described below). LIFT takes this dataset and a given significance value  $\alpha$  as input, and learns another FT  $\mathbf{F}$ , which can then be compared to the ground truth. We say that a learnt FT is “correct” if it is *structurally equivalent* to the ground-truth FT, i.e., syntactically (and not only semantically) equivalent, where only the order of the inputs to any gate may differ. We require that the learnt FT recovers the *exact gates* as in the ground truth, since these gates may model concrete system components, for which the correct causes of failure should be learnt. Our evaluation is thus stronger than an isomorphism check for the FTs.

Furthermore, we assess how noise and superfluous variables in the dataset influence the ratio of correctly learnt FTs.



**Fig. 4.** Evaluation approach: Randomly generate a dataset from an FT, apply LIFT to that dataset and compare the learnt FT with the ground truth.

*Generating all FTs of a Certain Size.* As ground truth, we generate *all possible* FTs over a fixed number (here, 8) of intermediate events, with no probabilities annotated on basic events. We only generate trees and leave DAGs (with shared variables) as future work. To mimic a manually constructed FT where readability is important, we set a minimum of 2 intermediate events and a maximum of 5 intermediate events as input to a gate, and thus obtain 76 different FTs.

*Generating a Synthetic Dataset from an FT.* Based on a generated FT, we mimic a real-life situation by randomly synthesising 1000-record datasets where basic events happen with a certain probability (and are not rare events). The generation process starts with valuating all basic events to either 0 or 1, with a randomly chosen probability between 20% and 50% that each basic event is 1. These values are propagated through the gates in the FT up to the top event; dependent on the type of each gate, the gate’s output event is assigned 0 or 1. Each iteration of this procedure results in one data record.

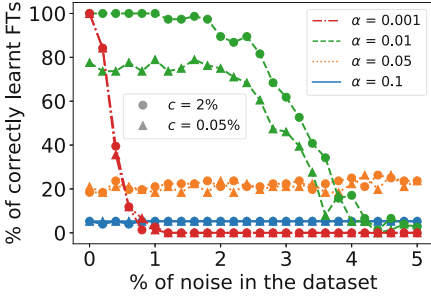
To assure that gates are correctly recognised and that every gate is at least once true, every combination of inputs for a gate occurs in at least  $c\%$  of the rows in the dataset (i.e. in the case of 1000 rows and  $c = 2$ , every combination occurs at least 20 times). We created datasets for both  $c = 0.5\%$  and  $c = 2\%$ . We leave the task of discriminating between rare events and noise for future work.

*Adding Noise to the Dataset.* In a real-life situation, having perfectly clean data is rare because of wrong measurements, sensor glitches or manual errors for example. To mimic noise, a number up to 5% of the rows in the dataset are added, each with 1–2 wrong (flipped) values.

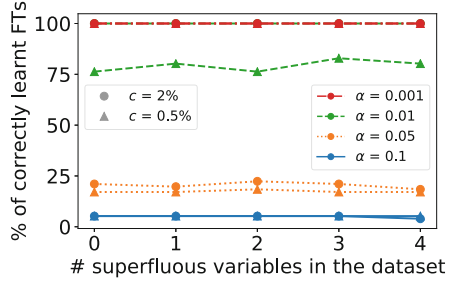
*Adding Superfluous Variables to the Dataset.* Our algorithm should also create a correct fault tree when there are variables in the dataset which have *no causal effect* and should not be included in the learnt FT. We thus experiment with adding up to 4 non-causal system variables. The case of a causal superfluous variable is discussed in Sect. 6.

## 5.1 Results

An analysis is done on the influence of noise or superfluous variables in the dataset on the number of correct fault trees obtained by LIFT.



**Fig. 5.** Percentage of correctly learnt fault trees relative to the percentage of rows with noise in the dataset. All 76 different FTs with 8 intermediate events are generated. The dataset for each FT contains no non-causal variables, and 1000 records plus an extra percentage of noisy records.



**Fig. 6.** Percentage of correctly learnt FTs relative to the number of non-causal variables in the dataset. All 76 different FTs with 8 intermediate events are generated. The dataset for each FT contains 1000 records, no noisy records, but up to 4 non-causal variables.

As we generated all ground-truth FTs with exactly 8 intermediate events, and 2–5 inputs for each gate (76 FTs in total), the basic datasets contain 8 columns (one for each intermediate event), and 1000 rows. Noisy rows are then added to the dataset, depending on the level of noise desired.

Figure 5 shows the percentage of correctly learnt fault trees relative to the percentage of rows with noise in the dataset. All learnt FTs are correct in the absence of noise and with the significance level  $\alpha = 0.001$ . However, this  $\alpha$  is by nature incapable of correctly dealing with noise, since LIFT may not find a significant gate due to the noise. A higher  $\alpha$  is less sensitive for noise, but does result in a lower number of correct FTs. A learnt FT may be incorrect when LIFT finds a significant gate with a smaller number of inputs than what should actually be the case. Therefore, the significance level should be chosen based on the amount of noise in the dataset. Furthermore, one can see that  $c$  naturally influences the number of correctly learnt FTs: the less rare the events are in the dataset, the more likely is LIFT to learn the correct FT.

Figure 6 shows the percentage of correctly learnt FTs relative to the number of non-causal random variables present in the dataset; one can see that these variables have little effect on the accuracy, showing that LIFT indeed finds only causal relationships.

## 5.2 Complexity

**Time Complexity.** LIFT exhaustively checks all input event combinations in order of their size, so in worst case there is one gate with all variables (except the top event) as input. This means that for all input sets, a stratum is created that loops over all  $r$  records and over  $k$  variables that are in that set. The number

of different combinations of size  $k$  is  $\binom{n}{k}$  where  $n = |\mathbf{V}| - 1$ . Therefore, the time complexity of these operations is  $r \cdot \sum_{k=2}^n k \cdot \binom{n}{k}$ . The PAMH-score of each stratum is calculated and compared with the significance level, which has a constant time complexity. This results in a time complexity of  $O(nr2^n)$ .

Learning boolean formulae, closely related to learning static fault trees, from examples obtained by querying an oracle is exponential in the size of the vocabulary in the general case as well as for many restrictions [16]. More precisely, a static fault tree with only AND and OR gates can be seen as a monotone boolean function for which the Vapnik-Chervonenkis (VC) dimension is exponential in  $n$  [20]. So, a general *exact* FT learning algorithm cannot be more efficient than the VC dimension. Reaching better complexity, which could be useful for large datasets, is then only possible when an approximated FT is learnt, instead of an exact solution. Such a variant of Algorithm 1 may apply a *greedy search-and-score* approach rather than our constraint-based approach with exhaustive search, as inspired by structure-learning algorithms for Bayesian networks. However, those algorithms may suffer from getting stuck in a local maximum, resulting in a lower reconstruction accuracy. Furthermore, the highest-scoring network structure is not necessarily the only viable hypothesis [21].

**Space Complexity.** The input for Algorithm 1 consists of dataset  $\mathbf{D}$  with  $r$  records and  $n$  columns, top event  $T$  and significance level  $\alpha$ . Therefore, the input space complexity is  $\Theta(rn)$ . If the generator of combinations is on-the-fly, its auxiliary memory complexity is  $O(n^2)$ .

## 6 Discussion

**Interpretation of Causality.** Currently, all intermediate events that should be in the fault tree have to be included in the dataset. However, obtaining a dataset containing all relevant variables may be impractical. One problem is the presence of hidden variables that influence measured variables but are not in the dataset themselves [18]. The other one is the selection bias: values of unmeasured variables may influence whether a unit is included in the dataset [21]. This can result in a learnt causal relationship between observed variables that does not correspond to the real causal relations. Drawing valid causal inferences using observational data is therefore not just a mechanistic procedure, but always depends on assumptions and justification that require domain knowledge [22]. We are aware of the critical assessment of causal claims based on observational data, but we think the learnt fault tree will still be valuable to give insights which possibly were unknown beforehand and facilitates further causal inference.

**Algorithm Variants.** We made certain design decisions for the basic LIFT algorithm in Algorithm 1. Below, we present some of the many possible variants.

*Multiple Gate Types.* In the case of multiple significant correct gates with the same number of inputs, the LIFT algorithm chooses the one with the highest PAMH-score. However, there may be cases where both an OR gate and an AND gate are correct. For example, in case of the dataset as shown in Table 3, an OR gate will be created when a very high significance level is chosen. However, two of these records may be noise, so with a lower significance level an AND gate will result in a correct gate as well. Selecting the gate type is then a matter of choice: one can argue to choose the OR gate as this matches exactly the dataset, or choose the AND gate since the interpretation of this gate is stricter than the OR gate. One can also argue that the algorithm need not make a decision at all and that it outputs multiple FTs. LIFT is easily modified for any design decision.

**Table 3.** Dataset where both an OR gate and an AND gate may be correct.

A	B	T	Count
0	0	0	1
1	0	1	1
0	1	1	1
1	1	1	10,000

*Multiple Significant FTs.* When there are *causal* superfluous variables in the dataset, there may be cases of multiple correct sets of intermediate events of the same size, that can all serve as input to a statistically significant gate. While the basic LIFT algorithm chooses the input set with the highest PAMH-score, it is easily modified for a different design choice, such as returning all correct FTs.

*The FT as a Directed Acyclic Graph (DAG).* The basic LIFT algorithm learns trees, so the examples and evaluation presented in this paper all learn tree structures. However, in general FTs may share subtrees, meaning that an intermediate event can be the input of multiple gates, and therefore have a directed acyclic structure [1]. The LIFT algorithm can be modified to create DAGs by generating broader combinations  $\mathbf{a}$  of intermediate events, outside the while loop at line 14 of Algorithm 1: instead of a generator of all combinations of size  $\geq 2$  from  $\mathbf{V} \setminus IE(\mathbf{F})$ , one can instead have a generator of all combinations from  $\mathbf{V} \setminus T$ , with an extra check that the created graph  $\mathbf{F}$  remains acyclic.

*More Efficient Exploration of Variable Combinations.* Other features of the dataset (e.g., the graph of dependencies between variables), or even domain knowledge, may be used to reduce the number of combinations of variables to be tried by LIFT as inputs to gates.



## 7 Conclusion

In this paper, we presented an algorithm to automatically learn a statistically significant fault tree from Boolean observational data, inspired by the construction algorithm for Causal Decision Trees. In absence of noise, all learnt FTs were found to be structurally equivalent to the ground truth when the significance level is 0.001. With up to 3% noise in the data, a significance level of 0.01 results in around 65% correct FTs. As a downside, the basic LIFT algorithm does an exhaustive search, and thus has exponential time complexity. It also cannot deal with hidden variables.

In future work, the algorithm can be extended to learn other elements of a fault tree, such as the XOR gate (true if and only if exactly one of its input events is true). Note that elements that need sequence information (such as the Priority-AND gate or the SPARE gate) cannot be implemented, since the required dataset format doesn't contain timing information. Learning fault trees from *timed observational data* is also a direction for future work. For this, learning Bayesian networks, closely related to FTs, may also be a competitive direction to take. Moreover, one may allow continuous data instead of only binary values, similar to the C4.5 algorithm for decision trees [23] that creates a binary expression for continuous values. This expression encodes the conditions under which a measurement results in a failure.

**Acknowledgements.** This research was supported by the Dutch STW project SEQUOIA (grant 15474). The authors would like to thank Joost-Pieter Katoen and Djoerd Hiemstra for valuable feedback.

## References

1. Ruijters, E., Stoelinga, M.: Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. *Comput. Sci. Rev.* **15**, 29–62 (2015)
2. Murthy, S.K.: Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Min. Knowl. Discov.* **2**(4), 345–389 (1998)
3. Tan, P., Steinbach, M., Kumar, V.: *Introduction to Data Mining*. Pearson Education (2006)
4. Li, J., Ma, S., Le, T., Liu, L., Liu, J.: Causal decision trees. *IEEE Trans. Knowl. Data Eng.* **29**(2), 257–271 (2017)
5. Mantel, N., Haenszel, W.: Statistical aspects of the analysis of data from retrospective studies of disease. *J. Nat. Cancer Inst.* **22**(4), 719–748 (1959)
6. Kabir, S.: An overview of fault tree analysis and its application in model based dependability analysis. *Expert Syst. Appl.* **77**, 114–135 (2017)
7. Aizpurua, J.I., Muxika, E.: Model-based design of dependable systems: limitations and evolution of analysis and verification approaches. *Int. J. Adv. Secur.* **6**(1–2), 12–31 (2013)
8. Sharvia, S., Kabir, S., Walker, M., Papadopoulos, Y.: Model-based dependability analysis: state-of-the-art, challenges, and future outlook. In: *Software Quality Assurance*, pp. 251–278. Elsevier (2016)
9. Madden, M.G., Nolan, P.J.: Generation of fault trees from simulated incipient fault case data. *WIT Trans. Inf. Commun. Technol.* **6**, 568–569 (1994)

10. Papadopoulos, Y., McDermid, J.: Safety-directed system monitoring using safety cases. Ph.D. thesis, University of York (2000)
11. Li, S., Li, X.: Study on generation of fault trees from Altarica models. *Procedia Eng.* **80**, 140–152 (2014)
12. Bozzano, M., Villaflorita, A.: The FSAP/NuSMV-SA safety analysis platform. *Int. J. Softw. Tools Technol. Transf.* **9**(1), 5 (2007)
13. Li, Y., Zhu, Y., Ma, C., Xu, M.: A method for constructing fault trees from AADL models. In: Calero, J.M.A., Yang, L.T., Mármol, F.G., García Villalba, L.J., Li, A.X., Wang, Y. (eds.) ATC 2011. LNCS, vol. 6906, pp. 243–258. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-23496-5\\_18](https://doi.org/10.1007/978-3-642-23496-5_18)
14. Leitner-Fischer, F., Leue, S.: Probabilistic fault tree synthesis using causality computation. *Int. J. Crit. Comput.-Based Syst.* **4**(2), 119–143 (2013)
15. Li, J., Shi, J.: Knowledge discovery from observational data for process control using causal Bayesian networks. *IIE Trans.* **39**(6), 681–690 (2007)
16. Jha, S., Raman, V., Pinto, A., Sahai, T., Francis, M.: On learning sparse Boolean formulae for explaining AI decisions. In: Barrett, C., Davies, M., Kahsai, T. (eds.) NFM 2017. LNCS, vol. 10227, pp. 99–114. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-57288-8\\_7](https://doi.org/10.1007/978-3-319-57288-8_7)
17. Chickering, D.M., Heckerman, D., Meek, C.: Large-sample learning of Bayesian networks is NP-hard. *J. Mach. Learn. Res.* **5**, 1287–1330 (2004)
18. Kleinberg, S.: *Why: A Guide to Finding and Using Causes*. O’Reilly (2015)
19. Birch, M.: The detection of partial association, I: the  $2 \times 2$  case. *J. Royal Stat. Soc. Ser. B (Methodological)* **26**, 313–324 (1964)
20. Kearns, M., Li, M., Valiant, L.: Learning Boolean formulas. *J. ACM (JACM)* **41**(6), 1298–1328 (1994)
21. Koller, D., Friedman, N.: *Probabilistic Graphical Models: Principles and Techniques*. MIT Press (2009)
22. Rohrer, J.M.: *Thinking clearly about correlations and causation: graphical causal models for observational data* (2017)
23. Quinlan, J.R.: *C4. 5: Programs for Machine Learning*. Elsevier (2014)



# A Robust Genetic Algorithm for Learning Temporal Specifications from Data

Laura Nenzi<sup>1(✉)</sup>, Simone Silvetti<sup>2,3(✉)</sup>, Ezio Bartocci<sup>1</sup>, and Luca Bortolussi<sup>4</sup>

<sup>1</sup> TU Wien, Vienna, Austria  
laura.nenzi@gmail.com

<sup>2</sup> DIMA, University of Udine, Udine, Italy

<sup>3</sup> Esteco S.p.A., Trieste, Italy  
simone.silvetti@gmail.com

<sup>4</sup> DMG, University of Trieste, Trieste, Italy

**Abstract.** We consider the problem of mining signal temporal logical requirements from a dataset of regular (good) and anomalous (bad) trajectories of a dynamical system. We assume the training set to be labeled by human experts and that we have access only to a limited amount of data, typically noisy. We provide a systematic approach to synthesize both the syntactical structure and the parameters of the temporal logic formula using a two-steps procedure: first, we leverage a novel evolutionary algorithm for learning the structure of the formula; second, we perform the parameter synthesis operating on the statistical emulation of the average robustness for a candidate formula w.r.t. its parameters. We compare our results with our previous work [9] and with a recently proposed decision-tree [8] based method. We present experimental results on two case studies: an anomalous trajectory detection problem of a naval surveillance system and the characterization of an Ineffective Respiratory effort, showing the usefulness of our work.

## 1 Introduction

Learning temporal logic requirements from data is an emergent research field gaining momentum in the rigorous engineering of cyber-physical systems. Classical machine learning methods typically generate very powerful black-box (statistical) models. However, these models often do not help in the comprehension of the phenomenon they capture. Temporal logic provides a precise formal specification language that can easily be interpreted by humans. The possibility to describe datasets in a concise way using temporal logic formulas can thus help to better clarify and comprehend which are the emergent patterns for the system at hand. A clearcut example is the problem of anomaly detection, where the input is a set of trajectories describing regular or anomalous behaviors, and the goal is to learn a classifier that not only can be used to detect anomalous behaviors at runtime, but also gives insights on what characterizes an anomalous behavior. Learning temporal properties is also relevant in combination with state of the art techniques for search-based falsification of complex closed-loop

systems [6, 22, 23, 28], as it can provide an automatic way to describe desired (or unwanted behaviors) that the system needs to satisfy.

In this paper, we consider the problem of learning a temporal logic specification from a set of trajectories which are labeled by human experts (or by any other method which is not usable for real-time monitoring) as “good” for the normal expected behaviors and “bad” for the anomalous ones. The goal is to automatically synthesize both the structure of the formula and its parameters providing a temporal logic classifier that can discriminate as much as possible the bad and the good behaviors. This specification can be turned into a monitor that output a positive verdict for good behaviors and a negative verdict for bad ones.

**Related Work.** Mining temporal logic requirements is an emerging field of research in the analysis of cyber-physical systems (CPS) [2, 5, 7–9, 14, 16, 17, 20, 26, 27]. This approach is orthogonal to active automata learning (AAL) such as  $L^*$  Angluin’s algorithm [3] and its recent variants [15, 25]. AAL is suitable to capture the behaviours of black-box reactive systems and it has been successfully employed in the field of CPS to learn how to interact with the surrounding environments [10, 13]. Mining temporal logic requirements has the following advantages with respect to AAL. The first is that it does not require to interact with a reactive system. AAL needs to query the system in order to learn a Mealy machine representing the relation between the input provided and the output observed. Mining temporal logic requirements can be applied directly to a set of observed signals without the necessity to provide an input. The second is the possibility to use temporal logic requirements within popular tools (such as Breach [12] and S-TaLiRo [4]) for monitoring and falsification analysis of CPS models.

Most of the literature related to temporal logic inference from data focuses in particular on the problem of learning the optimal parameters given a specific template formula [5, 7, 14, 16, 20, 26, 27]. In [5], Asarin et al. extend the *Signal Temporal Logic* (STL) [18] with the possibility to express the time bounds of the temporal operators and the constants of the inequalities as parameters. They also provide a geometric approach to identify the subset of the parameter space that makes a particular signal to satisfy an STL specification. Xu et al. have recently proposed in [26] a temporal logic framework called *CensusSTL* for multi-agent systems that consists of an *outer logic STL formula* with a variable in the predicate representing the number of agents satisfying an *inner logic STL formula*. In the same paper the authors propose also a new inference algorithm similar to [5] that given the templates for both the *inner* and *outer* formulas, searches for the optimal parameter values that make the two formulas capturing the trajectory data of a group of agents. In [14] the authors use the same parametric STL extension in combination with the quantitative semantics of STL to perform a counter-example guided inductive parameter synthesis. This approach consists in iteratively generating a counterexample by executing a falsification tool for a template formula. The counterexample found at each step is then used to further refine the parameter set and the procedure terminates when no other

counterexamples are found. In general, all these methods, when working directly with raw data, are potentially vulnerable to the noise of the measurements and they are limited by the amount of data available.

Learning both the structure and the parameters of a formula from a dataset poses even more challenges [7–9, 17]. This problem is usually addressed in two steps, learning the structure of the formula and synthesizing its parameters. In particular, in [17] the structure of the formula is learned by exploring a directed acyclic graph and the method exploits *Support Vector Machine* (SVM) for the parameter optimization. In [8] the authors use instead a *decision tree* based approach for learning the formula, while the optimality is evaluated using heuristic impurity measures.

In our previous works [7, 9] we have also addressed the problem of learning both the structure and the parameters of a temporal logic specification from data. In [7] the structure of the formula is learned using a heuristics algorithm, while in [9] using a genetic algorithm. The synthesis of the parameters is instead performed in both cases exploiting the *Gaussian Process Upper Confidence Bound* (GP-UCB) [24] algorithm, statistically emulating the satisfaction probability of a formula for a given set of parameters. In both these methodologies, it is required to learn first a statistical model from the training set of trajectories. However, the statistical learning of this model can be a very difficult task and this is one of the main reason for proposing our new approach.

***Our Contribution.*** In this work, we consider the problem of mining the formula directly from a data set without requiring to learn a generative model from data. To achieve this goal, we introduce a number of techniques to improve the potentials of the genetic algorithm and to deal with the noise in the data in the absence of an underlying model.

First, instead of using the probability satisfaction as evaluator for the best formula, we design a discrimination function based on the quantitative semantics (or robustness) of STL and in particular the average robustness introduced in [6]. The average robustness enables us to differentiate among STL classifiers that have similar discriminating performance with respect to the data by choosing the most robust one. This gives us more information than just having the probability of satisfaction: for each trajectory, we can evaluate how much is it closed to the “boundary” of the classifier, instead of only checking whether it satisfies or not a formula. We then modify the discrimination function and the GP-UCB algorithm used in [7, 9] to better deal with noisy data and to use the quantitative semantics to emulate the average robustness distribution with respect to the parameter space of the formula.

Second, we reduce the computational cost of the *Evolutionary Algorithm* (EA) by using a lightweight configuration (i.e., a low threshold of max number of iterations) of the GP-UCB optimization algorithm to estimate the parameters of the formulas at each generation. The EA algorithm generates, as a final result, an STL formula tailored for classification purpose.

We compare our framework with our previous methodology [9] and with the decision-tree based approach presented in [8] on an anomalous trajectory detection problem of a naval surveillance and on an Assisted Ventilation in

Intensive Care Patients. Our experiments indicate that the proposed approach outperforms our previous work with respect to accuracy and show that it produces in general more compact, and easy to read, temporal logic specifications with respect to the decision-tree based approach with a comparable speed and accuracy.

**Paper Structure.** The rest of the paper is organized as follows: in the next section we present the *Signal Temporal Logic* and its robust semantics. We then introduce the problem considered in Section 3. In Sect. 4, we describe our approach. The results are presented in Sect. 5. Finally, we conclude the paper in Sect. 6, by discussing the implications of our contribution, both from the practical and the methodological aspect and some directions of improvement.

## 2 Signal Temporal Logic

**STL.** *Signal Temporal Logic* (STL) [18] is a formal specification language to express temporal properties over real-values trajectories with dense-time interval. For the rest of the paper, let be  $\mathbf{x} : \mathbb{T} \rightarrow \mathbb{R}^n$  a trace/trajectory, where  $\mathbb{T} = \mathbb{R}_{\geq 0}$  is the time domain,  $x_i(t)$  is the value at time  $t$  of the projection on the  $i^{\text{th}}$  coordinate, and  $\mathbf{x} = (x_1, \dots, x_n)$ , as an abuse on the notation, is used also to indicate the set of variables of the trace considered in the formulae.

**Definition 1 (STL syntax).** *The syntax of STL is given by*

$$\varphi := \top \mid \mu \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathcal{U}_I \varphi_2$$

where  $\top$  is the Boolean true constant,  $\mu$  is an atomic proposition, inequality of the form  $y(x) > 0$  ( $y : \mathbb{R}^n \rightarrow \mathbb{R}$ ), negation  $\neg$  and conjunction  $\wedge$  are the standard Boolean connectives, and  $\mathcal{U}_I$  is the Until temporal modality, where  $I$  is a real positive interval. As customary, we can derive the disjunction operator  $\vee$  and the future eventually  $\mathcal{F}_I$  and always  $\mathcal{G}_I$  operators from the until temporal modality ( $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ ,  $\mathcal{F}_I\varphi = \top \mathcal{U}_I\varphi$ ,  $\mathcal{G}_I\varphi = \neg\mathcal{F}_I\neg\varphi$ ).

STL can be interpreted over a trajectory  $\mathbf{x}$  using a qualitative (yes/no) or a quantitative (real-value) semantics [11, 18]. We report here only the quantitative semantics and we refer the reader to [11, 18, 19] for more details.

**Definition 2 (STL Quantitative Semantics).** *The quantitative satisfaction function  $\rho$  returns a value  $\rho(\varphi, \mathbf{x}, t) \in \bar{\mathbb{R}}$ ,<sup>1</sup> quantifying the robustness degree (or satisfaction degree) of the property  $\varphi$  by the trajectory  $\mathbf{x}$  at time  $t$ . It is defined recursively as follows:*

$$\begin{aligned} \rho(\top, \mathbf{x}, t) &= +\infty \\ \rho(\mu, \mathbf{x}, t) &= y(\mathbf{x}(t)) \text{ where } \mu \equiv y(\mathbf{x}(t)) \geq 0 \\ \rho(\neg\varphi, \mathbf{x}, t) &= -\rho(\varphi, \mathbf{x}, t) \\ \rho(\varphi_1 \wedge \varphi_2, \mathbf{x}, t) &= \min(\rho(\varphi_1, \mathbf{x}, t), \rho(\varphi_2, \mathbf{x}, t)) \\ \rho(\varphi_1 \mathcal{U}_{[a,b]}\varphi_2, \mathbf{x}, t) &= \sup_{t' \in t+[a,b]} (\min(\rho(\varphi_2, \mathbf{x}, t'), \inf_{t'' \in [t,t']} (\rho(\varphi_1, \mathbf{x}, t'')))) \end{aligned}$$

Moreover, we let  $\rho(\varphi, \mathbf{x}) := \rho(\varphi, \mathbf{x}, 0)$ .

<sup>1</sup>  $\bar{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ .

The sign of  $\rho(\varphi, \mathbf{x})$  provides the link with the standard Boolean semantics of [18]:  $\rho(\varphi, \mathbf{x}) > 0$  only if  $\mathbf{x} \models \varphi$ , while  $\rho(\varphi, \mathbf{x}) < 0$  only if  $\mathbf{x} \not\models \varphi$ <sup>2</sup>. The absolute value of  $\rho(\varphi, \mathbf{x})$ , instead, can be interpreted as a measure of the robustness of the satisfaction with respect to noise in signal  $\mathbf{x}$ , measured in terms of the maximal perturbation in the secondary signal  $y(\mathbf{x}(t))$ , preserving truth value. This means that if  $\rho(\varphi, \mathbf{x}, t) = r$  then for every signal  $\mathbf{x}'$  for which every secondary signal satisfies  $\max_t |y_j(t) - y'_j(t)| < r$ , we have that  $\mathbf{x}(t) \models \varphi$  if and only if  $\mathbf{x}'(t) \models \varphi$  (*correctness property*).

**PSTL.** *Parametric Signal Temporal Logic* [5] is an extension of STL that parametrizes the formulas. There are two types of formula parameters: temporal parameters, corresponding to the time bounds in the time intervals associated with temporal operators (e.g.  $a, b \in \mathbb{R}_{\geq 0}$ , with  $a < b$ , s.t.  $\mathcal{F}_{[a,b]}\mu$ ), and the threshold parameters, corresponding to the constants in the inequality predicates (e.g.,  $k \in \mathbb{R}, \mu = x_i > k$ , where  $x_i$  is a variable of the trajectory). In this paper, we allow only atomic propositions of the form  $\mu = x_i \bowtie k$  with  $\bowtie \in \{>, \leq\}$ . Given an STL formula  $\varphi$ , let  $\mathbb{K} = (\mathcal{T} \times \mathcal{C})$  be the *parameter space*, where  $\mathcal{T} \subseteq \mathbb{R}_{\geq 0}^{n_t}$  is the temporal parameter space ( $n_t$  being the number of temporal parameters), and  $\mathcal{C} \subseteq \mathbb{R}^{n_k}$  is the threshold parameter space ( $n_k$  being the number of threshold parameters). A  $\theta \in \mathbb{K}$  is a parameter configuration that induces a corresponding STL formula; e.g.,  $\varphi = \mathcal{F}_{[a,b]}(x_i > k)$ ,  $\theta = (0, 2, 3.5)$  then  $\varphi_\theta = \mathcal{F}_{[0,2]}(x_i > 3.5)$ .

**Stochastic Robustness.** Let us consider an unknown stochastic process  $(\mathbf{X}(t))_{t \in T} = (X_1(t), \dots, X_n(t))_{t \in T}$ , where each vector  $\mathbf{X}(t)$  corresponds to the state of the system at time  $t$ . For simplicity, we indicate the stochastic model with  $\mathbf{X}(t)$ .  $\mathbf{X}(t)$  is a measurable also as a random variable  $\mathbf{X}$  on the space  $D$ -valued *cadlag functions*  $\mathcal{D}([0, \infty), D)$ , here denoted by  $\mathcal{D}$ , assuming the domain  $\mathbb{D}$  to be fixed. It means that the set of trajectories  $\mathbf{x}$  of the stochastic process  $\mathbf{X}$  is the set  $\mathcal{D}$ . Let us consider now an STL formula  $\varphi$ , with predicates interpreted over state variables of  $\mathbf{X}$ . Given a trajectory  $\mathbf{x}(t)$  of a stochastic system, its robustness  $\rho(\varphi, \mathbf{x}, 0)$  is a measurable functional  $R_\varphi$  [6] from the trajectories in  $\mathcal{D}$  to  $\mathbb{R}$  which defines the real-valued random variable  $R_\varphi = R_\varphi(\mathbf{X})$  with probability distribution:

$$\mathbb{P}(R_\varphi(\mathbf{X}) \in [a, b]) = \mathbb{P}(\mathbf{X} \in \{\mathbf{x} \in \mathcal{D} \mid \rho(\varphi, \mathbf{x}, 0) \in [a, b]\}).$$

Such distribution of robustness degrees can be summarized by the average robustness degree. Fixing the stochastic process  $\mathbf{X}$ ,  $\mathbb{E}(R_\varphi|\mathbf{X})$ , it gives a measure of how strongly a formula is satisfied. The satisfaction is more robust when this value is higher. In this paper, we will approximate this expectation by Monte Carlo sampling.

### 3 Problem Formulation

In this paper, we focus our attention on learning the best property (or set of properties) that discriminates trajectories belonging to two different classes, say good

<sup>2</sup> The case  $\rho(\varphi, \mathbf{x}) = 0$ , instead, is a borderline case, and the truth of  $\varphi$  cannot be assessed from the robustness degree alone.

and bad, starting from a labeled dataset of observed trajectories. Essentially, we want to tackle a supervised two-class classification problem over trajectories, by learning a temporal logic discriminant, describing the temporal patterns that better separate two sets of observed trajectories.

The idea behind this approach is that there exists an unknown procedure that, given a temporal trajectory, is able to decide if the signal is good or bad. This procedure can correspond to many different things, e.g., to the reason of the failure of a sensor that breaks when it receives certain inputs. In general, as there may not be an STL specification that perfectly explains/mimics the unknown procedure, our task is to approximate it with the most effective one.

Our approach works directly with observed data, and avoids the reconstruction of an intermediate generative model  $p(\mathbf{x}|z)$  of trajectories  $x$  conditioned on their class  $z$ , as in [7, 9]. The reason is that such models can be hard to construct, even if they provide a powerful regularization, as they enable the generation of an arbitrary number of samples to train the logic classifier.

In a purely data-driven setting, to build an effective classifier, we need to consider that training data is available in limited amounts and it is typically noisy. This reflects in the necessity of finding methods that guarantee good generalization performance and avoid overfitting. In our context, overfitting can result in overly complex formulae, de facto encoding the training set itself rather than guessing the underlying patterns that separate the trajectories. This can be faced by using a score function based on robustness of temporal properties, combined with suitably designed regularizing terms.

We want to stress that the approach we present here, due to the use of the average robustness of STL properties, can be easily tailored to different problems, like finding the property that best characterise a single set of observations.

## 4 Methodology

Learning an STL formula can be separated in two optimization problems: the learning of the formula structure and the synthesis of the formula parameters. The structural learning is treated as a discrete optimization problem using an *Genetic Algorithm* (GA); the parameter synthesis, instead, considers a continuous parameter space and exploits an active learning algorithm, called *Gaussian Process Upper Confidence Bound* (GP-UCB). The two techniques are not used separately but combined together in a bi-level optimization. The GA acts externally by defining a set of template formulae. Then, the GP-UCB algorithm, which acts at the inner level, finds the best parameter configuration such that each template formula better discriminates between the two datasets. For both optimizations, we need to define a score function to optimize, encoding the criterion to discriminate between the two datasets.

Our implementation, called *RObustness GENetic* (ROGE) algorithm is described in Algorithm 1. First, we give an overview of it and then we described each specific function in the next subsections. The algorithm requires as input the dataset  $\mathcal{D}_p$ (good) and  $\mathcal{D}_n$ (bad), the parameter space  $\mathbb{K}$ , with the bound of



**Algorithm 1.** ROGE – ROBustness GENetic

---

**Require:**  $\mathcal{D}_p, \mathcal{D}_n, \mathbb{K}, Ne, Ng, \alpha, s$

- 1:  $gen \leftarrow \text{GENERATEINITIALFORMULAE}(Ne, s)$
- 2:  $gen_\theta \leftarrow \text{LEARNINGPARAMETERS}(gen, G, \mathbb{K})$
- 3: **for**  $i = 1 \dots Ng$  **do**
- 4:    $subg_\theta \leftarrow \text{SAMPLE}(gen_\theta, F)$
- 5:    $newg \leftarrow \text{EVOLVE}(subg_\theta, \alpha)$
- 6:    $newg_\theta \leftarrow \text{LEARNINGPARAMETERS}(newg, G, \mathbb{K})$
- 7:    $gen_\theta \leftarrow \text{SAMPLE}(newg_\theta \cup gen_\theta, F)$
- 8: **end for**
- 9: **return**  $gen_\theta$

---

each considered variable, the size of the initial set of formulae  $Ne$ , the number of maximum iterations  $Ng$ , the mutation probability  $\alpha$  and the maximum initial formula size  $s$ . The algorithm starts generating (line 1) an initial set of PSTL formulae, called  $gen$ . For each of these formulae (line 2), the algorithm learns the best parameters accordingly to a discrimination function  $G$ . We call  $gen_\theta$  the generation for which we know the best parameters of each formula. During each iteration, the algorithm (line 4), guided by a fitness function  $F$ , extracts a subset  $subg_\theta$  composed by the best  $Ne/2$  formulae, from the initial set  $gen_\theta$ . From this subset (line 5), a new set  $newg$  composed of  $Ne$  formulae is created by employing the *EVOLVE* routine, which implements two *genetic* operators. Then (line 6), as in line 2, the algorithm identifies the best parameters of each formula belonging to  $newg$ . The new generation  $newg_\theta$  and the old generation  $gen_\theta$  are then merged together (line 7). From this set of  $2Ne$  formulae the algorithm extracts, with respect to the fitness function  $F$ , the new generation  $gen_\theta$  of  $Ne$  formulae. At the end of the iterations or when the *STOP* criterion is true (lines 8–12), the algorithm returns the last generated formulae. The best formula is the one with the highest value of the discrimination function, i.e.,  $\varphi_{best} = \text{argmax}_{\varphi_\theta \in gen_\theta} (G(\varphi_\theta))$ . We describe below in order: the discrimination function algorithm, the learning of the parameters of a formula template and the learning of the formula structure.

#### 4.1 Discrimination Function $G(\varphi)$

We have two datasets  $\mathcal{D}_p$  and  $\mathcal{D}_n$  and we search for the formula  $\varphi$  that better separates these two classes. We define a function able to discriminate between this two datasets, such that maximising this *discrimination function* corresponds to finding the best formula classifier. In this paper, we decide to use, as evaluation of satisfaction of each formula, the quantitative semantics of STL. As described in Sect. 2, this semantics computes a real-value (robustness degree) instead of only a yes/no answer.

Given a dataset  $\mathcal{D}_i$ , we assume that the data comes from an unknown stochastic process  $\mathbf{X}_i$ . The process in this case is like a black-box for which we observe

only a subset of trajectories, the dataset  $\mathcal{D}_i$ . We can then estimate the averages robustness  $\mathbb{E}(R_\varphi|\mathbf{X}_i)$  and its variance  $\sigma^2(R_\varphi|\mathbf{X}_i)$ , averaging over  $\mathcal{D}_i$ .

To discriminate between the two dataset  $\mathcal{D}_p$  and  $\mathcal{D}_n$ , we search the formula  $\varphi$  that maximizes the difference between the average robustness of  $\mathbf{X}_p$ ,  $\mathbb{E}(R_\varphi|\mathbf{X}_p)$ , and the average robustness of  $\mathbf{X}_n$ ,  $\mathbb{E}(R_\varphi|\mathbf{X}_n)$  divided by the sum of the respective standard deviation:

$$G(\varphi) = \frac{\mathbb{E}(R_\varphi|\mathbf{X}_p) - \mathbb{E}(R_\varphi|\mathbf{X}_n)}{\sigma(R_\varphi|\mathbf{X}_p) + \sigma(R_\varphi|\mathbf{X}_n)}. \quad (1)$$

This formula is proportional to the probability that a new point sampled from the distribution generating  $\mathbf{X}_p$  or  $\mathbf{X}_n$ , will belong to one set and not to the other. In fact, an higher value of  $G(\varphi)$  implies that the two average robustness will be sufficiently distant relative to their length-scale, given by the standard deviation  $\sigma$ .

As said above, we can evaluate only a statistical approximation of  $G(\varphi)$  because we have a limited number of samples belonging to  $\mathbf{X}_p$  and  $\mathbf{X}_n$ . We will see in the next paragraph how we tackle this problem.

## 4.2 GB-UCP: Learning the Parameters of the Formula

Given a formula  $\varphi$  and a parameter space  $\mathbb{K}$ , we want to find the parameter configuration  $\theta \in \mathbb{K}$  that maximises the score function  $g(\theta) := G(\varphi_\theta)$ , considering that the evaluations of this function are noisy. So, the question is how to best estimate (and optimize) an unknown function from observations of its value at a finite set of input points. This is a classic non-linear non-convex optimization problem that we tackle by means of the GP-UCB algorithm [24]. This algorithm interpolates the noisy observations using a stochastic process (a procedure called emulation in statistics) and optimize the emulated function using the uncertainty of the fit to determine regions where the true maximum can lie. More specifically, the GP-UCB bases its emulation phase on Gaussian Processes, a Bayesian non-parametric regression approach [21], adding candidate maximum points to the training set of the GP in an iterative fashion, and terminating when no improvement is possible (see [24] for more details).

After this optimization, we have found a formula that separates the two datasets, not from the point of view of the satisfaction (yes/no) of the formula but from the point of view of the robustness value. In other words, there is a threshold value  $\alpha$  such that  $\mathbb{E}(R_\varphi|\mathbf{X}_p) > \alpha$  and  $\mathbb{E}(R_\varphi|\mathbf{X}_n) \leq \alpha$ . Now, we consider the new STL formula  $\varphi'$  obtained translating the atomic predicates of  $\varphi$  by  $\alpha$  (e.g.,  $y(x) > 0$  becomes  $y(x) - \alpha > 0$ ). Taking into account that the quantitative robustness is achieved by combination of min, max, inf and sup, which are linear algebraic operators with respect to translations (e.g.,  $\min(x, y) \pm c = \min(x \pm c, y \pm c)$ ), we easily obtain that  $\mathbb{E}(R_{\varphi'}|\mathbf{X}_p) > 0$  and  $\mathbb{E}(R_{\varphi'}|\mathbf{X}_n) < 0$ . Therefore,  $\varphi'$  will divide the two datasets also from the point of view of the satisfaction.

### 4.3 Genetic Algorithm: Learning the Structure of the Formula

To learn the formula structure, we exploit a modified version of the Genetic Algorithm (GA) presented in [9]. GAs belongs to the larger class of evolutionary algorithms (EA). They are used for search and optimization problems. The strategy takes inspiration from the genetic area, in particular in the selection strategy of species. Let us see now in detail the genetic routines of the ROGE algorithm.

$gen = \mathbf{GENERATEINITIALFORMULAE}(Ne, s)$ . This routine generates the initial set of  $Ne$  formulae. A fraction  $n_l < Ne$  of this initial set is constructed by a subset of the temporal properties:  $\mathcal{F}_I\mu$ ,  $\mathcal{G}_I\mu$ ,  $\mu_1\mathcal{U}_I\mu_2$ , where the atomic predicates are of the form  $\mu = (x_i > k)$  or  $\mu = (x_i \leq k)$  or simple boolean combinations of them (e.g.  $\mu = (x_i > k_i) \wedge (x_j > k_j)$ ). The size of this initial set is exponential accordingly to the number of considered variables  $x_i$ . If we have few variables we can keep all the generated formulae, whereas if the number of variables is large we consider only a random subset. The remain  $n_r = Ne - n_l$  formulae are chosen randomly from the set of formulae with a maximum size defined by the input parameter  $s$ . This size can be adapted to have a wider range of initial formulae.

$subg_\Theta = \mathbf{SAMPLE}(gen_\Theta, F)$ . This procedure extracts from  $gen_\Theta$  a subset  $subg_\Theta$  of  $Ne/2$  formulae, according to a fitness function  $F(\varphi) = G(\varphi) - S(\varphi)$ . The first factor  $G(\varphi)$  is the discrimination function previously defined and  $S(\varphi)$  is a size penalty, i.e. a function penalizes formulae with large sizes. In this paper, we consider  $S(\varphi) = g \cdot p^{size(\varphi)}$ , where  $p$  is heuristically set such that  $p^5 = 0.5$ , i.e. formulae of size 5 get a 50% penalty, and  $g$  is adaptively computed as the average discrimination in the current generation. An alternative choice of  $p$  can be done by cross-validation.

$newg = \mathbf{EVOLVE}(subg_\Theta, \alpha)$ . This routine defines a new set of formulae ( $newg$ ) starting from  $subg_\Theta$ , exploiting two *genetic* operators: the *recombination* and the *mutation* operator. The recombination operator takes as input two formulae  $\varphi_1, \varphi_2$  (the parents), it randomly chooses a subtree from each formula and it swaps them, i.e. it assigns the subtree of  $\varphi_1$  to  $\varphi_2$  and viceversa. As a result, the two generated formulae (the children) share different subtrees of the parents' formulae. The mutation operator takes as input one formula and induce a change on a randomly selected node of its tree-structure. Specifically, it substitutes the node with a random formula (atomic predicates  $\mu$  or temporal properties of the form  $\mathcal{F}_I\mu$ ,  $\mathcal{G}_I\mu$ ,  $\mu_1\mathcal{U}_I\mu_2$ ). The purpose of the genetic operators is to introduce innovation in the offspring population which leads to the optimization of a target function ( $G$  in this case). In particular, recombination is related to exploitation, whereas mutation to exploration. The EVOLVE routine implements an iterative loop that at each iteration selects which genetic operators to apply. A number  $p \in [0, 1]$  is randomly sampled. If its value is lower than the mutation probability, i.e.,  $p \leq \alpha$ , then the mutation is selected, otherwise a recombination is performed. At this point the input formulae of the selected genetic operator are chosen randomly between the formulas composing  $subg_\Theta$  and the genetic operators are applied. In our implementation, we give more importance to the exploitation;

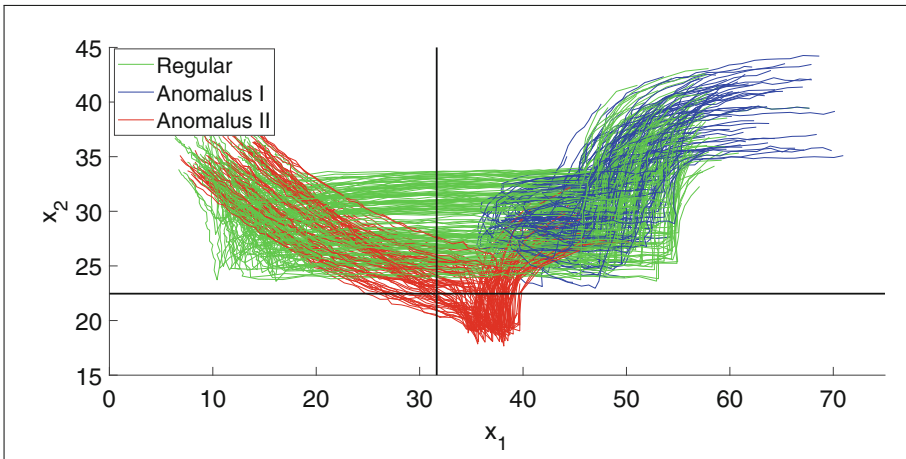
therefore the mutation acts less frequently than the recombination (i.e.,  $\alpha \leq 0.1$ ). The iteration loops will stop when the number of generated formula is equal to  $Ne$ , i.e. twice the cardinality of  $subg_{\Theta}$ .

## 5 Case Studies and Experimental Results

### 5.1 Maritime Surveillance

As first case study, We consider the maritime surveillance problem presented in [8] to compare our framework with their *Decision Tree* (DTL4STL) approach. The experiments with the DTL4STL approach were implemented in MATLAB, the code is available at [1]. We also test our previous implementation presented in [9] with the same benchmark. Both the new and the previous learning procedure were implemented in Java (JDK 1.8\_0) and run on a Dell XPS, Windows 10 Pro, Intel Core i7-7700HQ 2.8 GHz, 8 GB 1600 MHz memory.

The synthetic dataset of naval surveillance reported in [8] consists of 2-dimensional coordinates traces of vessels behaviours. It considers two kind of anomalous trajectories and regular trajectories, as illustrated in Fig. 1. The dataset contains 2000 total trajectories (1000 normal and 1000 anomalous) with 61 sample points per trace. We run the experiments using a 10-fold cross-validation in order to collect the mean and variance of the misclassified trajectories of the validation set. Results are summarized in Table 1, where we report also the average execution time. We test also our previous implementation [9] without a generative model from data. It performs so poorly on the chosen benchmark that is not meaningful to report it: the misclassification rate



**Fig. 1.** The regular trajectories are represented in green. The anomalous trajectories which are of two kinds, are represented respectively in blue and red. (Color figure online)

**Table 1.** The comparison of the computational time (in sec), the mean misclassification rate and its standard deviation between the learning procedure using the RObust GENetic algorithm, the Decision-Tree (DTL4STL) MATLAB code provided by the authors and the results reported in [8] (DTL4STL<sub>p</sub>).

	ROGE	DTL4STL	DTL4STL <sub>p</sub>
Mis. Rate	0	0.01 ± 0.013	0.007 ± 0.008
Comp. Time	73 ± 18	144 ± 24	-

on the validation set is around 50%. In Table 1, we also report DTL4STL<sub>p</sub> the DTL4STL performance declared in [8], but we were not be able to reproduce them in our setting.

In terms of accuracy our approach is comparable with respect to the performance of the DTL4STL. In terms of computational cost, the decision tree approach is slightly slower but it is implemented in MATLAB rather than Java.

An example of formula that we learn with ROGE is

$$\varphi = ((x_2 > 22.46) \mathcal{U}_{[49,287]} (x_1 \leq 31.65)) \quad (2)$$

The formula (2) identifies all the regular trajectories, remarkably resulting in a misclassification error equal to zero, as reported in Table 1. The red anomalous trajectories falsify the predicate  $x_2 > 22.46$  before verifying  $x_1 \leq 31.65$ , on the contrary the blue anomalous trajectories globally satisfy  $x_2 > 22.46$  but never verify  $x_1 \leq 31.65$  (consider that all the vessels start from the top right part of the graphic in Fig. 1). Both these conditions result in the falsification of Formula (2), which on the contrary is satisfied by all the regular trajectories. In Fig. 1, we have reported the threshold  $x_2 = 22.46$  and  $x_1 = 31.65$ .

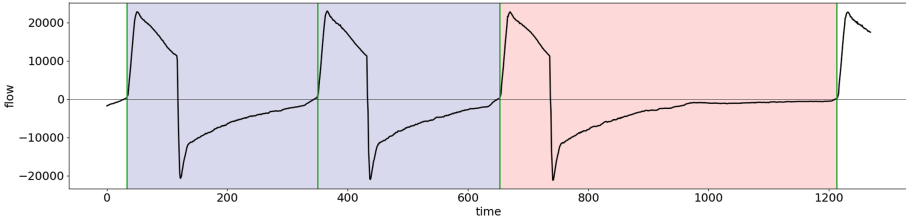
An example instead of formula found by the DTL4STL tool using the same dataset is the following:

$$\begin{aligned} \psi = & (((\mathcal{G}_{[187,196]} x_1 < 19.8) \wedge (\mathcal{F}_{[55.3,298]} x_1 > 40.8)) \vee ((\mathcal{F}_{[187,196]} x_1 > 19.8) \\ & \wedge ((\mathcal{G}_{[94.9,296]} x_2 < 32.2) \vee ((\mathcal{F}_{[94.9,296]} x_2 > 32.2) \wedge (((\mathcal{G}_{[50.2,274]} x_2 > 29.6) \\ & \wedge (\mathcal{G}_{[125,222]} x_1 < 47)) \vee ((\mathcal{F}_{[50.2,274]} x_2 < 29.6) \wedge (\mathcal{G}_{[206,233]} x_1 < 16.7)))))) \end{aligned}$$

The specific formula generated using ROGE is simpler than the formula generated using DTL4STL and indeed it is easier to understand it. Furthermore DTL4STL does not consider the until operator in the set of primitives (see [8] for more details).

## 5.2 Ineffective Inspiratory Effort (IIE)

The Ineffective Inspiratory Effort (IIE) is one of the major problems concerning the mechanical ventilation of patients in intensive care suffering from respiratory failure. The mechanical ventilation uses a pump to help the patient in the process of inspiration and expiration, controlling the *flow* of air into the lungs and the



**Fig. 2.** Example of regular (blue regions) and ineffective (red region) breaths. (Color figure online)

airway pressure, called *paw*. Inspiration is characterised by growth of pressure up to the selected *paw* value and positive *flow*, while expiration has a negative *flow* and a drop *paw*. The exact dynamics of these respiratory curves strictly depends on patient and on ventilatory modes. We can see an example in Fig. 2 of two regular (blue regions) and one ineffective (red region) breaths. An IIE occurs when the patients tries to inspire, but its effort is not sufficient to trigger the appropriate reaction of the ventilator. This results in a longer expiration phase. Persistence of IIE may cause permanent damages of the respiratory system.

An early detection of anomalies using low-cost methods is a key challenge to prevent IIE conditions and still an open problem. In [9], starting with a dataset of discrete time series and sampled *flow* values (labeled good and bad) from a single patient, the authors statically designed generative models of *flow* and of its derivative *flow'*, for regular and ineffective signals. Then they used the simulations of such models to identify the best formula/formulae discriminating between them. In that contribution trajectories with different lengths are considered, treating as false trajectories that are too short to verify a given formula, and use this to detect the length of trajectories and separate anomalous breaths which last longer than regular ones. However, using the information about the duration of a breath is not advisable if the goal is to monitor the signals at runtime and detect the IIE at their onset. For this reason, in our approach we consider a new dataset, cutting the original trajectories used to generate the stochastic model in Bufo et al. [9] to a maximum common length of the order of 2s. We also apply a moving average filter to reduce the noise in the computation of *flow'*.

Specifically, the new dataset consists of 2-dimensional traces of *flow* and its derivative, *flow'*, containing a total of 288 trajectories (144 normal and 144 anomalous) with 134 sample points per trace. The time scale is in hundredths of a second. We run the experiments using a 6-fold cross-validation and report our results and comparison on the new dataset with DTL4STL [8] in Table 2.

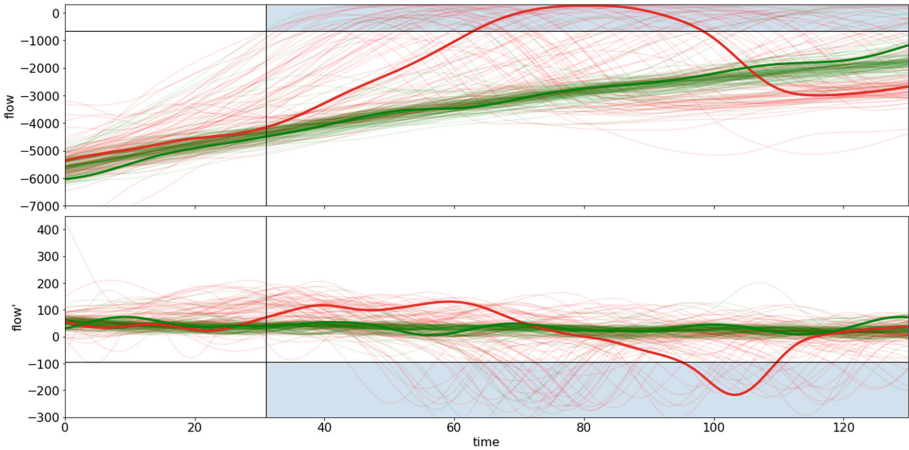
An example of formula that we learn with ROGE is:

$$\varphi = \mathcal{F}_{[31,130]}((\text{flow} \geq -670) \vee (\text{flow}' \leq -94)) \quad (3)$$

Formula  $\varphi$  identifies the anomalous trajectories, stating that at a time between 31s and 130s either *flow* is higher than  $-670$  or *flow'* is below  $-94$ .

**Table 2.** The comparison of the computational time (in sec), the mean misclassification rate and its standard deviation between the learning procedure using the ROust Genetic algorithm.

	ROGE	DTL4STL
Mis. Rate	$0.17 \pm 0.01$	$0.23 \pm 0.07$
False. Pos	$0.20 \pm 0.02$	$0.23 \pm 0.07$
False. Neg	$0.14 \pm 0.02$	$0.20 \pm 0.15$
Comp. Time	$65 \pm 7$	$201 \pm 7$



**Fig. 3.** 100 regular (green) and 100 ineffective (red) flow and  $flow'$  trajectories during the expiration phase. The light blue rectangles correspond to the satisfaction area of formula (3). One regular (green) and one ineffective (red) trajectories are showed thicker. (Color figure online)

This is in accordance with the informal description of an IIE, principally caused by an unexpected increment of the  $flow$  during the expiration phase followed by a rapid decrease. Indeed, one of the main characteristic of an IIE is the presence of a small hump in the  $flow$  curve during this phase. In Fig. 3 we report some of the trajectories of the dataset, showing the areas where the property is satisfied.

On average, formulae found by the DTL4STL tool on the new dataset are disjunctions or conjunctions of 10 eventually or always subformulae, which are not readily interpretable.

Our approach on the new dataset is better in term of accuracy and computation time with an improvement of 22% and 67%, respectively.

Similarly to the previous test case, we compare our approach with [9], performed directly on the dataset, and not on the generative model. That approach performs so poorly also in this benchmark, obtaining a misclassification rate of 0.47, which is comparable with a random classifier. The problem here is that



the methods proposed in [9], differently from the one presented here, relies on a large number of model simulations, and it is not suited to work directly with observed data.

If we give up to online monitoring and consider only full breaths, we can improve classification by rescaling their duration into  $[0, 1]$ , so that each breath lasts the same amount of time. In this case, we learn a formula with misclassification rate of  $0.05 \pm 0.01$ , while DTL4STL reaches a misclassification rate of  $0.07 \pm 0.02$ . This suggests that the high variability of durations of ineffective breaths has to be treated more carefully.

## 6 Conclusion

We present a framework to learn from a labeled dataset of normal and anomalous trajectories a *Signal Temporal Logic* (STL) specification that better discriminates among them. In particular, we design a Robust Genetic algorithm (ROGE) that combines an *Evolutionary Algorithm* (EA) for learning the structure of the formula and the *Gaussian Process Upper Confidence Bound* algorithm (GP-UCB) for synthesizing the formula parameters. We compare ROGE with our previous work [9] and with the Decision Tree approach presented in [8] on an anomalous trajectory detection problem of a maritime surveillance system and on an Ineffective Inspiratory Effort example.

A significant difference concerning our previous approach [7,9] is that we avoid reconstructing a generative statistical model from the dataset. Furthermore, we modified both the structure and parameter optimization procedure of the genetic algorithm, which now relies on the robustness semantics of STL. This structural improvement was necessary considering that a naive application of our previous approach [9] directly on datasets performs very poorly.

We compare our method also with the Decision Tree (DTL4STL) approach of [8] obtaining a low misclassification rate and producing smaller and more interpretable STL specifications. Furthermore, we do not restrict the class of the temporal formula to only *eventually* and *globally* and we allow arbitrary temporal nesting.

Our genetic algorithm can get wrong results if the formulae of the initial generation are chosen entirely randomly. We avoid this behavior by considering simpler formulae from the beginning as a result of the GENERATEINITIALFORMULAE routine. This approach is a form of regularization and resembles the choice of the set of primitive of DTL4STL.

As future work, we plan to develop an iterative method which uses the proposed genetic algorithm and the robustness of STL to reduce the misclassification rate of the generated formula. The idea is to use the robustness value of a learned formula  $\varphi$  to identify the region of the trajectory space  $\mathcal{D}$  where the generated formula  $\varphi$  has a high accuracy, i.e. trajectories whose robustness is greater than a positive threshold  $h^+$  or smaller than a negative threshold  $h^-$ . These thresholds can be identified by reducing the number of false positives or false negatives. We can then train a new STL classifier  $\varphi'$  on the remaining trajectories, having small



robustness for  $\varphi$ . This will create a hierarchical classifier, that first tests on  $\varphi$ , if robustness is too low it tests on  $\varphi'$ , and so on. The depth of such classification is limited only by the remaining data at each step. The method can be further strengthened by relying on bagging to generate an ensemble of classifiers at each level, averaging their predictions or choosing the best answer, i.e. the one with larger robustness.

**Acknowledgment.** E.B. and L.N. acknowledge the partial support of the Austrian National Research Network S 11405-N23 (RiSE/SHiNE) of the Austrian Science Fund (FWF). E.B., L.N. and S.S. acknowledge the partial support of the ICT COST Action IC1402 (ARVI).

## References

1. DTL4STL (2016). <http://sites.bu.edu/hyness/dtl4stl/>
2. Ackermann, C., Cleaveland, R., Huang, S., Ray, A., Shelton, C., Latronico, E.: Automatic requirement extraction from test cases. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 1–15. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-16612-9\\_1](https://doi.org/10.1007/978-3-642-16612-9_1)
3. Angluin, D.: Learning regular sets from queries and counterexamples. *Inf. Comput.* **75**(2), 87–106 (1987)
4. Annpureddy, Y., Liu, C., Fainekos, G., Sankaranarayanan, S.: S-TALiRO: a tool for temporal logic falsification for hybrid systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 254–257. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-19835-9\\_21](https://doi.org/10.1007/978-3-642-19835-9_21)
5. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-29860-8\\_12](https://doi.org/10.1007/978-3-642-29860-8_12)
6. Bartocci, E., Bortolussi, L., Nenzi, L., Sanguinetti, G.: System design of stochastic models using robustness of temporal properties. *Theor. Comput. Sci.* **587**, 3–25 (2015)
7. Bartocci, E., Bortolussi, L., Sanguinetti, G.: Data-driven statistical learning of temporal logic properties. In: Legay, A., Bozga, M. (eds.) FORMATS 2014. LNCS, vol. 8711, pp. 23–37. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10512-3\\_3](https://doi.org/10.1007/978-3-319-10512-3_3)
8. Bombara, G., Vasile, C.I., Penedo, F., Yasuoka, H., Belta, C.: A decision tree approach to data classification using signal temporal logic. In: Proceedings of HSCC, pp. 1–10 (2016)
9. Bufo, S., Bartocci, E., Sanguinetti, G., Borelli, M., Lucangelo, U., Bortolussi, L.: Temporal logic based monitoring of assisted ventilation in intensive care patients. In: Margaria, T., Steffen, B. (eds.) ISoLA 2014. LNCS, vol. 8803, pp. 391–403. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-662-45231-8\\_30](https://doi.org/10.1007/978-3-662-45231-8_30)
10. Chen, Y., Tumova, J., Ulusoy, A., Belta, C.: Temporal logic robot control based on automata learning of environmental dynamics. *Int. J. Robot. Res.* **32**(5), 547–565 (2013)
11. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39799-8\\_19](https://doi.org/10.1007/978-3-642-39799-8_19)

12. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-14295-6\\_17](https://doi.org/10.1007/978-3-642-14295-6_17)
13. Fu, J., Tanner, H.G., Heinz, J., Chandlee, J.: Adaptive symbolic control for finite-state transition systems with grammatical inference. *IEEE Trans. Autom. Control* **59**(2), 505–511 (2014)
14. Hoxha, B., Dokhanchi, A., Fainekos, G.E.: Mining parametric temporal logic properties in model-based design for cyber-physical systems. *STTT* **20**(1), 79–93 (2018)
15. Isberner, M., Howar, F., Steffen, B.: The TTT algorithm: a redundancy-free approach to active automata learning. In: Bonakdarpour, B., Smolka, S.A. (eds.) RV 2014. LNCS, vol. 8734, pp. 307–322. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-11164-3\\_26](https://doi.org/10.1007/978-3-319-11164-3_26)
16. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. *IEEE Trans. CAD Integr. Circuits Syst.* **34**(11), 1704–1717 (2015)
17. Kong, Z., Jones, A., Belta, C.: Temporal logics for learning and detection of anomalous behavior. *IEEE Trans. Autom. Control* **62**(3), 1210–1222 (2017)
18. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS/FTRTFT -2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30206-3\\_12](https://doi.org/10.1007/978-3-540-30206-3_12)
19. Maler, O., Nickovic, D.: Monitoring properties of analog and mixed-signal circuits. *STTT* **15**(3), 247–268 (2013)
20. Nguyen, L.V., Kapinski, J., Jin, X., Deshmukh, J.V., Butts, K., Johnson, T.T.: Abnormal data classification using time-frequency temporal logic. In: Proceedings of HSCC, pp. 237–242 (2017)
21. Rasmussen, C.E., Williams, C.K.I.: *Gaussian Processes for Machine Learning*. MIT Press, Cambridge (2006)
22. Sankaranarayanan, S., Kumar, S.A., Cameron, F., Bequette, B.W., Fainekos, G.E., Maahs, D.M.: Model-based falsification of an artificial pancreas control system. *SIGBED Rev.* **14**(2), 24–33 (2017)
23. Silveti, S., Policriti, A., Bortolussi, L.: An active learning approach to the falsification of black box cyber-physical systems. In: Polikarpova, N., Schneider, S. (eds.) IFM 2017. LNCS, vol. 10510, pp. 3–17. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-66845-1\\_1](https://doi.org/10.1007/978-3-319-66845-1_1)
24. Srinivas, N., Krause, A., Kakade, S.M., Seeger, M.W.: Information-theoretic regret bounds for gaussian process optimization in the bandit setting. *IEEE Trans. Inf. Theory* **58**(5), 3250–3265 (2012)
25. Steffen, B., Howar, F., Isberner, M.: Active automata learning: from DFAs to interface programs and beyond. In: Proceedings of ICGI 2012, pp. 195–209 (2012)
26. Xu, Z., Julius, A.A.: Census signal temporal logic inference for multiagent group behavior analysis. *IEEE Trans. Autom. Sci. Eng.* **15**(1), 264–277 (2018)
27. Zhou, J., Ramanathan, R., Wong, W.-F., Thiagarajan, P.S.: Automated property synthesis of ODEs based bio-pathways models. In: Feret, J., Koepl, H. (eds.) CMSB 2017. LNCS, vol. 10545, pp. 265–282. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-67471-1\\_16](https://doi.org/10.1007/978-3-319-67471-1_16)
28. Zutshi, A., Sankaranarayanan, S., Deshmukh, J.V., Kapinski, J., Jin, X.: Falsification of safety properties for closed loop control systems. In: Proceedings of HSCC, pp. 299–300 (2015)



# A Hemimetric Extension of Simulation for Semi-Markov Decision Processes

Mathias Ruggaard Pedersen<sup>(✉)</sup>, Giorgio Bacci, Kim Guldstrand Larsen,  
and Radu Mardare

Department of Computer Science, Aalborg University, Aalborg, Denmark  
mrp@cs.aau.dk

**Abstract.** Semi-Markov decision processes (SMDPs) are continuous-time Markov decision processes where the residence-time on states is governed by generic distributions on the positive real line.

In this paper we consider the problem of comparing two SMDPs with respect to their time-dependent behaviour. We propose a hemimetric between processes, which we call *simulation distance*, measuring the least acceleration factor by which a process needs to speed up its actions in order to behave at least as fast as another process. We show that this distance can be computed in time  $\mathcal{O}(n^2(f(l) + k) + mn^7)$ , where  $n$  is the number of states,  $m$  the number of actions,  $k$  the number of atomic propositions, and  $f(l)$  the complexity of comparing the residence-time between states. The theoretical relevance and applicability of this distance is further argued by showing that (i) it is suitable for compositional reasoning with respect to CSP-like parallel composition and (ii) has a logical characterisation in terms of a simple Markovian logic.

## 1 Introduction

Semi-Markov decision processes (SMDPs) are Markovian stochastic decision processes modelling the firing time of transitions via real-valued random variables describing the residence-time on states. Semi-Markov decision processes provide a more permissive model than continuous-time Markov decision processes, since they allow as residence-time distributions any generic distribution on the positive real line, rather than only exponential ones. The generality offered by SMDPs has been found useful in modelling several real-case scenarios. Successful examples include power plants [19] and power supply units [20], to name a few.

When considering such real-time stochastic processes, non-functional requirements are important, particularly requirements like response time and throughput, which depend on the timing behaviour of the process. We therefore wish to understand and be able to compare the timing behaviour of different processes.

To cope with the need for comparing the timing behaviour of different systems, in this paper we propose and study a quantitative extension of the simulation relation by Baier et al. [2], called  $\varepsilon$ -simulation, which puts the focus on the timing aspect of processes. The intuition is that a process  $s_2$   $\varepsilon$ -*simulates* another

process  $s_1$  if after accelerating the actions of  $s_2$  by a factor  $\varepsilon > 0$  it reacts to the inputs from the external environment as  $s_1$  with at least the same speed.

This type of quantitative reasoning is not new in the literature, and it dates back to the seminal work of Jou and Smolka [9, 14], who proposed the concept of probabilistic  $\varepsilon$ -bisimulation. This line of work has led to much work on probabilistic bisimulation distances [5, 7, 8]. While our work is conceptually similar to the bisimulation distances, it is technically very different. This is because bisimulation distances are constructed from a coalgebraic view as fixed points of operators. However, for the kind of timed systems that we are investigating, the coalgebraic perspective is much less understood. Moreover, since our distance generalises a preorder relation and not a congruence as the other distances do, it is not symmetric, which brings in new technical challenges.

Following the work of Jou and Smolka, our notion of  $\varepsilon$ -simulation naturally induces a distance between processes: For any pair of states  $s_1$  and  $s_2$ , we define their *simulation distance* as the least acceleration factor needed by  $s_2$  to speed up its actions in order to behave at least as fast as  $s_1$ . This definition does not provide a distance in the usual sense, but rather a *multiplicative hemimetric*, i.e. an asymmetric notion of distance satisfying a multiplicative version of the triangle inequality. Such a notion is not new, as it is extensively applied in the context of differential privacy to measure information leakage of systems (see e.g. [1, 4]).

The theoretical relevance and applicability of the simulation distance is argued by means of the following results, which are the main technical contributions of the paper:

1. We provide an algorithm for computing the simulation distance between arbitrary states of an SMDP running in time  $\mathcal{O}(n^2(f(l) + k) + mn^7)$ , where  $n$  is the number of states,  $m$  the number of actions,  $k$  the number of atomic propositions, and  $f(l)$  the complexity of comparing the residence time distributions on states.
2. We show that under some mild conditions on how residence-time distributions are combined in the parallel composition of two states, CSP-like parallel composition of SMDPs is non-expansive with respect to our hemimetric. This shows that the simulation distance is suitable for compositional reasoning.
3. We provide a logical characterisation of the distance in terms of a simple Markovian logic, stating that the distance from  $s_1$  to  $s_2$  is less than or equal to  $\varepsilon$  if and only if  $s_2$  satisfies the  $\varepsilon$ -perturbation of any logical property that  $s_1$  satisfies. Moreover, we prove that  $\varepsilon$ -simulation preserves the  $\varepsilon$ -perturbation of time-bounded reachability properties.

A full version of the paper containing all the technical proofs can be found at [18].

**Notation and Preliminaries.** Let  $\mathbb{N}$  denote the natural numbers,  $\mathbb{Q}_{\geq 0}$  the non-negative rational numbers,  $\mathbb{R}_{\geq 0}$  the non-negative real numbers, and  $\mathbb{R}_{> 0}$  the strictly positive ones. Given a set  $X$ , we will denote by  $\mathcal{D}(X)$  the set of all probability measures on  $X$ . If  $\mu \in \mathcal{D}(\mathbb{R}_{\geq 0})$ , then the *cumulative distribution*

function (CDF) will be denoted by  $F_\mu$  and is given by  $F_\mu(t) = \mu([0, t])$ . For  $x \in \mathbb{R}_{\geq 0}$ , we will write  $\delta_x$  for the Dirac measure at  $x$ , which is defined by  $\delta_x(E) = 1$  if  $x \in E$  and  $\delta_x(E) = 0$  otherwise. For any  $\theta \in \mathbb{R}_{> 0}$ , we will write  $Exp[\theta]$  for the CDF of an exponential distribution with rate  $\theta$ , and for  $a, b \in \mathbb{R}_{\geq 0}$  such that  $a < b$ , we will write  $Unif[a, b]$  for the CDF of a uniform distribution.

We will use the convention that  $\infty + x = \infty$  for  $x \in \mathbb{R}_{\geq 0}$  and  $\infty \cdot y = \infty$  for  $y \in \mathbb{R}_{> 0}$ . A function  $d: X \times X \rightarrow \mathbb{R}_{\geq 0} \cup \{\infty\}$  is called a *hemimetric* if it satisfies  $d(x, x) = 0$  and the triangle inequality  $d(x, z) \leq d(x, y) + d(y, z)$ . It is called a *pseudometric* if it is also symmetric, i.e.  $d(x, y) = d(y, x)$ , and it is called a *metric* if it is symmetric and furthermore  $d(x, y) = 0$  if and only if  $x = y$ .

## 2 Semi-Markov Decision Processes

In this section, we introduce semi-Markov decision processes, which are continuous-time reactive probabilistic systems. A semi-Markov decision process has residence time on states governed by generic distributions on the positive real line and reacts to inputs from an external environment by making a probabilistic transition to a next state.

Hereafter, we consider a non-empty finite set of *input actions*  $A$ , and a non-empty, finite set of *atomic propositions*  $AP$ .

**Definition 1.** A semi-Markov decision process (SMDP) is given by a tuple  $M = (S, \tau, \rho, L)$  where

- $S$  is a non-empty, countable set of states,
- $\tau: S \times A \rightarrow \mathcal{D}(S)$  is the transition function,
- $\rho: S \rightarrow \mathcal{D}(\mathbb{R}_{\geq 0})$  is the residence-time function, and
- $L: S \rightarrow 2^{AP}$  is the labelling function.

The operational behaviour of an SMDP is as follows. The SMDP at a given state  $s \in S$ , after receiving an input  $a \in A$ , goes to state  $s' \in S$  within time  $t$  with probability  $\tau(s, a)(s') \cdot \rho(s)([0, t])$ . An SMDP is said to be finite if it has a finite set of states. For  $s \in S$ , we will write  $F_s$  for the CDF of  $\rho(s)$ , i.e.  $F_s(t) = \rho(s)([0, t])$ .

Continuous-time Markov decision processes are a special case of SMDPs in which all residence-time functions are exponentially distributed, and discrete-time Markov decision processes are a special case of SMDPs where the residence-time distribution in each state is the Dirac measure at 0, representing the fact that transitions are taken instantaneously.

In defining simulation and bisimulation for SMDPs, we will use ingredients from the definition of simulation and bisimulation for Markov decision processes [21] and simulation and bisimulation for continuous-time Markov chains [3]. However, since we are also generalising to arbitrary distributions on time rather than just exponential distributions, the condition on rates for exponential distributions must be replaced with a more general condition on the distributions. There is a rich literature on so-called stochastic orders [22], which impose an ordering on random variables. We will consider here the most commonly used of these, known as the *usual stochastic order*.

**Definition 2.** For an SMDP  $M = (S, \tau, \rho, L)$ , a relation  $R \subseteq S \times S$  is a simulation (resp. bisimulation) on  $M$  if for all  $(s_1, s_2) \in R$  we have

1.  $L(s_1) = L(s_2)$ ,
2.  $F_{s_2}(t) \geq F_{s_1}(t)$  (resp.  $F_{s_2}(t) = F_{s_1}(t)$ ) for all  $t \in \mathbb{R}_{\geq 0}$ , and
3. for all  $a \in A$  there exists a weight function or coupling  $\Delta_a: S \times S \rightarrow [0, 1]$  between  $\tau(s_1, a)$  and  $\tau(s_2, a)$  such that
  - (a)  $\Delta_a(s, s') > 0$  implies  $(s, s') \in R$ ,
  - (b)  $\tau(s_1, a)(s) = \sum_{s' \in S} \Delta_a(s, s')$  for all  $s \in S$ , and
  - (c)  $\tau(s_2, a)(s') = \sum_{s \in S} \Delta_a(s, s')$  for all  $s' \in S$ .

We say that  $s_2$  simulates (resp. is bisimilar to)  $s_1$ , written  $s_1 \preceq s_2$  (resp.  $s_1 \sim s_2$ ), if there is a simulation (resp. bisimulation) relation containing  $(s_1, s_2)$ .

It is easy to show that the *similarity* relation  $\preceq$  is the largest simulation relation, and analogously that the *bisimilarity* relation  $\sim$  is the largest bisimulation relation. The coupling ensures that the simulation relation is preserved by successor states. Intuitively,  $s_1$  simulates  $s_2$  if the CDF of  $\rho(s_2)$  is pointwise greater than or equal to the CDF of  $\rho(s_1)$ , and the transition probability distribution of  $s_1$  can be matched by the transition probability function  $s_2$  by means of a coupling, in such a way that if two successor states  $s'_1$  and  $s'_2$  have a non-zero coupling, then  $s'_1$  again simulates  $s'_2$ . For bisimulation, we instead require that the CDFs behave exactly the same in each step.

Given a set  $C \subseteq S$  and a relation  $R \subseteq S \times S$ , let

$$R(C) = \{s' \in S \mid (s, s') \in R \text{ for some } s \in C\}$$

be the  $R$ -closure of  $C$ . If  $R$  is a preorder,  $R(C)$  is the upward closure of  $C$ .

The following result, which is a trivial generalisation of [25, Lemma 4.2.4], gives a different but equivalent definition of simulation which is sometimes useful.

**Proposition 1.** For finite  $S$ ,  $R \subseteq S \times S$  is a simulation relation if and only if for any  $(s_1, s_2) \in R$  the first two conditions for simulation are satisfied and

$$\tau(s_1, a)(C) \leq \tau(s_2, a)(R(C)), \text{ for all } C \subseteq S.$$

The following generalises [3, Proposition 25(3)] to the case of SMDPs.

**Proposition 2.**  $\preceq \cap \preceq^{-1} = \sim$ .

The above is analogous to a result stating that bisimulation and simulation equivalence coincide for deterministic labelled transition systems [2]. In our case, Proposition 2 holds because reactive systems are inherently deterministic.

### 3 Comparing the Speed of Residence-Time Distributions

For comparing the random variables describing the residence time on states, the similarity relation uses the usual stochastic order: if  $s_1 \preceq s_2$  then, for all  $t \in \mathbb{R}_{\geq 0}$ ,  $F_{s_1}(t) \leq F_{s_2}(t)$ . In words, if  $s_2$  simulates  $s_1$ , it is more likely that

$s_2$  will take a transition before  $s_1$ , that is,  $s_2$  is stochastically faster than  $s_1$  in reacting to an input.

In this section, we propose a different way of comparing residence-time distributions. The idea is to get quantitative information about how much a distribution should be accelerated to become at least as fast as another distribution.

**Definition 3.** *Let  $F$  and  $G$  be CDFs and  $\varepsilon \in \mathbb{R}_{>0}$ . We say that  $F$  is  $\varepsilon$ -faster than  $G$ , written  $F \sqsubseteq_\varepsilon G$ , if for all  $t$  we have  $F(\varepsilon \cdot t) \geq G(t)$ .*

Consider two states  $s_1$  and  $s_2$ , having residence time governed by the distributions  $F_{s_1}$  and  $F_{s_2}$ , respectively, and assume that  $F_{s_1} \sqsubseteq_\varepsilon F_{s_2}$  holds. If  $0 < \varepsilon \leq 1$ , then this means that  $s_1$  is stochastically faster than  $s_2$ , even if the residence time in  $s_1$  is slowed down by a factor  $\varepsilon$ . If instead we have  $\varepsilon > 1$ , then  $s_1$  is stochastically slower than  $s_2$ , but if we accelerate its residence-time distribution by a factor  $\varepsilon$ , then it becomes stochastically faster than  $s_2$ .

In the rest of the section we will argue that  $\sqsubseteq_\varepsilon$  is a good notion for gathering quantitative information about the speed of residence-time distributions on states. We will do this by comparing the most common distributions used in the literature for modelling residence time on states on stochastic systems: Dirac distributions, exponential distributions, and uniform distributions.

The Dirac measure at zero is the fastest measure, in the following sense.

**Proposition 3.** *Let  $F$  be any CDF. The following holds for any  $\varepsilon \in \mathbb{R}_{>0}$ .*

1. *Dirac[0]  $\sqsubseteq_\varepsilon F$ .*
2. *If  $F \neq \text{Dirac}[0]$ , then  $F \not\sqsubseteq_\varepsilon \text{Dirac}[0]$ .*

For comparing exponential distributions, it is simple to show that it is enough to accelerate by the ratio between the two rates. The same is true for uniform distributions, except we also need to consider whether the uniform distributions start at 0, since if a uniform distribution starts at 0, then we can only hope to make another uniform distribution faster than it if this other uniform distribution also starts at 0.

**Proposition 4**

1. *Exp[ $\theta_1$ ]  $\sqsubseteq_\varepsilon$  Exp[ $\theta_2$ ], where  $\varepsilon = \frac{\theta_2}{\theta_1}$ .*
2. *If  $c = 0$  and  $a > 0$ , then Unif[ $a, b$ ]  $\not\sqsubseteq_\varepsilon$  Unif[ $c, d$ ] for any  $\varepsilon \in \mathbb{R}_{>0}$ .*
3. *If  $c = 0$  and  $a = 0$ , then Unif[ $a, b$ ]  $\sqsubseteq_\varepsilon$  Unif[ $c, d$ ], where  $\varepsilon = \frac{b}{d}$ .*
4. *If  $c > 0$ , then Unif[ $a, b$ ]  $\sqsubseteq_\varepsilon$  Unif[ $c, d$ ], where  $\varepsilon = \max\{\frac{a}{c}, \frac{b}{d}\}$ .*

*In all cases, the given  $\varepsilon$  is the least such that the  $\varepsilon$ -faster than relation holds.*

Moreover, an exponential distribution can never be made faster than a uniform distribution, since uniform distributions become 1 eventually, whereas exponential distributions tend asymptotically to 1 but never reach it. Furthermore, whether or not a uniform distribution can be made faster than an exponential distribution depends on its value at 0.

**Proposition 5**

1.  $Exp[\theta] \not\sqsubseteq_\varepsilon Unif[a, b]$  for all  $\varepsilon \in \mathbb{R}_{>0}$ .
2. If  $a > 0$ , then  $Unif[a, b] \not\sqsubseteq_\varepsilon Exp[\theta]$  for all  $\varepsilon \in \mathbb{R}_{>0}$ .
3. If  $a = 0$ , then  $Unif[a, b] \sqsubseteq_\varepsilon Exp[\theta]$ , where  $\varepsilon = \theta \cdot b$ . Furthermore, this is the least  $\varepsilon$  such that the  $\varepsilon$ -faster-than relation holds.

The  $\varepsilon$ -faster-than relation enjoys a kind of monotonicity property, which is simply a consequence of the fact that CDFs are increasing.

**Lemma 1.** *Let  $\varepsilon \leq \varepsilon'$  and assume that  $F \sqsubseteq_\varepsilon G$ . Then  $F \sqsubseteq_{\varepsilon'} G$ .*

The probability distribution of the sum of two independent random variables is the *convolution* of their individual distributions. The general formula for the convolution of two measures  $\mu$  and  $\nu$  on the real line is given by

$$(\mu * \nu)(E) = \int_0^\infty \nu(E - x) \mu(dx).$$

Notably, the  $\varepsilon$ -faster-than relation is a congruence with respect to convolution of measures.

**Proposition 6.** *If  $F_{\mu_1} \sqsubseteq_\varepsilon F_{\mu_2}$  and  $F_{\nu_1} \sqsubseteq_\varepsilon F_{\nu_2}$ , then  $F_{(\mu_1 * \nu_1)} \sqsubseteq_\varepsilon F_{(\mu_2 * \nu_2)}$ .*

In Sect. 7.1 we will see that the above property is essential for the preservation of reachability properties. Intuitively, this is because convolution corresponds to sequential composition of the residence-time behaviour.

There are other possible ways to compare the relative speed of residence-time distributions quantitatively. In the following we explore some alternative definitions of the notion of the  $\varepsilon$ -faster-than relation, and argue that none of them are preferable to the one given in Definition 3. Given two CDFs  $F$  and  $G$ , we consider the following three alternative definitions of  $F \sqsubseteq_\varepsilon G$ :

1. for all  $t$ ,  $F(t) \cdot \varepsilon \geq G(t)$ ,
2. for all  $t$ ,  $F(t) + \varepsilon \geq G(t)$ , and
3. for all  $t$ ,  $F(\varepsilon + t) \geq G(t)$ .

If  $\sqsubseteq_\varepsilon$  is defined as in (3), then we see that  $Unif[a, b] \not\sqsubseteq_\varepsilon Unif[c, d]$ , for any  $\varepsilon \in \mathbb{R}_{>0}$  whenever  $c < a$ . This is because  $Unif[a, b](a) \cdot \varepsilon = 0 < Unif[c, d](a)$ . Hence we lose the properties of Proposition 4.

If  $\sqsubseteq_\varepsilon$  is defined as in (2), we trivially get that whenever  $\varepsilon \geq 1$ ,  $F \sqsubseteq_\varepsilon G$ , for any two CDFs  $F$  and  $G$ . Hence (2) is only interesting for  $0 \leq \varepsilon < 1$ . However, even in this case we would still lose the properties of Proposition 4. Indeed, whenever  $a \geq d$ ,  $Unif[a, b] \not\sqsubseteq_\varepsilon Unif[c, d]$ , for any  $0 \leq \varepsilon < 1$ . This follows because  $Unif[a, b](a) + \varepsilon = \varepsilon < 1 = Unif[c, d](a)$ .

Lastly, if  $\sqsubseteq_\varepsilon$  is defined as in (1), then it would not be a congruence with respect to convolution of distributions, i.e., Proposition 6 would not hold. For a counterexample, take  $F_{\mu_1} = Unif[2, 4]$ ,  $F_{\mu_2} = Unif[1, 3]$ ,  $F_{\nu_1} = Unif[3, 4]$ , and  $F_{\nu_2} = Unif[2, 4]$ . Then  $F_{\mu_1} \sqsubseteq_1 F_{\mu_2}$  and  $F_{\nu_1} \sqsubseteq_1 F_{\nu_2}$ , but  $F_{(\mu_1 * \mu_2)} \not\sqsubseteq_1 F_{(\nu_1 * \nu_2)}$ .



### 4 A Hemimetric for Semi-Markov Decision Processes

In this section, we are going to extend the definition of simulation relation between SMDPs to the quantitative setting. We will see that this relation naturally induces a notion of distance between SMDPs, describing the least acceleration factor required globally on the residence-time distributions to make a given SMDP as fast as another one.

**Definition 4.** Let  $\varepsilon \in \mathbb{R}_{>0}$ . For an SMDP  $M = (S, \tau, \rho, L)$ , a relation  $R \subseteq S \times S$  is a  $\varepsilon$ -simulation relation on  $M$  if for all  $(s_1, s_2) \in R$  we have that the first and third condition for simulation are satisfied, and  $F_{s_2} \sqsubseteq_\varepsilon F_{s_1}$ . We say that  $s_2$   $\varepsilon$ -simulates  $s_1$ , written  $s_1 \preceq_\varepsilon s_2$ , if there is a  $\varepsilon$ -simulation relation  $R$  such that  $(s_1, s_2) \in R$ .

*Example 1.* Let  $A = \{a\}$  and consider the SMDP  $M = (S, \tau, \rho, L)$  given by  $S = \{s_1, s_2\}$ ,  $\tau(s_1, a)(s_1) = 1 = \tau(s_2, a)(s_2)$ ,  $F_{s_1} = \text{Exp}[4]$ ,  $F_{s_2} = \text{Exp}[2]$ , and  $L(s_1) = L(s_2)$ . By Proposition 4 we see that  $s_1 \preceq_2 s_2$  and  $s_2 \preceq_{\frac{1}{2}} s_1$ .

It is easy to show that the  $\varepsilon$ -similarity relation  $\preceq_\varepsilon$  is the largest simulation relation, and with the previous section in mind, one immediately sees that  $\preceq_1$  coincides with  $\preceq$ . Moreover, the following holds.

**Proposition 7.** For any  $\varepsilon \leq 1$ , if  $s_1 \preceq_\varepsilon s_2$ , then  $s_1 \preceq s_2$ .

If  $\varepsilon > 1$ , the above implication does not hold. For an easy counterexample consider  $s_1$  and  $s_2$  from Example 1 where  $s_1 \preceq_2 s_2$  but  $s_1 \not\preceq s_2$ .

For  $\varepsilon > 1$ , we can obtain a result similar to Proposition 7 only if we “accelerate” the overall behaviour of  $s_2$ . Formally, for a given SMDP  $M = (S, \tau, \rho, L)$ , we define the SMDP  $M_\varepsilon = (S_\varepsilon, \tau_\varepsilon, \rho_\varepsilon, L_\varepsilon)$  as follows:

$$\begin{aligned} S_\varepsilon &= S \cup \{(s)_\varepsilon \mid s \in S\}, & \tau_\varepsilon(s, a)(s') &= \tau(s, a)(s'), \\ L_\varepsilon(s) &= L(s), & \tau_\varepsilon(s, a)((s')_\varepsilon) &= 0, \\ L_\varepsilon((s)_\varepsilon) &= L(s), & \tau_\varepsilon((s)_\varepsilon, a)(s') &= 0, \\ \rho_\varepsilon(s)([0, t]) &= \rho(s)([0, t]), & \tau_\varepsilon((s)_\varepsilon, a)((s')_\varepsilon) &= \tau(s, a)(s'). \\ \rho_\varepsilon((s)_\varepsilon)([0, t]) &= \rho(s)([0, \varepsilon \cdot t]), \end{aligned}$$

Intuitively, the states  $s \in S$  in  $M_\varepsilon$  are identical copies of those in  $M$ , whereas the states  $(s)_\varepsilon$  react to each input  $a \in A$  functionally identically to  $s$  but faster, since the residence-time on the states are all equally accelerated by a factor  $\varepsilon$ , thus  $(s)_\varepsilon \preceq_\varepsilon s$ . For this reason  $(s)_\varepsilon$  is called the  $\varepsilon$ -acceleration of  $s$ .

Given the definition of accelerated state, Proposition 7 can be generalised to arbitrary values of  $\varepsilon \in \mathbb{R}_{>0}$  in the following way.

**Proposition 8.** For any  $\varepsilon \in \mathbb{R}_{>0}$ ,  $s_1 \preceq_\varepsilon s_2$  if and only if  $s_1 \preceq (s_2)_\varepsilon$ .

The relevance of the above statement is twofold: it clarifies the relation between  $\preceq_\varepsilon$  and  $\preceq$ , and also provides a way to modify the behaviour of a state  $s_2$  of an SMDP in order to simulate a state  $s_1$  whenever  $s_1 \preceq_\varepsilon s_2$  holds.

Having this characterisation of similarity in terms of acceleration of processes one can think about the following problem: given two states,  $s_1$  and  $s_2$  such that  $s_1 \not\preceq s_2$ , what is the least  $\varepsilon \geq 1$  (if it exists) such that  $s_1 \preceq (s_2)_\varepsilon$  holds? We can answer this question by means of the following distance.

**Definition 5.** *The simulation distance  $d: S \times S \rightarrow [1, \infty]$  between two states  $s_1$  and  $s_2$  is given by*

$$d(s_1, s_2) = \inf\{\varepsilon \geq 1 \mid s_1 \preceq_\varepsilon s_2\}.$$

As usual, if there is no  $\varepsilon \geq 1$  such that  $s_1 \preceq_\varepsilon s_2$ , then  $d(s_1, s_2) = \infty$ , because  $\inf \emptyset = \infty$ . It is clear from the definition that  $s_1 \preceq s_2$  if and only if  $d(s_1, s_2) = 1$ .

Note that the definition above does not give a distance in the usual sense, for two reasons:  $d$  is not symmetric and it does not satisfy the triangle inequality. One can show instead that  $d$  satisfies a multiplicative version of the triangle inequality, namely, that for all  $s_1, s_2, s_3 \in S$ ,  $d(s_1, s_3) \leq d(s_1, s_2) \cdot d(s_2, s_3)$ . This is a direct consequence of the following properties of  $\preceq_\varepsilon$ . The first property states that  $\preceq_\varepsilon$  is monotonic with respect to increasing values of  $\varepsilon$ .

**Lemma 2.** *If  $s_1 \preceq_\varepsilon s_2$  and  $\varepsilon \leq \varepsilon'$ , then  $s_1 \preceq_{\varepsilon'} s_2$ .*

The second property is a quantitative generalisation of transitivity from which the multiplicative inequality discussed above follows.

**Lemma 3.** *If  $s_1 \preceq_\varepsilon s_2$  and  $s_2 \preceq_{\varepsilon'} s_3$ , then  $s_1 \preceq_{\varepsilon \cdot \varepsilon'} s_3$ .*

Typically, one still uses the term distance for such multiplicative distances, because by applying the logarithm one does obtain a hemimetric.

**Theorem 1.**  *$\log d(s_1, s_2)$  is a hemimetric.*

*Example 2.* Consider again the SMDP from Example 1. We can now see that  $d(s_1, s_2) = 2$  and  $d(s_2, s_1) = \frac{1}{2}$ . This also shows that our distance is not symmetric, and hence not a pseudometric.

## 5 Computing the Simulation Distance

In this section we provide an algorithm to compute the simulation distance given in Definition 5 for finite SMDPs. The algorithm is shown to run in polynomial time for the distributions we have considered so far.

The following technical lemma will provide a sound basis for the correctness of the algorithm. Given two CDFs  $F$  and  $G$ , let

$$c(F, G) = \inf\{\varepsilon \geq 1 \mid F \sqsubseteq_\varepsilon G\}$$

denote the least acceleration factor needed by  $F$  to be faster than  $G$ .

**Lemma 4.** *For an SMDP  $M$ , define the set  $\mathcal{C}(M) = \{c(F_{s'}, F_s) \mid s, s' \in S\}$ . If  $d(s_1, s_2) \neq \infty$ , then*

- $s_1 \preceq_c s_2$ , for some  $c \in \mathcal{C}(M)$  and
- $d(s_1, s_2) = \min\{c \in \mathcal{C}(M) \mid s_1 \preceq_c s_2\}$ .

Lemma 4 provides a strategy for computing the simulation distance between any two states  $s_1$  and  $s_2$  of a given SMDP  $M$  as follows. First, one constructs the set  $\mathcal{C}(M)$ . If  $s_1 \preceq_c s_2$  does not hold for any  $c \in \mathcal{C}(M)$ , then the distance must be infinite; otherwise, it is the smallest  $c \in \mathcal{C}(M)$  for which  $s_1 \preceq_c s_2$  holds.

In order for this strategy to work, we need two ingredients: first, we should be able to compute the set  $\mathcal{C}(M)$  and second, for any  $c \in \mathcal{C}(M)$ , we need an algorithm for checking whether  $s_1 \preceq_c s_2$ .

Recall that SMDPs allow for *arbitrary* residence-time distributions on states. Therefore, it is not guaranteed that for any SMDP  $M$  the set  $\mathcal{C}(M)$  can be computed. With the following definition we identify the class of SMDPs for which this can be done.

**Definition 6.** *A class  $\mathcal{C}$  of CDFs is effective if for any  $F, G \in \mathcal{C}$ ,  $c(F, G)$  is computable. An SMDP  $M$  is effective if  $\{F_s \mid s \in S\}$  is an effective class.*

In particular, for any pair of states  $s, s'$  of an effective SMDP  $M$ , we can decide whether  $F_{s'} \sqsubseteq_\varepsilon F_s$  by simply checking whether  $\varepsilon \geq c(F_{s'}, F_s)$ . We will denote by  $f(l)$  the complexity of computing  $c(F_{s'}, F_s)$  for two arbitrary  $s, s' \in S$  as a function of the length  $l$  of the representation of the residence-time distributions of  $M$ .

Let  $\mathcal{C}_A$  denote the class consisting of the Dirac distribution at 0 as well as uniform and exponential distributions with rational parameters. By Propositions 3–5 we immediately see that  $\mathcal{C}_A$  is an effective class, and in fact it can be computed using only simple operations such as multiplication, division, and taking maximum. Hence  $f(l)$  has constant complexity<sup>1</sup> whenever  $M$  takes residence-time distributions from  $\mathcal{C}_A$ .

Next we consider how to decide  $s_1 \preceq_\varepsilon s_2$  for a given rational  $\varepsilon \geq 1$ . A decision procedure can be obtained by adapting to our setting the algorithm by Baier et al. [2] for deciding the simulation preorder between probabilistic labelled transition systems. The algorithm from [2] uses a partition refinement approach to compute the largest simulation relation and runs in time  $\mathcal{O}(mn^7/\log n)$  for reactive systems, where  $m = |A|$  is the number of actions, and  $n = |S|$  is the number of states. Given  $\varepsilon \geq 1$ , we can proceed correspondingly to compute  $\varepsilon$ -similarity: we start from the relation  $R = S \times S$  and update it by removing all the pairs  $(s, s')$  of states not satisfying the conditions of Definition 4. This process is repeated on the resulting updated relation until no more pairs of states are removed. The resulting relation is the largest  $\varepsilon$ -simulation. Hence, checking  $s_1 \preceq_\varepsilon s_2$  corresponds to determining whether the pair  $(s_1, s_2)$  is contained in the relation returned by the above algorithm.

**Theorem 2.** *Let  $M$  be a finite and effective SMDP. Given  $s_1, s_2 \in S$  and  $\varepsilon \geq 1$ , deciding whether  $s_1 \preceq_\varepsilon s_2$  can be done in time  $\mathcal{O}(n^2(f(l) + k) + (mn^7)/\log n)$ , where  $k = |AP|$  is the number of atomic propositions.*

<sup>1</sup> As is standard, we consider numbers to be represented as floating points of bounded size in their binary representation.

```

1 Order the elements of  $\mathcal{C}(M) \setminus \{\infty\}$  such that  $c_1 < c_2 < \dots < c_n$ ;
2 if  $s_1 \preceq_{c_1} s_2$  then return  $c_1$  ;
3 else if  $s_1 \not\preceq_{c_n} s_2$  then return  $\infty$  ;
4 else
5    $i \leftarrow 1, j \leftarrow n$ ;
6   while  $i < j$  do
7      $h \leftarrow \lceil \frac{j-i}{2} \rceil$ ;
8     if  $s_1 \preceq_{c_{j-h}} s_2$  then  $j \leftarrow j - h$  ;
9     else  $i \leftarrow i + h$  ;
10  end
11  return  $c_j$ ;
12 end

```

**Algorithm 1:** Computing the simulation distance between  $s_1$  and  $s_2$ .

The algorithm for computing the simulation distance is given in Algorithm 1. The algorithm starts by ordering the elements of  $\mathcal{C}(M)$  as  $c_1, \dots, c_n$  while removing  $\infty$  from the list. Then it searches for the smallest  $c_i$  such that  $s_1 \preceq_{c_i} s_2$  holds. This is done by means of a bisection method. If  $s_1 \preceq_{c_1} s_2$  holds, then  $c_1$  is the smallest element such that this holds, so we return it. If  $s_1 \preceq_{c_n} s_2$  does not hold, then, by Lemma 2,  $s_1 \preceq_{c_i} s_2$  does not hold for any  $1 \leq i \leq n$ , so we return  $\infty$ . If none of the above apply, at this point of the algorithm (line 4) we have that  $s_1 \not\preceq_{c_1} s_2$  and  $s_1 \preceq_{c_n} s_2$ .

We use the variables  $i$  and  $j$ , respectively, as the left and right endpoints of the bisection interval. The bisection interval keeps track of those elements  $c_n$  for which we still do not know whether  $s_1 \preceq_{c_n} s_2$ . At the beginning,  $i = 1$  and  $j = n$ . At line 7,  $h = \lceil \frac{j-i}{2} \rceil$  is used as the decrement factor for the length of the bisection interval at each step. Since  $h > 0$ , the bisection interval decreases in size for each iteration. If  $s_1 \preceq_{c_{j-h}} s_2$  holds, then  $j - h$  is the current smallest element in  $\mathcal{C}(M)$  for which this holds, hence  $j - h$  will become the new right endpoint of the interval; otherwise  $i + h$  is the new left endpoint. The bisection method stops when the endpoints meet or cross each other, at which point we know that  $s_1 \not\preceq_{c_n} s_2$  for all  $n < j$  and  $s_1 \preceq_{c_n} s_2$  for all  $n \geq j$ , and hence we return  $c_j$ .

Computing the set  $\mathcal{C}(M)$  at line 1 has complexity  $n^2 f(l)$ , and sorting it can be done in time  $\mathcal{O}(n \log n)$  using mergesort. By Theorem 2, and since we have already computed  $\mathcal{C}(M)$ , each of the  $\varepsilon$ -simulation checks in lines 2, 3, and 8 can be done in time  $\mathcal{O}(n^2 k + (mn^7)/\log n)$ , but the complexity  $n^2 k$  from comparing labels only needs to be computed once. Since the bisection interval is halved each time, the while-loop is taken at most  $\log n$  times. We therefore get an overall time complexity of  $\mathcal{O}(n^2(f(l) + k) + mn^7)$ .

**Theorem 3.** *Let  $M$  be a finite and effective SMDP. The simulation distance between any two states can be computed in time  $\mathcal{O}(n^2(f(l) + k) + mn^7)$ .*

## 6 Compositional Properties of the Simulation Distance

In this section we will prove that some natural notions of parallel composition on SMDPs are non-expansive with respect to the simulation distance.

First we define what it means to compose two SMDPs in parallel. As argued in [23], the style of synchronous CSP is the one that is most suitable for SMDPs, so this is the one we will adopt here.

**Definition 7.** *A function  $\star : \mathcal{D}(\mathbb{R}_{\geq 0}) \times \mathcal{D}(\mathbb{R}_{\geq 0}) \rightarrow \mathcal{D}(\mathbb{R}_{\geq 0})$  is a residence-time composition function if it is commutative.*

**Definition 8.** *Let  $\star$  be a residence-time composition function. Then the  $\star$ -composition of  $M_1 = (S_1, \tau_1, \rho_1, L_1)$  and  $M_2 = (S_2, \tau_2, \rho_2, L_2)$ , denoted  $M_1 \parallel_\star M_2 = (S, \tau, \rho, L)$ , is given as follows, for arbitrary  $s_1, s'_1 \in S_1$ ,  $s_2, s'_2 \in S_2$ , and  $a \in A$ .*

1.  $S = S_1 \times S_2$ ,
2.  $\tau((s_1, s_2), a)((s'_1, s'_2)) = \tau_1(s_1, a)(s'_1) \cdot \tau_2(s_2, a)(s'_2)$ ,
3.  $\rho((s_1, s_2)) = \star(\rho_1(s_1), \rho_2(s_2))$ , and
4.  $L(s_1 \parallel_\star s_2) = L(s_1) \cup L(s_2)$ .

Given a composite system  $M_1 \parallel_\star M_2 = (S, \tau, \rho, L)$ , we write  $s_1 \parallel_\star s_2$  to mean  $(s_1, s_2) \in S$ . The residence-time composition function  $\star$  allows us to accommodate many different ways of combining timing behaviour, including those found in the literature on process algebras. We recall here some of these.

**Maximum composition:**  $F_{\star(\mu, \nu)}(t) = \max(F_\mu(t), F_\nu(t))$ .

For exponential distributions,  $F_\mu = \text{Exp}[\theta]$  and  $F_\nu = \text{Exp}[\theta']$ , the following alternatives can be found.

**Product rate composition:**  $F_{\star(\mu, \nu)} = \text{Exp}[\theta \cdot \theta']$ .

**Minimum rate composition:**  $F_{\star(\mu, \nu)} = \text{Exp}[\min\{\theta, \theta'\}]$ .

**Maximum rate composition:**  $F_{\star(\mu, \nu)} = \text{Exp}[\max\{\theta, \theta'\}]$ .

Maximum composition is used for interactive Markov chains [11], product rate composition is used in SPA [12], minimum rate composition is used in PEPA [13], and maximum rate composition is used in TIPP [10].

In order to have non-expansiveness for  $\star$ -composition of SMDPs, we will need to restrict to residence-time composition functions  $\star$  that are monotonic.

**Definition 9.** *A residence-time composition function  $\star$  is monotonic if for all  $\varepsilon \geq 1$  and  $\mu, \nu, \eta \in \mathcal{D}(\mathbb{R}_{\geq 0})$ , it holds that  $F_\mu \sqsubseteq_\varepsilon F_\nu$  implies  $F_{\star(\mu, \eta)} \sqsubseteq_\varepsilon F_{\star(\nu, \eta)}$ .*

Requiring monotonicity is not a significant restriction, as many of the composition functions that are found in the literature are indeed monotonic.

**Lemma 5.** *Maximum composition as well as product, minimum, and maximum rate composition are all monotonic.*

Now we can prove that the  $\star$ -composition of finite SMDPs is non-expansive with respect to the simulation distance, provided that  $\star$  is monotonic.

**Theorem 4.** *For finite SMDPs and monotonic  $\star$ ,*

$$d(s_1, s_2) \leq \varepsilon \quad \text{implies} \quad d(s_1 \parallel_\star s_3, s_2 \parallel_\star s_3) \leq \varepsilon.$$

We conclude this section by exploring the computational aspects of composition of SMDPs. In particular, we would like to be able to also compute the distance between composite systems.

By Lemma 4, we know that computing the simulation distance amounts to being able to compute the constants  $c(F_s, F_{s'})$ , for each pair of states  $s, s'$  of the SMDP. Hence we would like that, whenever two distributions  $\mu$  and  $\nu$  have effective CDFs then also their composition  $\star(\mu, \nu)$  has an effective CDF. By Proposition 4, it is easy to see that this holds for product, minimum, and maximum rate composition, since these compositions are still exponential distributions.

For maximum composition, the class  $\mathcal{C}_\Delta$  is unfortunately not closed under composition. However, the following result holds.

**Proposition 9.** *Let  $\star$  be maximum composition. For any  $\mu, \nu, \eta \in \mathcal{C}_\Delta$ , the constants  $c(F_\mu, F_{\star(\nu, \eta)})$  and  $c(F_{\star(\mu, \eta)}, F_\nu)$  are computable.*

The above results tells us that if we are interested in computing the distance  $d(s_1, s_2 \parallel_\star s_3)$  or  $d(s_1 \parallel_\star s_2, s_3)$ , when  $\star$  is maximum composition, then we can indeed compute the constants  $c$  that are needed for Algorithm 1 to work.

## 7 Logical Properties of the Simulation Distance

If the distance between two processes is small, then we would also expect that they satisfy almost the same properties. In order to make this idea precise, in this section we introduce and study a slight extension of Markovian logic [15], which we will call *timed Markovian logic* (TML). The syntax of TML is given by the following grammar, where  $\alpha \in AP$ ,  $p \in \mathbb{Q}_{\geq 0} \cap [0, 1]$ ,  $t \in \mathbb{Q}_{\geq 0}$ , and  $a \in A$ .

$$\text{TML} : \quad \varphi ::= \alpha \mid \neg\alpha \mid \ell_p t \mid m_p t \mid L_p^a \varphi \mid M_p^a \varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi'$$

The semantics of TML is given by

$$\begin{array}{llll} s \models \alpha & \text{iff } \alpha \in L(s) & s \models \ell_p t & \text{iff } F_s(t) \geq p \\ s \models \neg\alpha & \text{iff } \alpha \notin L(s) & s \models m_p t & \text{iff } F_s(t) \leq p \\ s \models \varphi \wedge \varphi' & \text{iff } s \models \varphi \text{ and } s \models \varphi' & s \models L_p^a \varphi & \text{iff } \tau(s, a)(\llbracket \varphi \rrbracket) \geq p \\ s \models \varphi \vee \varphi' & \text{iff } s \models \varphi \text{ or } s \models \varphi' & s \models M_p^a \varphi & \text{iff } \tau(s, a)(\llbracket \varphi \rrbracket) \leq p \end{array}$$

where  $\llbracket \varphi \rrbracket = \{s \in S \mid s \models \varphi\}$  is the set of states satisfying  $\varphi$ .

We also isolate the following two fragments of TML.

$$\text{TML}^{\geq} : \quad \varphi ::= \alpha \mid \neg\alpha \mid \ell_p t \mid L_p^a \varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi'$$

$$\text{TML}^{\leq} : \quad \varphi ::= \alpha \mid \neg\alpha \mid m_p t \mid M_p^a \varphi \mid \varphi \wedge \varphi' \mid \varphi \vee \varphi'$$

Intuitively, the formula  $L_p^a \varphi$  says that the probability of taking an  $a$ -transition to where  $\varphi$  holds is *at least*  $p$ , and  $M_p^a \varphi$  says the probability is *at most*  $p$ .  $\ell_p t$  and  $m_p t$  are similar in spirit, but talk about the probability of firing a transition instead. Thus,  $\ell_p t$  says that the probability of firing a transition before time  $t$  is *at least*  $p$ , whereas  $m_p t$  says that the probability is *at most*  $p$ .

For any  $\varphi \in \text{TML}$  and  $\varepsilon \geq 1$  we denote the  $\varepsilon$ -perturbation of  $\varphi$  by  $(\varphi)_\varepsilon$  and define it inductively as

$$\begin{aligned} (\alpha)_\varepsilon &= \alpha & (\ell_p t)_\varepsilon &= \ell_p \varepsilon \cdot t & (L_p^a \varphi)_\varepsilon &= L_p^a(\varphi)_\varepsilon & (\varphi \wedge \varphi')_\varepsilon &= (\varphi)_\varepsilon \wedge (\varphi')_\varepsilon \\ (\neg\alpha)_\varepsilon &= \neg\alpha & (m_p t)_\varepsilon &= m_p \varepsilon \cdot t & (M_p^a \varphi)_\varepsilon &= M_p^a(\varphi)_\varepsilon & (\varphi \vee \varphi')_\varepsilon &= (\varphi)_\varepsilon \vee (\varphi')_\varepsilon. \end{aligned}$$

By making use of the alternative characterisation for simulation given in Proposition 1 and drawing upon ideas from [6], we can now prove the following logical characterisation of the  $\varepsilon$ -simulation relation.

**Theorem 5.** *Let  $\varepsilon \in \mathbb{Q}_{\geq 0}$  with  $\varepsilon \geq 1$ . For any finite SMDP, the following holds.*

- $s_1 \preceq_\varepsilon s_2$  if and only if  $\forall \varphi \in \text{TML}^{\geq}. s_1 \models \varphi \implies s_2 \models (\varphi)_\varepsilon$ .
- $s_1 \preceq_\varepsilon s_2$  if and only if  $\forall \varphi \in \text{TML}^{\leq}. s_2 \models \varphi \implies s_1 \models (\varphi)_\varepsilon$ .

As a special case of Theorem 5, we have also shown that  $\text{TML}^{\geq}$  and  $\text{TML}^{\leq}$  characterise simulation for SMDPs. Conceptually, Theorem 5 says that if  $s_1$   $\varepsilon$ -simulates  $s_2$ , then  $s_2$  satisfies the  $\varepsilon$ -perturbation of any property that  $s_2$  satisfies for the  $\text{TML}^{\geq}$  fragment of TML, and vice versa for the  $\text{TML}^{\leq}$  fragment.

By Lemma 4 and Theorem 5, we get the following corollary, connecting our simulation distance with the properties expressible in the logic TML.

**Corollary 1.** *Let  $\varepsilon \in \mathbb{Q}_{\geq 0}$  with  $\varepsilon \geq 1$ . For finite SMDPs the following holds.*

- $d(s_1, s_2) \leq \varepsilon$  if and only if  $\forall \varphi \in \text{TML}^{\geq}. s_1 \models \varphi \implies s_2 \models (\varphi)_\varepsilon$ .
- $d(s_1, s_2) \leq \varepsilon$  if and only if  $\forall \varphi \in \text{TML}^{\leq}. s_2 \models \varphi \implies s_1 \models (\varphi)_\varepsilon$ .

By Proposition 2, we also get a logical characterisation of bisimulation for SMDPs in terms of TML, which is simpler than the one given in [17, 24].

**Theorem 6.** *For any finite SMDP, it holds that*

$$s_1 \sim s_2 \quad \text{if and only if} \quad \forall \varphi \in \text{TML}. s_1 \models \varphi \iff s_2 \models \varphi.$$

## 7.1 Reachability Properties

We will now argue that the simulation distance behaves nicely also with respect to linear-time properties, by proving preservation of reachability properties up to perturbations.

The probability of reaching a given set of states in an SMDP depends on the choice of actions in each state. The non-determinism introduced by this choice is typically resolved by means of *schedulers*. Here we consider probabilistic

schedulers  $\sigma$  of type  $S^* \rightarrow \mathcal{D}(A)$ , telling us what the probability is of selecting an action  $a \in A$  depending on the history of the states visited so far. Given a scheduler  $\sigma$ , we denote by  $\mathbb{P}_s^\sigma(\diamond^t X)$  the probability, under the scheduler  $\sigma$ , that starting from the state  $s$  the SMDP will eventually reach a state in  $X \subseteq S$  within time  $t \geq 0$  (for a rigorous definition of this probability see e.g. [17]).

Given our notion of  $\varepsilon$ -simulation, we can prove the following result.

**Theorem 7.** *Let  $\beta$  be a Boolean combination of atomic propositions. If we have  $s_1 \preceq_\varepsilon s_2$ , then for any scheduler  $\sigma$  there exists a scheduler  $\sigma'$  such that*

$$\mathbb{P}_{s_1}^\sigma(\diamond^t \llbracket \beta \rrbracket) \leq \mathbb{P}_{s_2}^{\sigma'}(\diamond^{\varepsilon \cdot t} \llbracket \beta \rrbracket) \quad (\text{or equivalently, } \mathbb{P}_{s_1}^\sigma(\neg \diamond^t \llbracket \beta \rrbracket) \geq \mathbb{P}_{s_2}^{\sigma'}(\neg \diamond^{\varepsilon \cdot t} \llbracket \beta \rrbracket)).$$

Note that the above result might find useful applications for speeding up the computation time required by model checking tools to disprove certain types of reachability properties. For example, consider the atomic proposition `bad`, identifying all the states considered “not safe” in the SMDP. Usually, given a process  $s$ , one wants to verify that, under all possible schedulers  $\sigma$ , the probability  $\mathbb{P}_s^\sigma(\neg \diamond^t \llbracket \text{bad} \rrbracket)$  is above a certain threshold value  $\delta \leq 1$ , meaning that the SMDP is unlikely to end up in an unsafe configuration within a time horizon bounded by  $t$ . Then, to disprove this property one only needs to provide a scheduler  $\sigma'$  and a process  $s'$  such that  $s' \preceq_\varepsilon s$  and  $\mathbb{P}_{s'}^{\sigma'}(\neg \diamond^{\frac{t}{\varepsilon}} \llbracket \text{bad} \rrbracket) < \delta$ . Indeed, given that  $s' \preceq_\varepsilon s$ , by Theorem 7

$$\mathbb{P}_{s'}^{\sigma'}(\neg \diamond^{\frac{t}{\varepsilon}} \llbracket \text{bad} \rrbracket) < \delta \xrightarrow{Th.7} \exists \sigma. \mathbb{P}_s^\sigma(\neg \diamond^t \llbracket \text{bad} \rrbracket) < \delta.$$

Since  $s$  simulates  $s'$ ,  $s'$  can be thought of as a simplified abstraction of  $s$ , which is usually a smaller process. Hence, finding a scheduler  $\sigma'$  for  $s'$  which gives a counterexample may be much simpler than finding one for  $s$ . Moreover, the above technique is robust to perturbations of  $\varepsilon$ .

## 8 Conclusions and Open Problems

We have proposed a quantitative extension of the notion of simulation relation on SMDPs, called  $\varepsilon$ -simulation, comparing the relative speed of different processes. This quantitative notion of simulation relation induces a multiplicative hemimetric, which we call simulation distance, measuring the least acceleration factor needed by a process to speed up its actions in order to behave at least as fast as another process.

We have given an efficient algorithm to compute the simulation distance and identified a class of distributions for which the algorithm works on finite SMDPs. Furthermore, we have shown that, under mild conditions on the composition of residence-time distributions on states, a generalised version of CSP-like parallel composition on SMDPs is non-expansive with respect to this distance, showing that our distance is suitable for compositional reasoning. Lastly, we have shown the connection between our distance and properties expressible in a timed extension of Markovian logic. Namely, we have shown that if the simulation distance



between  $s_1$  and  $s_2$  is at most  $\varepsilon$ , then  $s_1$  satisfies the  $\varepsilon$ -perturbation of any property that  $s_2$  satisfies. This result also gives a novel logical characterisation of simulation and bisimulation for semi-Markov decision processes.

Instead of using the usual stochastic order to relate the timing behaviour of states as we have done, one could also consider many other kinds of stochastic orders, for example ones that compare the expected value of the distributions. This may be more natural for applications where one wants to consider an exponential distribution with a high enough rate to be faster than a uniform distribution.

We have shown that the timing distributions that are obtained when composing systems are compatible with the algorithm for computing the distance only in the case when composing systems either on the left or on the right. A more general result showing that this also happens when composing on both sides an arbitrary number of components seems difficult. Nonetheless, we are confident that such a result can be obtained for any concrete case involving common types of distributions used in the literature.

Since we have both a distance and a logical characterisation, it makes sense to ask whether the set of states satisfying a formula is a closed or an open set in the topology induced by the distance. If such a set is indeed closed, this means that the approximate reasoning at the limit is sound: if every state in a sequence satisfies a formula, then the limit of that sequence also satisfies the formula. Such an investigation has already been done for Markovian logic [16], and some of the ideas from there may carry over to our setting.

**Acknowledgments.** We thank the anonymous reviewers for helpful suggestions as well as Robert Furber and Giovanni Bacci for insightful discussions. This research was supported by the Danish FTP project ASAP, the ERC Advanced Grant LASSO, and the Sino-Danish Basic Research Center IDEA4CPS.

## References

1. Alvim, M.S., Chatzikokolakis, K., McIver, A., Morgan, C., Palamidessi, C., Smith, G.: Additive and multiplicative notions of leakage, and their capacities. In: CSF, pp. 308–322 (2014)
2. Baier, C., Engelen, B., Majster-Cederbaum, M.E.: Deciding bisimilarity and similarity for probabilistic processes. *J. Comput. Syst. Sci.* **60**(1), 187–231 (2000)
3. Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. *Inf. Comput.* **200**(2), 149–214 (2005)
4. Chatzikokolakis, K., Andrés, M.E., Bordenabe, N.E., Palamidessi, C.: Broadening the scope of differential privacy using metrics. In: De Cristofaro, E., Wright, M. (eds.) PETS 2013. LNCS, vol. 7981, pp. 82–102. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-39077-7\\_5](https://doi.org/10.1007/978-3-642-39077-7_5)
5. Chen, D., van Breugel, F., Worrell, J.: On the complexity of computing probabilistic bisimilarity. In: Birkedal, L. (ed.) FoSSaCS 2012. LNCS, vol. 7213, pp. 437–451. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-28729-9\\_29](https://doi.org/10.1007/978-3-642-28729-9_29)
6. Desharnais, J.: Logical characterization of simulation for labelled Markov chains. In: PROBMIV, pp. 33–48. University of Birmingham, Technical report, CS-99-8, August 1999

7. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labelled Markov processes. *Theor. Comput. Sci.* **318**(3), 323–354 (2004)
8. Ferns, N., Panangaden, P., Precup, D.: Bisimulation metrics for continuous Markov decision processes. *SIAM J. Comput.* **40**(6), 1662–1714 (2011)
9. Giacalone, A., Jou, C.-C., Smolka, S.A.: Algebraic reasoning for probabilistic concurrent systems. In: *Proceedings of the IFIP TC2 Working Conference on Programming Concepts and Methods*, North-Holland, pp. 443–458 (1990)
10. Götz, N., Herzog, U., Rettelbach, M.: Multiprocessor and distributed system design: the integration of functional specification and performance analysis using Stochastic Process Algebras. In: Donatiello, L., Nelson, R. (eds.) *Performance/SIGMETRICS - 1993*. LNCS, vol. 729, pp. 121–146. Springer, Heidelberg (1993). <https://doi.org/10.1007/BFb0013851>
11. Hermanns, H.: *Interactive Markov Chains: The Quest for Quantified Quality*. LNCS, vol. 2428. Springer, Heidelberg (2002). <https://doi.org/10.1007/3-540-45804-2>
12. Hermanns, H., Herzog, U., Mertsiotakis, V.: Stochastic process algebras - between LOTOS and Markov chains. *Comput. Netw.* **30**(9–10), 901–924 (1998)
13. Hillston, J.: *A Compositional Approach to Performance Modelling*. Distinguished Dissertations in Computer Science. Cambridge University Press, New York (2005)
14. Jou, C.-C., Smolka, S.A.: Equivalences, congruences, and complete axiomatizations for probabilistic processes. In: Baeten, J.C.M., Klop, J.W. (eds.) *CONCUR 1990*. LNCS, vol. 458, pp. 367–383. Springer, Heidelberg (1990). <https://doi.org/10.1007/BFb0039071>
15. Kozen, D., Mardare, R., Panangaden, P.: Strong completeness for Markovian logics. In: Chatterjee, K., Sgall, J. (eds.) *MFCS 2013*. LNCS, vol. 8087, pp. 655–666. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40313-2\\_58](https://doi.org/10.1007/978-3-642-40313-2_58)
16. Larsen, K.G., Mardare, R., Panangaden, P.: Taking it to the limit: approximate reasoning for Markov processes. In: Rovan, B., Sassone, V., Widmayer, P. (eds.) *MFCS 2012*. LNCS, vol. 7464, pp. 681–692. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-32589-2\\_59](https://doi.org/10.1007/978-3-642-32589-2_59)
17. Neuhäüßer, M.R., Katoen, J.-P.: Bisimulation and logical preservation for continuous-time Markov decision processes. In: Caires, L., Vasconcelos, V.T. (eds.) *CONCUR 2007*. LNCS, vol. 4703, pp. 412–427. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-74407-8\\_28](https://doi.org/10.1007/978-3-540-74407-8_28)
18. Pedersen, M.R., Bacci, G., Larsen, K.G., Mardare, R.: A hemimetric extension of simulation for semi-Markov decision processes. Technical report, Aalborg University, Department of Computer Science (2018). <http://people.cs.aau.dk/mrp/pubs/simuldist.pdf>
19. Perman, M., Senegacnik, A., Tuma, M.: Semi-Markov models with an application to power-plant reliability analysis. *IEEE Trans. Reliab.* **46**(4), 526–532 (1997)
20. Pievatolo, A., Tironi, E., Valade, I.: Semi-Markov processes for power system reliability assessment with application to uninterruptible power supply. *IEEE Trans. Power Syst.* **19**(3), 1326–1333 (2004)
21. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. *Nord. J. Comput.* **2**(2), 250–273 (1995)
22. Shaked, M., Shanthikumar, G.: *Stochastic Orders*. Springer Series in Statistics. Springer, New York (2007). <https://doi.org/10.1007/978-0-387-34675-5>

23. Sokolova, A., de Vink, E.P.: Probabilistic automata: system types, parallel composition and comparison. In: Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P., Siegle, M. (eds.) *Validation of Stochastic Systems: A Guide to Current Research*. LNCS, vol. 2925, pp. 1–43. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-24611-4\\_1](https://doi.org/10.1007/978-3-540-24611-4_1)
24. Song, L., Zhang, L., Godskesen, J.C.: Bisimulations and logical characterizations on continuous-time Markov decision processes. In: McMillan, K.L., Rival, X. (eds.) *VMCAI 2014*. LNCS, vol. 8318, pp. 98–117. Springer, Heidelberg (2014). [https://doi.org/10.1007/978-3-642-54013-4\\_6](https://doi.org/10.1007/978-3-642-54013-4_6)
25. Zhang, L.: *Decision algorithms for probabilistic simulations*. Ph.D. thesis, Saarland University, Saarbrücken, Germany (2009)



# Policy Synthesis for Collective Dynamics

Paul Piho<sup>(✉)</sup> and Jane Hillston

University of Edinburgh, Edinburgh, UK  
paul.piho@ed.ac.uk

**Abstract.** In this paper we consider the problem of policy synthesis for systems of large numbers of simple interacting agents where dynamics of the system change through information spread via broadcast communication. By modifying the existing modelling language CARMA and giving it a semantics in terms of continuous time Markov decision processes we introduce a natural way of formulating policy synthesis problems for such systems. However, solving policy synthesis problems is difficult since all non-trivial models result in very large state spaces. To combat this we propose an approach exploiting the results on fluid approximations of continuous time Markov chains to obtain estimates of optimal policies.

## 1 Introduction

The study of collective dynamics has a wealth of interesting applications in collective adaptive systems (CAS) where examples range from swarming behaviour of insects to patterns of epidemic spread in humans. Such systems are highly distributed and robust in nature and for that reason are an interesting paradigm for the design of highly-distributed computer-based systems.

The study of such systems concentrates on the emergent behaviour arising from simple behaviour and communication rules at the level of individuals. However, due to CAS exhibiting non-linear dynamics it is difficult to verify or predict the emergent behaviour. It is harder still to know a priori how to design the behaviour and capabilities of individual agents in order to achieve a system level goal. Moreover, the individual agents in such systems are often unreliable and prone to failure making it natural to express objectives at the collective level in terms of a proportion of the population achieving a goal within a time bound.

We seek to develop a framework in which the flexibility afforded by having a homogeneous population of agents can be leveraged in planning decisions and explore how communication can alter the likelihood of satisfying the collective goals. The basis of the framework is a process algebra supporting formal expression of basic behaviour of population members, including patterns of communication in the collective. Based on the formal specification the policy synthesis problems are framed in terms of continuous time Markov decision processes (CT-MDP) which give a versatile framework for a variety of stochastic control and planning problems.

Policy synthesis for CT-MDP models is a computationally complex problem in general. Consequently, there have been a number of recent works on policy

synthesis and closely related model checking of CT-MDPs e.g., [1,2]. Our problem domain, the collective dynamics of homogeneous agents, offers a way to approximate system dynamics through fluid approximation describing the state of the system in terms of continuous variables. In many cases this alleviates the problem of state space explosion. We study leveraging fluid approximation results, inspired by similar results for discrete time Markov decision processes [3], in the context of policy synthesis for collective dynamics. We note that broadcast communication is a natural way to model information spread in collective systems. However, this makes it impossible to directly apply the existing results like [4]. Thus, we propose an approximations for a class of systems involving broadcast communication.

The paper makes the following contributions. Firstly, we propose a class of systems arising from collective systems involving broadcast communication. Secondly, we present a process algebra based on the existing language CARMA [5] which together with specification of goals from [6] provides a high level framework for formulating policy synthesis problems. Next, we propose an efficient policy synthesis approach, exploiting the structure suggested by the language level description, to find configurations that can satisfy the defined collective goals. We concentrate on applications of fluid approximation results in cases involving broadcast communication where standard results do not apply. Finally, we frame a simplified foraging example, inspired by the design of a robot swarm, in the presented framework and consider the policy synthesis.

The paper is structured as follows: in Sect. 2 we detail the formal specification of policy synthesis for collective dynamics. Particularly, in Sect. 2.1 we introduces a class of systems arising from broadcast communication in collectives being treated as a switch in the dynamics. The Sect. 2.2 outlines a process algebraic language with its semantics for specifying models in the CT-MDP framework. In Sect. 3 we discuss ideas arising from fluid approximations and adapting them for policy synthesis for collective dynamics. In particular, we suggest an approximation, based on, fluid results for dealing with broadcast communication. In Sect. 4 we present a simplified case-study inspired by swarm robotics to motivate the developed framework. Finally, we end with concluding remarks and discussion of further work in Sect. 5.

## 2 System Specification

**Carma Process Algebra.** CARMA [5] is a stochastic process algebra designed to support specification and analysis of CAS. A CARMA system consists of a *collective* operating in an *environment*. The collective describes a set of interacting *agents* and models the behaviour of a system. The description of an agent consist of a *process*, that describes the agent's behaviour, and of a *store*, that models its *knowledge*. Here *knowledge* is limited to the values of key attributes.

The processes described within components interact via a rich set of communication primitives. In particular, the language supports both broadcast and unicast communication. Both are attribute-based so that a component can only

receive a message when its store satisfies the sender’s target predicate and receivers use a predicate to identify acceptable message sources.

The environment is responsible for setting the rates and probabilities governing action execution and message exchange. Thus the environment models all aspects intrinsic to the context the system operates in, e.g., the rate at which a component moves may depend on the terrain at the given location.

The formal semantics give rise to a continuous time Markov chain (CTMC) – the state space of the system is represented as a finite, discrete set of states and the times of transitions are governed by the rates given in the model description where each rate is taken to be the parameter of an exponential distribution.

**Continuous-Time Markov Decision Processes.** Our target mathematical object for policy synthesis is a continuous-time Markov decision process (CT-MDPs) rather than a CTMC. CT-MDPs are a common framework in stochastic control theory and operations research providing a natural formalisation of policy optimisation problems. Here we give relevant standard definitions for CT-MDPs.

**Definition 1.** *Continuous-time Markov decision process is defined by the tuple  $\{\mathcal{S}, \mathcal{A}, \{\mathcal{A}(i), i \in \mathcal{S}\}, q(j \mid i, a)\}$  where  $\mathcal{S}$  is the countable set of states,  $\mathcal{A}$  is the Borel measurable set of actions,  $\{\mathcal{A}(i), i \in \mathcal{S}\}$  is the set of feasible actions in state  $i$  and  $q(j \mid i, a)$  gives the transition rates  $i \rightarrow j$  given the control action  $a$ .*

The evolution of CT-MDPs is described by the following: after the process reaches some state and an action is chosen the process performs a transition to the next state depending only on the current state and the chosen action. The time it takes for state transitions to happen is governed by exponential distributions with rates given by the function  $q$  in Definition 1. The actions at every such step are chosen according to some policy as defined below.

**Definition 2.** *A policy is a measurable function  $\pi : \mathbb{R}_{\geq 0} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$  which for every time  $t \in \mathbb{R}_{\geq 0}$ , state  $s \in \mathcal{S}$  and action  $a \in \mathcal{A}(s)$  assigns a probability  $\pi(t, s, a)$  that the action  $a$  is chosen in  $s$  at time  $t$ . We call a policy where for every  $t \in \mathbb{R}_{\geq 0}$  and  $s \in \mathcal{S}$  we have that  $\pi(t, s, a) \in \{0, 1\}$  a deterministic policy. A policy  $\pi$  independent of  $t$  is a stationary policy.*

For the purpose of leveraging fluid approximations we introduce the concept of population CT-MDPs which result from models where components are only distinguished through their state. Thus the state of the system is represented as a vector of counting variables detailing the number of components in each state.

**Definition 3.** *A population CT-MDP is a tuple  $(\mathbf{X}, \mathcal{T}, \mathcal{A}, \beta)$  defined by:  $\mathbf{X} = (X_1, \dots, X_n) \in \mathcal{S} = \mathbb{Z}_{\geq 0}^n$  where each  $X_i$  takes values in a finite domain  $\mathcal{D}_i \subset \mathbb{Z}_{\geq 0}$ ;  $\beta$  is a function such that  $\beta(a, \mathbf{X})$  returns a boolean value indicating whether action  $a \in \mathcal{A}$  is available from state  $\mathbf{X}$ ;  $\mathcal{T}$  is a set of transitions of the form  $\tau = (a, \mathbf{v}_\tau, r_\tau(\mathbf{X}))$  such that  $\beta(a, \mathbf{X}) = 1$ ,  $\mathbf{v}_\tau$  is an update vector specifying that the state after execution of transition  $\tau$  is  $\mathbf{s} + \mathbf{v}_\tau$  and  $r_\tau(\mathbf{X})$  is a rate function.*

A population CT-MDP is associated with a CT-MDP in the following way: the state and action space of the corresponding CT-MDP is the same as for the population CT-MDP; the set of feasible actions for state  $\mathbf{i} \in \mathcal{S}$ , denoted  $\mathcal{A}(\mathbf{i})$ , is defined by  $\mathcal{A}(\mathbf{i}) = \{a \in \mathcal{A} \mid \beta(a, \mathbf{i}) = 1\}$ ; the rate function  $q$  is defined as

$$q(\mathbf{i} \mid \mathbf{j}, a) = \sum \{r_\tau(\mathbf{j}) \in \mathcal{T} \mid \tau = (a, \mathbf{v}_\tau, r_\tau(\mathbf{j})) \wedge \mathbf{i} = \mathbf{j} + \mathbf{v}_\tau\}$$

## 2.1 Broadcast Communication

Broadcast is a natural communication pattern to consider in CAS. However, in general it makes it impossible to apply the fluid approximation results which informally rely on the effect of actions being bounded as more components are introduced to the system. This limits the usefulness of fluid approximation methods when analysing collective systems.

We propose a class of population systems for which the problems arising from broadcast communication can be mitigated. In particular, we consider cases where broadcast can be thought of as a switch between dynamic modes of the population. More generally, we consider population processes in the CT-MDP framework with the mode switching dynamics as described in the following. Let  $\mathcal{M} = (\mathbf{X}, \mathcal{T}, \mathcal{A}, \beta)$  be a population CT-MDP with  $\mathbf{X} = (X_1, \dots, X_{2n})$ . The model  $\mathcal{M}$  exhibits mode switching dynamics if, up to reordering of variables, there exist  $\mathbf{X}_1 = (X_1, \dots, X_n)$  and  $\mathbf{X}_2 = (X_{n+1}, \dots, X_{2n})$  ( $\mathbf{X}$  is the concatenation of  $\mathbf{X}_1$  and  $\mathbf{X}_2$ ) such that

- $\mathbf{X}_1 \neq \mathbf{0}$  if and only if  $\mathbf{X}_2 = \mathbf{0}$  – the system can be in one mode or the other.
- there is a transition  $(a, \mathbf{v}_\tau, r_\tau(\mathbf{X})) \in \mathcal{T}$  with an update vector  $(-s_1, \dots, -s_n, s_1, \dots, s_n)$  that happens with a non-zero rate from state  $(s_1, \dots, s_n, 0, \dots, 0)$ . This ensures there exists at least one transition between the modes.
- There is no state  $(0, \dots, 0, s_{n+1}, \dots, s_n)$  for which there exists a transition with an update vector  $(s_{n+1}, \dots, s_n, -s_{n+1}, \dots, -s_n)$  happening at a non-zero rate – we consider models where the mode switching is unidirectional.
- $(\mathbf{X}_1, \mathcal{T}, \mathcal{A}, \beta)$  and  $(\mathbf{X}_2, \mathcal{T}, \mathcal{A}, \beta)$  define a population CT-MDP.

Although the definition is given for two dynamic modes we can easily extend it. As mentioned, such population models can arise from considering broadcast communication. In particular, we study situations where broadcast is used to propagate knowledge acquired by a component in the system to the rest of the components leading to a change in component behaviour. Such knowledge, once acquired, is not lost or forgotten leading to unidirectional mode changes. In the following section we introduce a process algebra based language for constructing such models and set up our running example of mode switching.

## 2.2 Language and Configuration

We propose a process algebra based language, CARMA-C, which uses a subset of syntactic constructs of CARMA to simplify the creation of the described

population models. We focus on models of collectives involving broadcast communication and thus retain the constructs of CARMA for broadcast communication, knowledge stores and the environment. For specification of policy synthesis problems, we introduce non-determinism in transitions interpreted as possible control actions. The control actions are encoded in terms of attributes in knowledge stores where the values of such attributes are left partially specified—instead of particular values we define the value domains for the attributes.

As unicast and attribute based communication are not the focus of this paper we do not carry over the syntactic constructs for these from CARMA. However, note that the material in this section can easily be extended to specify systems making use of unicast and attribute based communication.

### 2.3 Syntax

As in CARMA, we say a system consists of a collective  $N$  operating in an environment  $\mathcal{E}$ . We let  $\text{SYS}$  be the set of systems  $S$ , and  $\text{COL}$  be the set of collectives  $N$  where a collective is either a component  $C$  in the set of components  $\text{COMP}$  or the parallel composition of collectives. A component  $C$  can either be the inactive component, denoted  $\mathbf{0}$ , or a term of the form  $(P, \gamma)$  where  $P$  is a process and  $\gamma$  is a store. In particular, systems, collectives and components are generated by the following grammar:

$$S ::= N \mathbf{in} \mathcal{E} \quad N ::= C \mid N_1 \parallel N_2 \quad C ::= \mathbf{0} \mid (P, \gamma)$$

The grammar for the processes is given by the following:

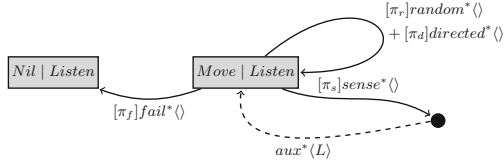
$$\begin{aligned} P, Q &::= \mathbf{nil} \mid \mathit{act}.P \mid P + Q \mid P \mid Q \mid [\pi]P \mid A (A \triangleq P) \\ \mathit{act} &::= \alpha^* \langle \vec{e} \rangle \sigma \mid \alpha^* (\vec{e}) \sigma \end{aligned}$$

The processes are defined using standard constructs—action prefix, choice, and parallel composition—from process algebras literature. In addition we allow the definition of the inactive process  $\mathbf{nil}$  and guards on processes. The action primitives are defined for broadcast output in the form  $\alpha^* \langle \vec{e} \rangle \sigma$  and broadcast input in the form  $\alpha^* (\vec{e}) \sigma$ . The broadcast output action is defined as non-blocking and an output action with no corresponding input is interpreted as a spontaneous action of a component.

The following notation is used:  $\alpha$  is an action type used to distinguish between different actions;  $\vec{e}$  is an expression specifying the message sent over broadcast communication;  $\pi$  is a boolean expression such that a process guarded by  $\pi$  is only activated when  $\pi$  evaluates to true;  $\sigma$  is an update specifying a probability distribution over the possible store configurations following the given action.

*Example 1.* For the running example we consider a scenario where robots need to locate a source by sensing their local environment and move to the location of the source. Components in the system correspond to the robots with the following behaviour: robots can move on a grid, take measurements from their





**Fig. 1.** Behaviour of individual *Robot* components.

location and broadcast this information to the rest of the swarm. The model we use to describe the behaviour of the robots is illustrated in Fig. 1. The action *random\** corresponds to the robot exploring the environment through a random walk while *directed\** corresponds to moving towards a found source location. The action *sense\** models the robot taking measurements of its locale. If a source is detected then the auxiliary action *aux\** immediately broadcasts the set of source locations  $L$ . The action *fail\**, resulting in the robot not performing further actions, models failure. Finally, the process *Listen* receives the corresponding broadcast input action  $aux^*(L)$ .

## 2.4 Semantics

The novelty of relating a CARMA-C model to a CT-MDP lies in defining the set of admissible controls via the stores. In particular, instead of fully specifying store variables, as done in CARMA, we allow them to take values in a general Borel measurable set defined in the model. The set of feasible actions (as in Definition 1) then corresponds to possible refinements of stores to particular values.

**Store.** In CARMA a store is a function that maps attribute names to particular values which are then used in the semantics for the transition rate calculations. In CARMA-C, we instead define the store as a function that maps attribute names to permitted value domains. That is, a store  $\gamma$  maps a set of attribute names  $a_0, \dots, a_n$  in its domain to the value domains of the attributes. This introduces non-determinism in the choice of particular store values. Such non-determinism for system specification has previously been considered in the case of interval Markov chains (IMC) [7], constraint Markov chains (CMC) [8] and probabilistic constraint Markov chains (PCMC) [9]. The non-determinism in these cases is treated as arising from a transition probability or a rate for which the true value is not known or that the probability can take any value in the given region. In our approach the non-determinism will be resolved by a policy maker.

*Example 2.* For the running example we define the local store of each robot consisting of attributes *location* giving the location of the robot, and *source* holding the set of locations identified as source. Figure 2 illustrates the effects of actions on the local stores. In particular, *random\** and *directed\** change the location of

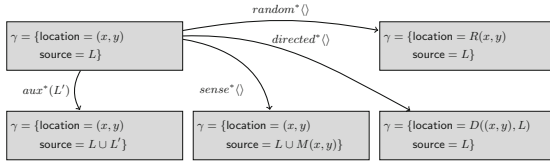


Fig. 2. Local component store changes induced by actions.

the robot. The former picks with uniform probability a target location reachable from location  $(x, y)$  – the corresponding update is denoted by  $R(x, y)$ . The latter picks a location that takes the robot closer to the source  $L$  – update denoted by  $D((x, y), L)$ . If the robot’s location corresponds to a source location the action  $sense^*$  includes the location in the set of sources with some probability thus modelling the possibility of false negatives resulting from noisy measurements. In particular, we define  $M$  such that if the location  $(x, y)$  is a source then  $M$  returns  $(x, y)$  with probability  $p$  and the empty set with probability  $1 - p$ . The input action  $aux^*(L')$  adds the locations  $L'$  in the message to the set of sources.

We use guards to achieve the desired behaviours of components. Specifically, the guard  $\pi_r$  for  $random^*$  is true when the attribute `source` corresponds to the empty set – source location has not yet been found. For simplicity, we also allow the action  $sense^*$  only if the source has not been discovered. Conversely, the guard for  $directed^*$  is true when the attribute `source` defines a non-empty set of locations. The guard  $\pi_f$  for  $fail^*$  evaluates to true everywhere except when the robot is at the source location.

**Environment.** Like in CARMA, the environment in CARMA-C models aspects intrinsic to the context where the agents under consideration are operating – it sets the rates of actions and mediates the interactions between components.

For a system  $S \in \text{SYS}$  we say that the environment  $\mathcal{E}$  is defined by the global store  $\gamma_g$  and an evolution rule  $\rho$ . The evolution rule  $\rho$  is a function which given a global store  $\gamma_g$  and the current state of the collective  $N \in \text{COL}$  returns a tuple of functions  $\varepsilon = \langle \mu_p, \mu_r, \mu_u \rangle$  called the evaluation context. The functions given by the evaluation context are interpreted as follows:  $\mu_p$  expresses the probability of receiving a broadcast;  $\mu_r$  specifies the execution rates of actions;  $\mu_u$  determines the updates on the environment store.

*Example 3.* For the running example we define the following environment: the global store  $\gamma_g$  defines a store attribute `failr`  $\in [0, 1]$ ; the probability of receiving the broadcast for the  $aux^*$  action is set to 1; the constant rate of spontaneous actions  $random^*$  and  $directed^*$  is given by  $r_m$  and the rate of  $sense^*$  is given by  $r_s$ ; the action  $aux^*$  emulates an instantaneous action with its rate set very high; the rate  $r_f$  of  $fail^*$  is set equal to the store attribute `failr` introducing non-determinism in the behaviour of the system; we set  $\mu_u$  such that the global store remains unchanged through the evolution.

**Resolving Non-determinism.** The semantics for the construction of a CT-MDP model from a syntactic description of a CARMA-C model is done in two stages. The first part of the semantics resolves the non-determinism in the model. In particular, we consider a system  $S$  defined by  $(P_1, \gamma_1) \mid \cdots \mid (P_n, \gamma_n)$  **in**  $(\gamma_g, \rho)$ .

The set of control actions  $\mathcal{A}(S)$  available from  $S$  is defined by the following: let  $\mathcal{A}(S)$  be a set of functions such that for all  $\gamma \in \{\gamma_1, \dots, \gamma_n, \gamma_g\}$ ,  $f \in \mathcal{A}(S)$  maps all attributes  $a$  in the domain of  $\gamma$  to particular value in  $\gamma(a)$ . That is, a set of feasible control actions from a system  $S$  corresponds to the set of possible functions that fix the store values. In the formal semantics we introduce a refinement step labelled by a chosen control action  $f$  that transforms  $S$  into

$$S_f \stackrel{\text{def}}{=} (P_1, f(\gamma_1)) \mid \cdots \mid (P_n, f(\gamma_n)) \text{ in } (f(\gamma_g), \rho)$$

Let us define the sets  $\text{SYS}^f$ ,  $\text{COL}^f$  and  $\text{COMP}^f$  as sets of systems, collectives and components after application of  $f$ . We assume that elements of  $\text{SYS}^f$ ,  $\text{COL}^f$  and  $\text{COMP}^f$  are derived only from elements in  $\text{SYS}$ ,  $\text{COL}$  and  $\text{COMP}$  for which  $f$  is sufficient to fully resolve the non-determinism in the behaviour. We call such sets *resolved systems*, *collectives* and *components*, respectively.

*Example 4.* For the running example the set of control actions corresponds to the possible assignments of `failr`. Lower values of `failr` correspond to lower failure rate and thus more robust components.

**Interleaving Semantics.** The second stage of the semantics determines the rates at which a system changes state given a control action. This is achieved through construction of functions  $\mathbb{C}_f$ ,  $\mathbb{N}_{\varepsilon, f}$  and  $\mathbb{S}_f$  parametrised by a chosen control action  $f$ . In particular, the function  $\mathbb{C}_f$  takes a resolved component in  $\text{COMP}^f$  and an action label and returns a probability distribution over components in  $\text{COMP}$ . Components assigned a non-zero probability are reachable from the resolved component. The function,  $\mathbb{N}_{\varepsilon, f}$  builds on  $\mathbb{C}_f$  to describe the behaviour of collectives. Based on a resolved collective in  $\text{COL}^f$  and an action label and it returns a probability distribution over  $\text{COL}$ . As before non-zero probabilities are assigned to reachable collectives. Finally, function  $\mathbb{S}_f$  takes a resolved system in  $\text{SYS}^f$  and an action label and returns a function over systems  $\text{SYS}$  that specifies the rate at which the transitions happen. As for CARMA, these function are constructed via FuTS-style [10] operational semantics. The semantic rules for the second step resulting in the transition rates between systems closely match the semantics given for CARMA in [5] and are not detailed here.

**Population Model.** The population CT-MDP model  $\mathcal{M} = (\mathbf{X}, \mathcal{T}, \mathcal{A}, \beta)$  for a system  $S \in \text{SYS}$  can be derived iteratively based on the assumption that components with the same configuration (same process state and store) are indistinguishable. We start with  $S$  consisting of a collective  $C_1 \parallel \cdots \parallel C_N$  operating in an environment  $\mathcal{E}$ . The function  $\mathbb{C}_f$  can be used to determine all possible future configurations of each of the components  $C_i$ . If the union of all possible

component configurations is finite we can define the finite state space  $\mathcal{S}$  of  $\mathcal{M}$  as the space of counting vectors specifying all possible future configurations of  $S$ .

For each state  $\mathbf{s} \in \mathcal{S}$  we have a set  $Sys_{\mathbf{s}} \subset \text{SYS}$  of corresponding CARMA-C systems. For each system  $S \in Sys_{\mathbf{s}}$  the set of feasible actions will be the same by construction. For the derivation of the population model we add a restriction that any control action acts in the same way on the set of indistinguishable components. The rates corresponding to chosen actions and the reachable states are found using the function  $\mathbb{S}_f$ . In particular, given a control action  $f$  denote the system  $S$  resolved by  $f$  by  $S_f$ . The rate of transition from  $\mathbf{s} \in \mathcal{S}$  to  $\mathbf{s}' \in \mathcal{S}$  given control action  $f$  is then given by

$$\sum_{S \in Sys_{\mathbf{s}}} \sum_{S' \in Sys_{\mathbf{s}'}} \sum_{\ell \in LAB_S} \mathbb{S}_f[S_f, \ell](S')$$

### 3 Policy Synthesis

The main contribution in this paper is a method leveraging fluid approximation results for CTMCs to policy synthesis in the context of collective dynamics involving broadcast communication. Here we state the relevant optimisation problem and discuss how fluid approximation results can be exploited.

**Fluid Approximations of CTMCs.** The aim of fluid approximation of CTMCs, as introduced by Kurtz in [4], is to derive a set of ordinary differential equations (ODEs) for which the sample paths of the CTMC lie, with high probability, close to the solution of the chosen set of ODEs.

Consider a system of  $N$  components each evolving in a finite state space  $SS = \{1, \dots, K\}$  and where components are only distinguishable through their state. Let the state of the object  $n$  at time  $t$  be denoted by  $Y_n^{(N)}(t)$ . Let the variable  $\mathbf{X}^{(N)}(t) \in \mathbb{R}^K$  be a counting vector giving the state of the system at time  $t$ . In particular, the  $i$ -th entry of  $\mathbf{X}^{(N)}(t)$  is given by  $X_i^{(N)}(t) = \sum_n \mathbb{1}\{Y_n^{(N)}(t) = i\}$ .

Next consider the set of transitions, denoted  $\mathcal{T}^{(N)}$ , consisting of elements  $\tau = (R_{\tau}, r_{\tau}^{(N)})$  where  $R_{\tau}$  is a multi-set of update rules of the form  $i \rightarrow j$  specifying that an agent in state  $i$  goes to state  $j$  if the transition  $\tau$  fires. The  $r_{\tau}^{(N)}$  denotes a rate function  $r_{\tau}^{(N)} : \mathbb{R}^K \rightarrow \mathbb{R}_{\geq 0}$  depending on the state of the system. We assume that  $R_{\tau}$  is independent of the population size  $N$ —all transitions involve a finite and fixed number of individuals. The update vector  $\mathbf{v}_{\tau}$  is constructed from  $R_{\tau}$  so that the transition  $\tau$  changes the state of  $\mathbf{X}^{(N)}$  to  $\mathbf{X}^{(N)} + \mathbf{v}_{\tau}$ . We define the Markov population model by a tuple  $\mathcal{X}^{(N)} = (\mathbf{X}^{(N)}, \mathcal{T}^{(N)}, \mathbf{X}_0^{(N)})$  where  $\mathbf{X}_0^{(N)}$  denotes the initial state of the system. Given  $\mathcal{X}^{(N)}$  it is trivial to construct the underlying CTMC  $\mathbf{X}^{(N)}(t)$  describing the time-evolution of the model.

The fluid approximation is achieved by first considering the normalised population counts obtained by dividing each variable by the total population  $N$ — $\hat{\mathbf{X}}^{(N)} = \frac{\mathbf{X}^{(N)}}{N}$ . The initial conditions are scaled similarly— $\hat{\mathbf{X}}_0^{(N)} = \frac{\mathbf{X}_0^{(N)}}{N}$ . The transitions are scaled as follows: for each  $(R_{\tau}, r_{\tau}^{(N)}) \in \mathcal{T}^{(N)}$ , let  $\hat{\mathbf{r}}_{\tau}^{(N)}(\hat{\mathbf{X}})$  be the

rate function expressed in terms of normalised variables. The corresponding transition in the normalised model is  $(R_\tau, \hat{\mathbf{r}}_\tau^{(N)}(\hat{\mathbf{X}}))$  with update vector  $\frac{1}{N} \mathbf{v}_\tau$ . Suppose that for all transitions  $\tau \in \mathcal{T}^{(N)}$  there exists a bounded and Lipschitz continuous function  $f_\tau : \mathbb{R}^K \rightarrow \mathbb{R}_{\geq 0}$  such that  $\frac{1}{N} \hat{\mathbf{r}}_\tau^{(N)}(\hat{\mathbf{X}}) \rightarrow f_\tau(\hat{\mathbf{X}})$  uniformly as  $N \rightarrow \infty$ . To define the limit ODEs, we introduce the drift  $\mathbf{F}(\hat{\mathbf{X}}) = \sum_{\tau \in \hat{\mathcal{T}}^{(N)}} \mathbf{v}_\tau f_\tau(\hat{\mathbf{X}})$ .

**Theorem 1 (Deterministic approximation theorem [4]).** *With  $\hat{\mathbf{X}}^{(N)}(t)$  we assume there exists a point  $\mathbf{x}_0$  such that  $\hat{\mathbf{X}}^{(N)}(0) \rightarrow \mathbf{x}_0$  in probability. Let  $\mathbf{x}(t)$  be a solution to  $\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x})$  with  $\mathbf{x}(0) = \mathbf{x}_0$ . Then, for any finite time horizon  $0 \leq t \leq T$ ,  $\epsilon \in \mathbb{R}_{\geq 0}$  we have*

$$\mathbb{P} \left( \sup_{0 \leq t \leq T} \|\hat{\mathbf{X}}^N(t) - \mathbf{x}(t)\| \geq \epsilon \right) \xrightarrow{N \rightarrow \infty} 0$$

**Policy Optimisation.** Consider a population model  $\mathcal{M}^N = (\mathbf{X}^{(N)}, \mathcal{T}, \mathcal{A}, \beta)$  derived from a CARMA-C model with  $N$  components. We can easily extend the normalisation of CTMC described previously to consider the corresponding normalised population CT-MDP denoted  $\hat{\mathcal{M}}^N$ . We deal with the following problem: *find a policy  $\pi$  in the space of stationary deterministic policies of  $\hat{\mathcal{M}}^N$  that maximises some reward function over a finite time horizon.* In particular, consider the functional  $Q^N : \Pi \rightarrow \mathbb{R}$ , where  $\Pi$  is the set of stationary deterministic policies. The optimisation problem is thus defined as maximising some defined functional  $Q^N$ , i.e., finding a policy  $\pi^*$  that satisfies

$$Q^N[\pi^*] = \sup_{\pi \in \Pi} Q^N[\pi]$$

Suppose we fix a policy  $\pi$  and consider the resulting normalised population CTMC model denoted  $\hat{\mathcal{X}}_\pi^{(N)}$ . As before, let  $\hat{\mathbf{X}}_\pi^{(N)}(t)$  denote the stochastic process describing the time-evolution of the population CTMC. Let  $V : \mathcal{D}_\mathcal{S} \rightarrow \mathbb{R}$  be a reward function on the space of trajectories of the stochastic process  $\hat{\mathbf{X}}_\pi^{(N)}(t)$ . Corresponding reward functional on the space of policies is then given by

$$Q^N[\pi] \stackrel{\text{def}}{=} V(\hat{\mathbf{X}}_\pi^{(N)}(t))$$

*Example 5.* Take a state reward function  $r : \mathcal{S} \rightarrow \mathbb{R}_{\geq 0}$  mapping the states of the CT-MDP to positive real values. Define a value function corresponding to reward function  $r$  and policy  $\pi$  as the expected finite time-horizon ( $0 \leq t \leq T$ ) cumulated reward:

$$Q^N[\pi] \stackrel{\text{def}}{=} V(\hat{\mathbf{X}}_\pi^{(N)}(t)) \stackrel{\text{def}}{=} \mathbb{E} \int_0^T r(\hat{\mathbf{X}}_\pi^{(N)}(t)) dt$$

### 3.1 Policy Synthesis via Fluid Approximation

Evaluating a given functional  $Q$  usually reduces to considering the transient evolution or steady state of  $\hat{\mathcal{X}}_\pi^{(N)}$  which, especially for large population sizes  $N$ ,

is computationally expensive. One way to alleviate this problem is to consider Monte Carlo estimates of the functionals based on simulated trajectories of  $\hat{\mathbf{X}}_\pi^{(N)}$ , e.g., using the Gillespie algorithm. This is done in the context of statistical model checking [11] and has recently been applied to learning effective time-dependent policies for CT-MDPs [12]. Here, we argue that in the case of some reward functionals a good estimate can be achieved via fluid approximation. Indeed, suppose that  $\hat{\mathbf{X}}_\pi^{(N)}$  converges to  $\mathbf{x}_\pi$  in the sense of Theorem 1. Then as a simple consequence of the Portmanteau lemma [13] we can say that for any bounded and continuous reward function  $V$  we get

$$Q^N[\pi] = \mathbb{E}(V(\hat{\mathbf{X}}_\pi^N(t))) \xrightarrow{N \rightarrow \infty} \mathbb{E}(V(\mathbf{x}_\pi(t))) = q[\pi]$$

*Example 6.* For the running example consider the system of  $N$  robots and the global goal: *at least 80% of the robots reach the source location in time  $T$ .* We translate this goal into a value function by considering a logistic function  $I(x) = 1/(1 + e^{-2k(x-0.8)})$  which for large  $k$  approximates a step function. We define a reward function corresponding to the goal by  $V(\hat{\mathbf{X}}_\pi^{(N)}(t)) = I(X_{\pi,s}^{(N)}(T))$  where  $X_{\pi,s}^{(N)}(t)$  denotes the evolution of the population at the source location. Thus if  $\mathbf{x}_\pi$  approximates  $\hat{\mathbf{X}}_\pi^{(N)}$ , in the sense of Theorem 1, then as  $I$  is both continuous and bounded then  $\mathbb{E}(V(\mathbf{x}_\pi(t)))$  approximates  $\mathbb{E}(V(\hat{\mathbf{X}}_\pi^{(N)}(t)))$ .

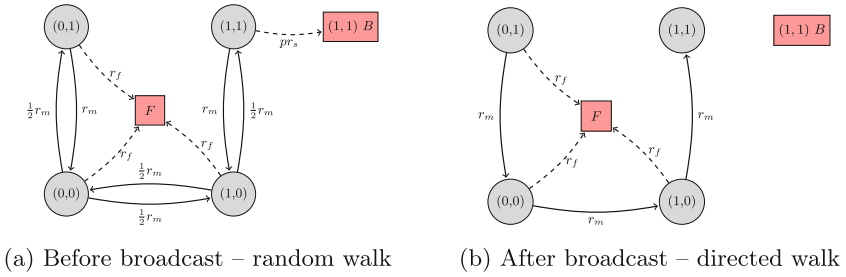
### 3.2 Approximation for Mode Switching

In this section we present a method for approximating the behaviour of population systems exhibiting switching behaviour described in Sect. 2.1. Again the discussion here concentrates on systems with two such dynamic modes where mode changes are unidirectional but the idea can be extended to more modes. The method we propose is based on the observation that the behaviour of the system within a single dynamic mode can be given a fluid approximation as described in Sect. 3. Note that this has similarities to hybrid limit behaviour of Markov population processes considered in [14]. However, for mode switches arising from broadcast communication the rate of switching can depend on the population size which restricts the applicability of results provided in [14].

In detail, consider a normalised population model  $\hat{\mathcal{M}}^N$  with two modes described by  $(\mathbf{X}_1, \mathcal{T}, \mathcal{A}, \beta)$  and  $(\mathbf{X}_2, \mathcal{T}, \mathcal{A}, \beta)$ . Fix a policy  $\pi$  of  $\hat{\mathcal{M}}^N$  and consider the resulting stochastic processes  $\hat{\mathbf{X}}_{\pi,1}^{(N)}$  and  $\hat{\mathbf{X}}_{\pi,2}^{(N)}$  corresponding to the two modes for which we can give a fluid approximation. Denote the resulting approximations by  $\mathbf{x}_{\pi,1}$  and  $\mathbf{x}_{\pi,2}$ . The difficulty now is related to combining the two approximations into an approximation for the mean behaviour of the full process. We propose the following method: identify a variable and a threshold in the fluid approximation of the first mode which serves as an indicator for the mode switch – when the variable reaches the given threshold we expect the mode switch to take place; use this to estimate the switching time  $t^*$ ; approximate the behaviour of the full process by  $\mathbf{x}_\pi(t) = \mathbb{1}\{t \leq t^*\}\mathbf{x}_{\pi,1} + \mathbb{1}\{t > t^*\}\mathbf{x}_{\pi,2}$ . In the following section we evaluate the accuracy of such approximation for the running example and use it to obtain estimates for policy synthesis tasks.

## 4 Analysis: Running Example

In this section we make use of the presented ideas to analyse the running example of a foraging robot swarm. In particular, we consider the system consisting of  $N$  robots and restrict the robots to a  $2 \times 2$  grid with paths  $(0,0) \leftrightarrow (1,0)$ ,  $(0,0) \leftrightarrow (0,1)$  and  $(1,0) \leftrightarrow (1,1)$  to keep the constructions manageable by hand. All robots start by following the behaviour illustrated in Fig. 3a where the location  $(1,1)$  is designated as the source location. At location  $(1,1)$ , with a probability  $p = 0.1$ , the sense action results in a broadcast. A robot sending out such broadcast at location  $(1,1)$  causes the rest of the collective to follow the behaviour given in Fig. 3b giving rise to two dynamic modes for the system.



**Fig. 3.** Behaviour of individual robots.

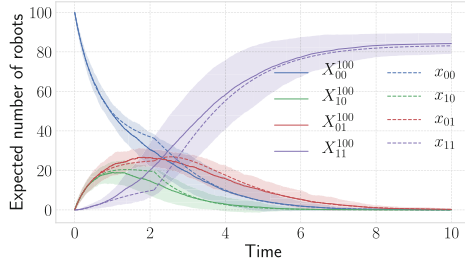
We construct an approximation for the system dynamics by considering variables  $x_{00}$ ,  $x_{10}$ ,  $x_{01}$ ,  $x_{11}$  giving the proportion of robots in locations  $(0,0)$ ,  $(1,0)$ ,  $(0,1)$  and  $(1,1)$  respectively. Additionally, let  $s_{11}$  be the proportion of robots that have sensed the source resulting in a broadcast being sent out and let  $f$  denote the proportion of robots that have broken down. We construct  $\mathbf{x}_1$  and  $\mathbf{x}_2$  as below and claim that these give a fluid approximation for the system dynamics before and after the broadcast respectively.

$$\mathbf{x}_1(t) = \mathbf{x}_2(t) = [x_{00} \ x_{01} \ x_{10} \ x_{11} \ s_{11} \ f]$$

$$\begin{aligned} \frac{d\mathbf{x}_1}{dt} = & [-x_{00}(r_m + r_f) + r_m(x_{10} + \frac{1}{2}x_{01}) \ \frac{1}{2}r_mx_{00} - x_{01}(r_m + r_f) \\ & \frac{1}{2}r_mx_{00} + r_mx_{11} - x_{10}(r_m + r_f) \ \frac{1}{2}r_mx_{10} - x_{11}(p_s r_s - r_m) \\ & p r_s x_{11} \ r_f(x_{00} + x_{10} + x_{01})] \end{aligned}$$

$$\begin{aligned} \frac{d\mathbf{x}_2}{dt} = & [-x_{00}(r_m + r_f) + r_mx_{01} - x_{01}(r_m + r_f) \ r_mx_{00} - x_{10}(r_m + r_f) \\ & x_{01}r_m \ 0 \ r_f(x_{00} + x_{10} + x_{01})] \end{aligned}$$

Suppose all robots start from the location  $(0,0)$ , i.e. the initial condition for the approximation is  $\mathbf{x}_1(0) = [1 \ 0 \ 0 \ 0 \ 0 \ 0]$ . To combine the two modes we use



**Fig. 4.** Comparison of expected trajectories for  $\text{failr} = 0.05 = r_f$  with rate of move and sense actions set to 1 (mean and variance of 100 simulated trajectories: solid lines and fluid approximations: dashed).

the estimate proposed in Sect. 3.2 by considering the time evolution of variable  $s_{11}$  and taking the expected time till the first broadcast to be the time  $t^*$  such that  $s_{11}(t^*) = 1$ . Thus, we approximate the mean behaviour of the system by

$$\hat{\mathbf{x}}(t) = \mathbb{1}\{t \leq t^*\} \mathbf{x}_1(t) + \mathbb{1}\{t > t^*\} \mathbf{x}_2(t)$$

where  $\mathbf{x}_2$  is such that  $\mathbf{x}_2(t^*) = \mathbf{x}_1(t^*)$ . The trajectories are compared with the stochastic simulation in Fig. 4 and Table 1 for different parameters giving an empirical justification for the approximation. Figure 4 suggests that a good approximation is achieved for times away from the mode switching. For Table 1, we consider the mean of relative errors between the stochastic and approximate trajectories for location (1, 1) at time points  $t = 2.0, 4.0, 6.0, 8.0, 10.0$ . For each parametrisation, the table gives a mean figure over 10 comparisons and shows that as expected the approximation is better for larger population sizes.

### 4.1 Policy Synthesis

In the context of the running example we consider the synthesis of  $\text{failr}$  parameter as a special case of policy synthesis. In particular, how robust should the behaviour of the robots be for the collective to satisfy its goal. We provide a simple application for the presented fluid approximation ideas by studying the action (or parameter) space of the running example through logistic regression and via direct optimisation of a more complex reward functional.

**Logistic Regression.** We consider the logistic reward function defined in Example 6 and note that the reward function is defined so that the specified goal (at least 80% of the collective reaches (1, 1)) is satisfied for rewards greater or equal to 0.5. As a first example we consider the following question: *what is the region of failr values for which we are expecting the policy to be satisfied.* We treat  $\text{failr}$  as an indication of robustness of individual components and classify the different possible values based on goal satisfaction. Throughout the rest of this section we set  $r_m = r_s = 1$ .



**Table 1.** Mean approximation error.

$(r_m, r_s, \text{failr})$	Pop. size	Mean error
(1, 1, 0.05)	100	10.8%
(0.8, 1, 0.01)	100	11%
(2.0, 1, 0.1)	100	4.3%
(1, 1, 0.05)	500	1.6%
(1, 1, 0.02)	500	2.0%
(2.0, 1, 0.1)	100	0.6%

**Table 2.** Logistic regression. Fitting based on 100 trajectories.

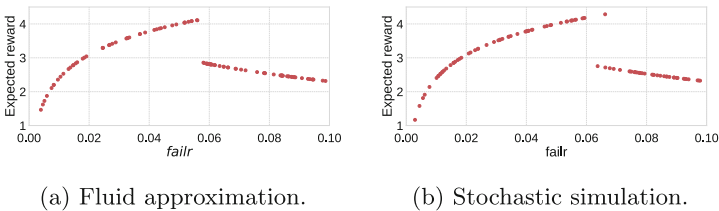
	Decision boundaries
Fluid	0.0566, 0.0553
	0.0579, 0.0560
Stochastic	0.0594, 0.0624
	0.0616, 0.0610

The set-up for this is standard: consider a linear function  $y = w_0 + w_1 r$  of single explanatory variable (in this case value of `failr`, denoted  $r$ ) and a logistic function  $\sigma(r) = 1/(1 + e^{-w_0 - w_1 r})$  where  $\sigma(r)$  is interpreted as the probability of success given `failr` value  $r$ . We are going to expect the goal to be satisfied if  $\sigma(r) > 0.5$ . The weights for the regression model are going to be fitted based on trajectories sampled using stochastic simulation and the constructed approximation for 100 random `failr` values. Table 2 gives the comparison of decision boundaries obtained by the two methods. Results suggest that for the considered parameters we slightly under-approximate the proportion of robots at  $t = 10.0$ .

**Direct Optimisation of a Reward Functional.** Alternatively, we can consider direct optimisation of a reward functional. For example, consider the following functional to find maximal value for `failr` that results in the goal being satisfied

$$Q^N[\pi] = \begin{cases} I(X_{\pi,s}^{(N)}(T)) + \log(\pi) + c_0 & \text{for } I(X_{\pi,s}^{(N)}(T)) \geq 0.5 \\ I(X_{\pi,s}^{(N)}(T)) - \log(\pi) & \text{otherwise} \end{cases}$$

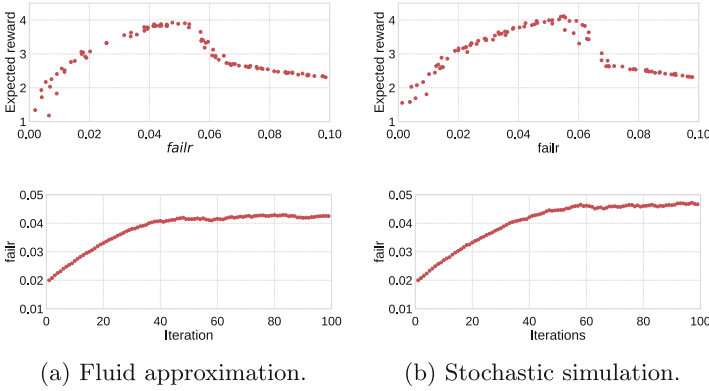
where  $c_0$  is some chosen constant and  $\pi \in [0, \infty)$  corresponds to the chosen policy. The first part of the reward functional corresponds to the satisfaction of the goal as defined previously. The constraint of `failr` being non-negative is taken into account by adding a logarithmic barrier function. The term  $-\log(\pi)$  is used to penalise `failr` values that are further away from satisfying the goal. Figures 5a



**Fig. 5.** Sampled reward functional.

and b show the reward functional sampled uniformly from  $\pi \in [0.0, 0.1]$ . Note that due to stochastic variance we are going to have values of  $\pi$  for which we are uncertain about whether the goal is going to be satisfied or not.

To optimise for this, currently discontinuous, reward functional we consider the method of policy gradient presented, for example, in [15] with parametrised policies  $\pi \sim \mathcal{N}(\mu, \sigma^2)$ —in a control scenario this would correspond to looking at a stochastic controller. Figure 6 shows the expected reward functional for  $\pi \sim \mathcal{N}(\mu, 0.05)$  with 100 samples of  $\mu$  taken uniformly from  $(0.0, 0.1]$ .



**Fig. 6.** Sampled reward functional for Gaussian policies and the gradient ascent initialised at  $\pi \sim \mathcal{N}(0.02, 0.005)$ . Reward estimates based on 10 samples of  $\pi$ .

Considering such Gaussian policies together with the approximation to mean behaviour allows us to implement a fast policy gradient algorithm for synthesising approximations to optimal policies for the system. In particular, the gradient of the functional at  $\pi$  is going to be estimated based on the ideas in Sect. 3 by  $\nabla Q^N[\pi] \sim (q[\pi + \epsilon] - q[\pi])/\epsilon$  where  $q$  is the functional corresponding to the expected reward of the fluid approximation. The evolution of policy parameter values for a simple gradient ascent algorithm is given in Fig. 6.

**Performance.** Multiple stochastic trajectories are needed to get a good estimate of the mean behaviour—for the model of 100 robots we used 100 trajectories, which took about 3s of computing on a single thread. The approximation took about 0.04s translating into a non-trivial speed-up in cases where trajectories for lots of parametrisations have to be considered. In particular, when generating 100 samples for considered reward functional this translates into roughly 5 min via stochastic simulation and 4s via the approximation.

## 5 Conclusion

In this paper we presented a framework for exploiting fluid approximation results in the context of policy synthesis for collective dynamics involving broadcast

communication. To that end we proposed a class of population CT-MDPs arising from systems with broadcast communication where the communication can be thought to separate the dynamics of the system into modes. To aid the construction of such models we introduced a language CARMA-C, based on CARMA, and outlined the semantics which gives a natural way for specifying policy synthesis problems in a high-level language. We discussed the application of fluid approximation in policy synthesis and suggested an approximation for the population models with mode switching for which the classic results cannot be applied directly. We used the proposed approximation to analyse the running example of a robot swarm.

For further work we plan to give a more formal treatment for the approximations for mode switching. In particular, the current method gives a good approximation to mean behaviour for times sufficiently far from where the mode change is expected to happen. However, this presents a limitation when we are interested in the behaviour of the system around the time of mode change or in the case of multiple modes where two mode changes can happen close to each other. To address this we aim to devise a more sophisticated approximation for switching time. For that we plan to consider the linear noise approximation [16], as done for example in [17], to help recover information about stochasticity and estimate the distribution of switching times.

**Acknowledgement.** This work was supported by EPSRC grant EP/L01503X/1 (CDT in Pervasive Parallelism).

## References

1. Buchholz, P., Dohndorf, I., Scheftelowitsch, D.: Optimal decisions for continuous time Markov decision processes over finite planning horizons. *Comput. OR* **77**, 267–278 (2017)
2. Butkova, Y., Hatefi, H., Hermanns, H., Krčál, J.: Optimal continuous time Markov decisions. In: Finkbeiner, B., Pu, G., Zhang, L. (eds.) ATVA 2015. LNCS, vol. 9364, pp. 166–182. Springer, Cham (2015). [https://doi.org/10.1007/978-3-319-24953-7\\_12](https://doi.org/10.1007/978-3-319-24953-7_12)
3. Gast, N., Gaujal, B.: A mean field approach for optimization in discrete time. *Discrete Event Dyn. Syst.* **21**(1), 63–101 (2011)
4. Kurtz, T.G.: Solutions of ordinary differential equations as limits of pure jump Markov processes. *J. Appl. Probab.* **7**(1), 49–58 (1970)
5. Loreti, M., Hillston, J.: Modelling and analysis of collective adaptive systems with CARMA and its tools. In: Bernardo, M., De Nicola, R., Hillston, J. (eds.) SFM 2016. LNCS, vol. 9700, pp. 83–119. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-34096-8\\_4](https://doi.org/10.1007/978-3-319-34096-8_4)
6. Piho, P., Georgoulas, A., Hillston, J.: Goals and resource constraints in CARMA. In: Proceedings of the Ninth International Workshop on the Practical Application of Stochastic Modelling (PASM), pp. 155–172 (2018)
7. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: Proceedings of the Sixth Annual Symposium on Logic in Computer Science (LICS 1991), pp. 266–277 (1991)

8. Caillaud, B., Delahaye, B., Larsen, K.G., Legay, A., Pedersen, M.L., Wasowski, A.: Constraint Markov chains. *Theor. Comput. Sci.* **412**(34), 4373–4404 (2011)
9. Georgoulas, A., Hillston, J., Milios, D., Sanguinetti, G.: Probabilistic programming process algebra. In: Norman, G., Sanders, W. (eds.) QEST 2014. LNCS, vol. 8657, pp. 249–264. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-10696-0\\_21](https://doi.org/10.1007/978-3-319-10696-0_21)
10. De Nicola, R., Latella, D., Loret, M., Massink, M.: A uniform definition of stochastic process calculi. *ACM Comput. Surv.* **46**(1), 5:1–5:35 (2013)
11. Legay, A., Delahaye, B., Bensalem, S.: Statistical model checking: an overview. In: Barringer, H., et al. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010). [https://doi.org/10.1007/978-3-642-16612-9\\_11](https://doi.org/10.1007/978-3-642-16612-9_11)
12. Bartocci, E., Bortolussi, L., Brázdil, T., Milios, D., Sanguinetti, G.: Policy learning in continuous-time Markov decision processes using Gaussian processes. *Perform. Eval.* **116**, 84–100 (2017)
13. Billingsley, P.: *Convergence of Probability Measures*, 2nd edn. Wiley, New York (1999)
14. Bortolussi, L.: Hybrid behaviour of Markov population models. *Inf. Comput.* **247**, 37–86 (2016)
15. Peters, J., Schaal, S.: Reinforcement learning of motor skills with policy gradients. *Neural Netw.* **21**(4), 682–697 (2008)
16. Van Kampen, N.: *Stochastic Processes in Physics and Chemistry*. North-Holland Personal Library. Elsevier Science, Amsterdam (2011)
17. Bortolussi, L., Lanciani, R.: Model checking Markov population models by central limit approximation. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 123–138. Springer, Heidelberg (2013). [https://doi.org/10.1007/978-3-642-40196-1\\_9](https://doi.org/10.1007/978-3-642-40196-1_9)



# Modeling Humans: A General Agent Model for the Evaluation of Security

Michael Rausch<sup>(✉)</sup>, Ahmed Fawaz, Ken Keefe, and William H. Sanders

University of Illinois at Urbana-Champaign, Urbana, IL, USA  
{mjrausc2,afawaz2,kjkeefe,whs}@illinois.edu

**Abstract.** Careful planning is needed to design cyber infrastructures that can achieve mission objectives in the presence of deliberate attacks, including availability and reliability of service and confidentiality of data. Planning should be done with the aid of rigorous and sound security models. A security modeling formalism should be easy to learn and use, flexible enough to be used in different contexts, and should explicitly model the most significant parts of the system of interest. In particular, the research community is increasingly realizing the importance of human behavior in cyber security. However, security modeling formalisms often explicitly model only the adversary, or simplistic interactions between adversaries and defenders, or are tailored to specific use cases, or are difficult to use. We propose and define a novel security modeling formalism that explicitly models adversary, defender, and user behavior in an easy and general way, and illustrate its use with an example.

**Keywords:** Human modeling · Quantitative cyber security modeling  
GAMES formalism · Cost benefit analysis · Risk analysis

## 1 Introduction

Institutions and enterprises rely on properly functioning cyber infrastructures to fulfill their intended missions. Impairments to proper operation can come from both accidental and malicious sources. While a sound engineering approach exists for designing and assessing systems that must tolerate accidental faults, there is no such corresponding methodology for systems that must be resilient to malicious attacks. Given the increasing likelihood of such attacks in the modern world, the lack of such a methodology will lead to increased uncertainty regarding the resilience of systems, and may result in designs that are susceptible to catastrophic failure when attacked.

Unfortunately, as society comes to rely more on cyber infrastructure, it is becoming an increasingly attractive target. The cyber infrastructure must be protected from the increased threat of attacks to ensure that it remains functional. Successful cyber attacks against such infrastructures can be very impactful. A few prominent examples include the cyber-attack on Ukraine's power grid in 2015, which deprived 230,000 residents of power [12]; the Stuxnet worm, which delayed Iran's nuclear program [11]; and data exfiltration attacks on U.S.

Democratic National Committee emails [2]. Those historical examples show how influential such attacks may be.

Cyber defenses must be carefully planned from the earliest stages of design to mitigate threats and ensure that systems will achieve availability, integrity, and confidentiality objectives, even in the face of attack. Practitioners and academics alike have long known the importance of accurate risk assessment [3,21], as it is key to designing effective security architectures. Risk assessment, while important, is difficult to perform correctly.

Formal computer security models help security experts overcome limitations, gain additional insight into a system, and confirm conjectures. Assumptions are made explicit in models, and the assumptions, input parameters, methodology, and results of a model may be audited by an outside party. The modeling formalism could also serve as a common language that would allow security experts and experts from other domains to more easily collaborate on a security model. Finally, the models would add additional mathematical and scientific rigor to risk analysis. All of these benefits speak to the need for security models.

We believe that, when constructing security models, it is necessary to consider not only the system to be defended, but also the humans that interact with that system: adversaries, defenders, and users. If the model does not consider these human entities, it is much less likely to be accurate. Unfortunately, many security modeling formalisms in use today fail to explicitly model all of the human entities in the system, or do so in an overly simplistic way. Models that ignore or improperly model human users are significantly less likely to be helpful to system architects.

To address this issue, we propose a new security modeling formalism, the *General Agent Model for the Evaluation of Security (GAMES)*, which allows a system engineer to explicitly model and study the adversaries, defenders, and users of a system, in addition to the system itself. These models are executed to generate security-relevant metrics to support design decisions. The formalism is used to easily build realistic models of a cyber system and the humans who interact with it. We define an agent to be a human who may perform some action in the cyber system: an adversary, a defender, or a user. The formalism enables the modular construction of individual state-based agent models of the three types. The formalism also allows the modeler to compose these individual agent models into one model so the interaction among the adversaries, defenders, and users may be studied. Once constructed, this composed model can be executed. During the execution, each individual adversary, defender, or user utilizes an algorithm or policy to decide on what action the agent will take to attempt to move the system to a state that is advantageous for that agent. The outcome of the action is then probabilistically determined, and the state updated. Modelers using GAMES have the flexibility to determine how the agents will behave: either optimally, or according to a modeler-defined policy. The model execution generates metrics that aid in risk assessment, and helps the security analyst suggest appropriate defensive strategies. The formalism helps security architects make cost-effective, risk-aware decisions as they design new cyber systems.

## 2 Related Work

Many academics and practitioners have recognized the need for models for computer security. Many examples may be found in surveys, e.g., surveys of papers on game-theoretic security modeling approaches [14, 17], a survey of attack tree and attack-defense tree models [9], and a survey that includes a useful taxonomy of security models [22].

Such modeling approaches are a step in the right direction, but pose their own sets of limitations, especially in the ways they model the humans who interact with the cyber portion of the system. Some modeling approaches explicitly model only the adversary, e.g., the well-known attack tree formalism [18]. Other formalisms model the adversary and defender, but neglect the role and impact of users. For example, the attack-defense tree formalism [8] and related attack-defense graph formalism [10] extend the attack tree formalism to include one attacker/defender pair, but do not model multiple adversaries or multiple defenders, or any users, which limits the effectiveness of the models. There exist some approaches and tools for modeling multiple adversaries, defenders, and users in a system, e.g., Haruspex [1], and some agent-based simulation approaches [4, 23]. However, these approaches and tools are not in common use, for a number of reasons. Often, the models lack realism because of model oversimplification, are tailored to narrow use cases, produce results that are difficult to interpret, or are difficult to use, among other problems. Finally, security modeling formalisms, particularly those that take a game-theoretic approach, often assume that attackers and defenders are rational. Some examples include [5, 7, 24]. However, this assumption may not produce useful security models [22].

The GAMES approach is inspired, in part, by the ADVISE formalism [13]. In particular, we extend and generalize the ADVISE formalism's Attack Execution Graph. However, there are important differences between the two formalisms. Models constructed using the ADVISE formalism can explicitly model only an adversary. Defenders, users, and the interactions between actors cannot be explicitly modeled with ADVISE. The GAMES formalism recognizes the importance of considering defender and user actions when designing secure systems, and explicitly incorporates defenders and users as first class model elements.

## 3 A General Agent Model for Evaluation of Security

In this section, we give a comprehensive overview of the *General Agent Model for the Evaluation of Security* (GAMES) formalism. We will first explain in detail how the individual agent models are defined and composed together. Next, we will describe how the models are executed, including the algorithms or policies that the various adversaries, defenders, or users of the system may utilize to decide upon the best course of action. Finally, we will describe the kind of results these models will produce, how they may be interpreted, and how the models may be used. One or more agent models may be joined in a composed model.

The composed model may then be combined with a reward model to create a model that may be executed to obtain security-relevant metrics, which a security analyst may use to gain additional insight into the system being modeled.

### 3.1 Model Formalism Definition

A model defined using the GAMES formalism will consist of one or more agent submodels and a model composed of the agent submodels, that describes how the submodels relate to one another. An agent model represents one acting entity in the cyber system: an adversary, a defender, or a user. The framework allows a modeler to define many different kinds of agents. For example, a modeler may define a malicious insider adversary, a nation-state adversary, a network operator defender, and a customer user of the system. Once the individual agent models have been composed into a model that defines their relationships, the modeler can study, for example, (1) how two adversaries may cooperate to achieve a goal on the network, (2) the effectiveness of the network operator's defensive actions, and (3) how the actions of the adversaries and defender impact the user's experience and behavior. We shall first describe the agent models, and then how they may be composed together.

**Agent Model:** An *agent model* describes the state an agent may read or write, how this state is initialized, the set of actions the agent may utilize to change the state of the model, the payoff the agent receives given a particular state, and the agent decision algorithm that determines the action the agent will take given the state of the system. The agent model is composed of an *Action Execution Graph* (AEG) and an *agent profile*. The AEG may be thought of as an extension of the attack-tree formalism [18]. Unlike attack trees, an AEG contains state, and therefore shares some similarities with generalized stochastic Petri nets (GSPNs) [15] and stochastic activity networks (SANs) [16]. However, it is most similar to the Attack Execution Graphs of ADversary VIEw Security Evaluation (ADVISE) models [13]. Action Execution Graphs are more general than Attack Execution Graphs, since they can be used to model any agent type, whereas Attack Execution Graphs are used only to model adversaries. The agent profile defines how the state is initially defined; the payoff function, which assigns the payoff the agent achieves given a particular model state; and the agent decision algorithm the agent uses to decide what actions to take to change the state of the system.

An *Action Execution Graph* (AEG) is a labeled transition system defined by

$$\langle S, A, C \rangle,$$

where  $S$  is some finite set of state variables that an agent may read or write,  $A$  is some finite set of actions an agent may take, and the relation  $C$  defines a set of directed connecting arcs from  $p \in S$  to  $a \in A$ , and from  $a \in A$  to  $e \in S$ , where  $p$  is a prerequisite state variable whose value directly affects the behavior of the action  $a$  (e.g., whether the action may be attempted, how much it will cost, and how long it will take), and  $e$  is a state variable that may be changed when action



$a$  is performed. The states and actions together,  $S \cup A$ , are the vertices of the AEG, while the connecting arcs,  $C$ , are the edges.

Each state variable in  $S$  may have a single value or a finite sequence of values. Each value can be drawn from any countable subset of the real numbers. The state variables may be further subdivided, at the discretion of the modeler, into different classes. For example, state variables relating to an adversary agent may be divided into those that relate to access (like physical access to the system, network access, or administrator access to individual machines), knowledge (of the routing protocols used, company policies, encryption schemes, etc.), or skill (in decryption, social engineering, privilege escalation, and the like), as proposed by LeMay [13]. This subdivision of state variables is superficial, but may serve as a conceptual aid to the modeler.

An action,  $a \in A$ , may be used by an agent to attempt to change the state of the system. It is defined by the tuple

$$\langle B, T, C, O \rangle.$$

First, let  $Q$  be the countable set of model states, where a model state (also known as a *marking*) is given by a mapping  $\mu : S \rightarrow \mathbb{R}$ .

$B : Q \rightarrow \{True, False\}$  is a Boolean precondition that indicates whether the action is currently enabled. An agent may not take an action that is not enabled.

$T : Q \rightarrow \mathbb{R}^{\geq 0}$  is the length of time needed to complete the action, and is a random variable.

$C : Q \rightarrow \mathbb{R}^{\geq 0}$  is the cost to the agent for attempting the action.

$O$  is the finite set of outcomes of the action (such as success or failure). Each outcome  $o \in O$  is defined by the tuple

$$\langle Pr, E \rangle$$

where

$Pr : Q \rightarrow [0, 1]$  is the probability that outcome  $o \in O$  will occur.

$E : Q \rightarrow Q$  is the *effect* of the outcome, i.e., the function that transitions the system to a new state upon the selection of  $o \in O$ .

An agent profile is composed of three distinct components. The first specifies the initial value for each state variable in the Action Execution Graph. The second is a function that accepts as input the state of the model and outputs the payoff the agent accrues given that the model is in that particular state.

The third and final component of the agent profile is the agent decision algorithm. The agent decision algorithm will take as input the state of the model, and output an action. In general, the agent decision algorithm will attempt to select an action that will maximize the agent's payoff. The modeler may choose to assign to the agent one of several predefined agent decision functions, or specify a custom decision function. The various predefined decision functions may be based on well-studied techniques drawn from game theory and artificial intelligence, or novel algorithms that may be developed in the future. If, in the opinion of the modeler, none of the predefined decision functions realistically describe an agent's real behavior, the modeler may define a custom agent decision algorithm.

We will explain the various adversary decision functions in greater detail in the section on model execution.

**Model Composition:** Agent models should be composed together to exploit the full power of the GAMES approach. Agent models are modular and independently functional, so one agent model could be executed by itself if the application warrants. For example, if the modeler is only interested in studying the adversary's behavior and is not interested in the defender's response or the impact on the user's behavior, he or she could build a standalone adversary agent model similar to an ADVISE model [13]. However, the chief aim of the GAMES formalism is to give those in charge of making security decisions the ability to easily model the interaction among adversaries, defenders, and users in cyber systems. The composed model will define how the agent models will be allowed to interact with one another. We utilize the state-sharing approach of the Replicate/Join formalism [19] to enable agents to interact with one another. This state-sharing approach allows an agent to read and write state in another agent model. Agents can pass messages to one another through shared state variables that serve as communication channels. Defenders can collaborate and adversaries can collude using the channels.

### 3.2 Model Execution

**Overview:** Model execution is accomplished via discrete-event simulation paired with the agent decision algorithms, as seen in Algorithm 1.

---

#### Algorithm 1. Model Execution

---

```

1: procedure EXECUTE MODEL
2:   Time  $\leftarrow$  0
3:   State  $\leftarrow$   $s_0$ 
4:   Agents  $\leftarrow$  {agent1, agent2, ... agentn}
5:   while Time < EndTime do
6:     for all agent  $\in$  Agents do
7:       agentaction  $\leftarrow$  AgentDecisionAlgorithm(State)
8:       Time  $\leftarrow$  time to complete earliest completed agent action
9:       Outcome  $\leftarrow$   $o$ , where  $o$  is the randomly chosen outcome of the action
10:      State  $\leftarrow$  Effect(State, Outcome)

```

---

**Decisions:** The formalism allows the modeler to specify adversary, defender, and user behavior as realistically as possible to produce high-quality results that will support design and defense decisions. Today, security modeling formalisms are commonly built around one agent behavior specification approach. The agent behavior specification is driven by the agent decision algorithm, which the agent uses to choose actions to perform. Often this agent behavior specification is inspired by and draws from some particular technique in game theory, artificial intelligence, or psychology. The GAMES formalism may be used to test and

compare a wide variety of agent decision algorithms. The modeled entities may act cooperatively or competitively with others to achieve a goal, using well-established techniques from artificial intelligence research.

### 3.3 Metrics

The primary intended use of the GAMES formalism is to provide insight to support those charged with making decisions that affect system security. That insight comes in the form of quantitative metrics, which are results of the executed model. We leverage the theory of reward-model-based performance variables [20] to construct metrics for GAMES models. It provides a simple formulation that is very flexible, and thereby allows us a great deal of freedom and creativity in the variety of metrics we may specify. For a demonstration, see Sect. 4. We believe that the estimated payoffs, costs, and probabilities of success for the various actors will be among the most interesting metrics that GAMES models can calculate, particularly for modelers that are interested in cost-benefit analysis.

### 3.4 Limitations

We argue that quantitative model results can produce useful insights into the security of the system being considered, but we do not claim that the metrics produced by executing a GAMES model are infallible, or that they should be treated as such. The metrics may not accurately reflect reality; they may give misleading results if the model is constructed incorrectly or if the model input parameters are inaccurate. Security analysts must work closely with system architects and other experts to verify that the model is a good representation of the actual system. In addition, whenever possible, the results given by the model should be validated by experiments performed on the system itself. Experiments performed on the system can also be used to improve the model. The adversary, defender, and user behavior in the model may be checked against data from intrusion detection systems, analytics data, academic studies, and surveys results. The issue of inaccurate input parameters may be mitigated with a well-executed design exploration, which may be achieved by a sensitivity analysis.

Some are concerned that models that produce quantitative security metrics may be misused, to the detriment of the security of systems, in part because people may have more confidence in the model results than they should [22]. We believe, however, that the alternative of having no formal security model is much worse. Security analysts almost always carry informal mental models of their systems in their heads, and those models suffer from many of the same limitations as formal security models, while also suffering from their own additional limitations. The GAMES formalism gives an analyst the ability to turn a mental model into a formal, concrete, rigorous, auditable model. Indeed, the quantitative metrics produced by these models represent an important step towards the development of a science of security [6].

## 4 Example

We present an example model to illustrate how security practitioners could use the GAMES formalism to make security decisions. The goal of the model is to help an operator make an informed choice among several strategies that could be used to defend user accounts from being compromised by an adversary. We limit the size of the model for ease of explanation; we could obtain a more realistic model by expanding the size to include more details. We solved the example model using an implementation of the GAMES framework written in Python.

In the model, there are four agent types: adversary, operator/defender, user, and media. The operator's goal is to provide a service to the users in exchange for a fee. The operator must maintain a Web-accessible account for every user using the system. Each user has the goal of using the service (and consequently the account) with minimal effort; if the effort of using the service becomes too great, the user will switch to a different service. The adversary wants to obtain unauthorized access to as many accounts as possible. The last agent type, the media, will publicize a successful hack if it learns about it. The model contains one adversary instance, one defender instance, one media instance, and two hundred user instances. We model the two hundred users as two hundred copies of the user submodel with different state initializations.

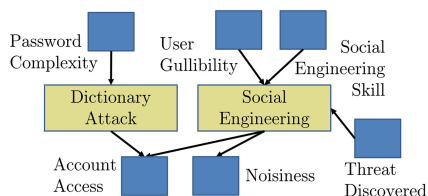
The model can be used to help a security practitioner choose among different defensive strategies by comparing their effectiveness across a variety of metrics. Specifically, we show how it may be used to compare three different operator defensive policies (passive, aggressive, and balanced) with respect to net defender profit, the time to discover the initial breach in security, and the number of accounts compromised in the attack. The metrics calculated by the model would help the defender choose a policy to follow in an implemented system.

### 4.1 Composed Model

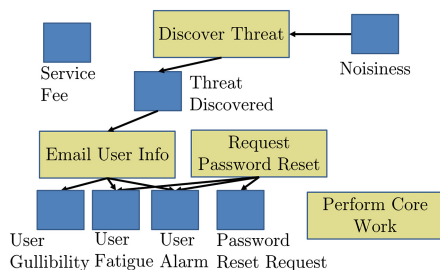
The composed model consists of submodel instances of four submodel types, one for each of the four agent types in the model. There are two hundred instances of the user submodel type, and one instance of each other submodel type. Each submodel instance shows a particular agent's view of the environment (expressed in the Action Execution Graph) and the decision algorithm the agent uses to choose an action at each stage of the simulation.

There are twelve state variables in this model, as follows.

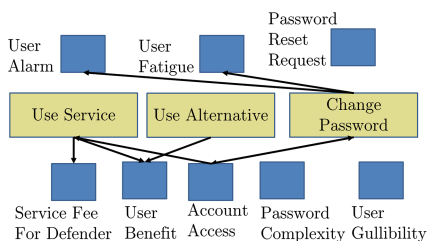
- **Account Access:** The status of a user's account.
- **Noisiness:** The cumulated evidence the attacker left (observable by the defender) as a result of the actions the attacker took.
- **Password Complexity:** A user's password strength.
- **Password Reset Requested:** This Boolean-valued state variable indicates whether the defender has requested that a user change an account password. In effect, it serves as a communication channel that the defender utilizes to convey a request for a particular action.



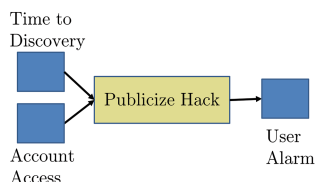
**Fig. 1.** A graphical representation of the adversary's Action Execution Graph.



**Fig. 2.** A graphical representation of the defender's Action Execution Graph.



**Fig. 3.** A graphical representation of a user's Action Execution Graph.



**Fig. 4.** A graphical representation of the media's Action Execution Graph.

- **Social Engineering Skill:** The adversary's level of skill in this attack.
- **Service Fee for Defender:** Payment made by the user for account access.
- **Threat Discovered:** Defender's knowledge of an attempted attack.
- **Time to Discovery:** Days until the media learn of a successful attack.
- **User Alarm:** User's fear of loss due to an account compromise.
- **User Benefit:** The value of the benefit the user accrues.
- **User Fatigue:** The user's level of fatigue from using the defender's service.
- **User Gullibility:** Susceptibility of a user to social engineering.

In the model, the *Noisiness*, *Password Reset Requested*, *Social Engineering Skill*, *Service Fee for Defender*, *Threat Discovered*, and *Time to Discovery* state variables each have a single value. The *Account Access*, *Password Complexity*, *User Alarm*, *User Benefit*, *User Fatigue*, and *User Gullibility* state variables have a sequence (or array) of values, with one value for each of the two hundred users in the model. The user submodels form an ordered list, such that the  $i^{th}$  user can only read from or write to the  $i^{th}$  value of those state variables that have a sequence of values. (Note that each of those state variables is included in the users' Action Execution Graphs.) That allows us to use one state variable, for example, to hold account status information for every individual user in the system, rather than two hundred individual copies of a state variable.

Every value of every state variable is initialized to zero, with three exceptions. The first exception, *Social Engineering Skill*, is initialized to 7/10 at the beginning of the simulation. This indicates that the adversary has above-average skill in social engineering. The second and third exceptions are *Password Complexity* and *User Gullibility*. Some users have more cybersecurity knowledge, and motivation to act on that knowledge, than others; in this model, we have 80 sophisticated users and 120 average users, and these user types are reflected in the values assigned to these two state variables. We randomly selected 80 indices from 200 users to represent sophisticated users, and for each index in this set of 80 the corresponding values of *Password Complexity* and *User Gullibility* are set to 8/10 and 2/10, respectively. That indicates that the sophisticated users have strong passwords and are relatively unlikely to be susceptible to social engineering attacks. The remaining 120 values of *Password Complexity* and *User Gullibility* are initialized to 4/10 and 8/10, respectively.

## 4.2 Adversary Submodel

This submodel contains the attacker's Action Execution Graph (visually represented in Fig. 1), and the agent decision algorithm.

**Action Execution Graph:** The attacker may choose either (1) to perform a dictionary attack on the database in an attempt to discover passwords via the *Dictionary Attack* action, or (2) to conduct a social engineering attack to trick users into revealing their passwords via the *Social Engineering Attack* action. These actions, along with their precondition and postcondition state variables, form the adversary's Action Execution Graph. The probability of a successful social engineering attack depends on the user's gullibility, the adversary's skill in social engineering, and whether or not the defender discovers the attack. If the attack is attempted, there is some probability that it will generate a small amount of noise. If successful, the attack will give the adversary access to the account. The probability of a successful dictionary attack depends directly on the complexity of the user's password.

**Decision Algorithm:** The attacker has two actions to choose from: (1) starting a dictionary-based brute-force attack to obtain the password, or (2) attempting a social engineering attack to trick users into giving the attacker access to their passwords. The adversary will attempt a dictionary attack on a password if (1) the minimum value in *Password Complexity* is less than 5, (2) *Social Engineering Skill* is less than 5, and (3) *User Gullibility* is greater than 5. If these conditions have not been satisfied, the adversary will choose to perform a social engineering attack.

## 4.3 Defender Submodel

This submodel contains the defender/operator's view of the overall model's state and the actions that the agent could take to change the state. For a visual depiction of the defender's Action Execution Graph, see Fig. 2.

**Defender Action Execution Graph:** The defender can choose from four actions. The defender has the option of attempting to discover the adversary's activities (the *Discover Threat* action), sending an email to inform users of relevant threats or educate them about cybersecurity best practices (the *Email User Info* action), requesting that the user perform a password reset (the *Request Password Reset* action), or working on core business activities (the *Perform Core Work* action).

First, if a threat has not already been discovered, the defender can try to discover the adversary's attack. The probability of successfully uncovering the attack depends on the noisiness of the attack (the value of the *Noisiness* state variable). If the defender is successful, the attack is discovered (as indicated by *Threat Discovered*). Second, the defender can choose to send an email to the users with the *Email User Info* action to reduce their susceptibility to social engineering attacks. The email's effectiveness increases if the threat has been discovered. If *Threat Discovered* indicates that the threat has been discovered, the action will greatly reduce the user's gullibility; otherwise, the action will reduce it only slightly. Sending the email also slightly increases every user's fatigue, and, if the threat has been discovered, alarm. Third, the defender can also use the *Request Password Reset*, which sets the *Password Reset Requested* state variable to true. It also greatly increases the users' fatigue (because they have to create and memorize new passwords) and slightly increases the users' alarm (as tracked by the *User Fatigue* and *User Alarm* state variables, respectively). Fourth, and finally, at any time, the defender can attempt the *Perform Core Work* action. This action represents work that the operator can do that isn't directly related to cybersecurity. It exists as a sort of placeholder, because in reality, the operator is likely to spend most of his or her time on tasks that are unrelated to cybersecurity.

**Decision Algorithm:** We evaluate three different policies that a defender could choose to employ: aggressive, passive, and balanced.

First, the *aggressive policy* calls for defenders to take actions frequently to defend their networks and keep their users educated. Specifically, a defender will (1) email the users every ninety days to educate them about cyber security and how to avoid social engineering attacks, (2) request that the users reset the account password every ninety days (and whenever an ongoing attack has been detected), (3) attempt to discover threats by thoroughly analyzing security logs (from tools such as intrusion detection systems) every seven days, and (4) do work unrelated to cybersecurity (such as maintaining infrastructure, providing services to account users, etc.) on the remaining days.

If the defender employs the *passive policy*, he or she will take defensive actions infrequently. Specifically, the defender will send out a password reset request and an email educating users once every year and do work unrelated to cybersecurity on the remaining days.

Finally, the *balanced policy* can be used by a defender to strike a balance between the aggressive and passive approaches. The policy calls for the defender (1) to send an email educating users every ninety days about how to avoid falling

for social engineering attacks, (2) to request a password reset every year (and whenever a threat has been discovered), and (3) to attempt to discover threats by examining security logs every two weeks.

#### 4.4 User Submodel

As loyal customers, the users wish to keep using the defender's service, because it meets their needs more effectively than competitor services do. As a result, the customers are cooperative and will reset their passwords if the defender asks them to. However, the defender can take actions that annoy the user (e.g., sending out emails too often), or the media could alarm the users if a compromise is publicized. Either case could drive the users away from the defender's service.

**Action Execution Graph:** Each user's Action Execution Graph consists of three actions and seven state variables. A graphical representation can be found in Fig. 3. The three actions that the user could choose are *Use Service*, *Use Alternative Service*, and *Change Password*. First, the *Use Service* action is essentially the default action for this agent. It is enabled as long as *Account Access* does not indicate that the user has abandoned the account. *Service Fee for Defender* and *User Benefit* are incremented (by values of one and three, respectively) when *Use Service* is attempted. Second, the *Use Alternative Service* action can be taken by a user who is frustrated with the defender's service. The action is always enabled. It has only one postcondition state variable, *User Benefit*, which is incremented by two when the action is attempted. Third, the main effect of the *Change Password* action is to remove the adversary's access to the account (if it had previously been gained) by changing the password. The action is enabled as long as the user hasn't abandoned his or her account. The *Password Reset Request* state variable may signal to the user that the defender believes it would be beneficial to change the password. *User Alarm* is incremented by a small amount when the user changes the password, but *User Fatigue* is incremented by a much larger amount, to model the difficulty of creating and remembering a new password.

**User Decision Algorithm:** The user's policy is expressed in the following list: (1) If the defender requests that the user reset the password and the user has not abandoned the account, reset the password. (2) Otherwise, if the *User Alarm* and *User Fatigue* state variables both have values of less than 9, and the account has not been abandoned, use the defender's service. (3) Otherwise, if the user's fatigue is greater than or equal to 9, but the user's alarm is less than 9, and the account has not been abandoned, with some low probability switch to a competitor's service, otherwise continue to use the defender's service. (4) Otherwise, if the user's alarm is greater than or equal to 9, and the account has not been abandoned, with some high probability switch to a competitor's service, otherwise, continue using the defender's service. (5) Otherwise, use a competitor's service.



## 4.5 Media Submodel

The media submodel is the simplest agent in the model. Its one goal is to publicize a successful widespread attack against the defender's service. A graphical representation of the media's Action Execution Graph is given in Fig. 4. The media's Action Execution Graph consists of only a single action, the *Publicize Hack* action, along with the state variables *Time to Discovery*, *Account Access*, and *User Alarm*. If at least 10% of user accounts have been compromised, the action will become enabled. Once the action is taken, it will complete at the end of the time indicated by *Time to Discovery*. If at least 10% of user accounts remain compromised at the end of that time, the actions will set the value of every user's *User Alarm* state variable to the maximum value, which could trigger abandonment of accounts by users. The media agent has a straightforward decision algorithm since the media have only one choice of action. The media will publicize the attack, increasing the users' alarm, six months after the first account has been compromised, if the defender doesn't quickly contain the problem.

## 4.6 Reward Model: Defining Metrics

The model gains its usefulness by supplying relevant metrics that will help the system architects and policy designers make good security choices. The model calculates seven metrics that give insight into the modeled system.

To begin, we track the defender's profit. The defender gains revenue from every user each day that he or she uses the service. The total revenue gained at the end of a 2-year period can be found by merely observing the value of the *Service Fee for Defender* state variable at the end of a 2-year simulation. The defender incurs costs by performing defensive actions. Every time a user takes action during a simulation, the cost to perform that action is calculated and stored in a running counter. At the end of the simulation, the value of that counter can be observed to determine the total direct cost of a particular policy. The profit is revenue minus costs.

Also, three metrics track the state of the 200 accounts in the defender's care at the end of the simulation: the mean number of uncompromised active accounts, the mean number of compromised accounts, and the mean number of abandoned but uncompromised accounts. The first of the three metrics gives the number of actively used, secure accounts (accounts that the adversary cannot access and have not been abandoned). The second of the three metrics tracks the number of accounts that are compromised by the adversary at the end of the simulation (whether or not the account has been abandoned by its user). The last of the three metrics gives the number of uncompromised accounts that have been abandoned by their users (because of alarm or fatigue) at the end of the simulation. All three of these metrics are calculated by observing the value held by each of the 200 instances of the *Account Access* state variable.

The final metric gives the time from the first successful account compromise to the time the defender discovers the threat. The first time the adversary compromises an account, the current simulation time is recorded. Similarly, the first

time the *Discover Attack* action successfully completes, the current simulation time is noted. The metric is the difference between the two times.

#### 4.7 Model Execution and Results

The complete model may be executed to calculate the defined metrics, as described in Sects. 3.2 and 3.3. Every action in this particular model takes one day to complete (except the *Publicize Hack* action, which takes 180 days to complete). For that reason, the simulation step size is one day. We simulate the agents' behaviors over two years of simulation time to obtain the relevant metrics. All results are simulated with a 95% confidence interval.

**Table 1.** Simulation results

	Passive	Balanced	Aggressive
<i>Defender stats</i>			
Revenue	93770 ± 540	145800 ± 0	144029 ± 111
Costs	2200 ± 0	28300 ± 0	59610 ± 20
Profit	91570 ± 536	117500 ± 0	84418 ± 122
Days to detect	Never detected	17 ± 2	12 ± 1
<i>Account info</i>			
# Uncompromised	19 ± 1	200 ± 0	182 ± 1
# Compromised	120 ± 0	0 ± 0	0 ± 0
# Abandoned	61 ± 1	0 ± 0	18 ± 1

The results are presented in Table 1. They show that the balanced policy has a clear advantage over the other two policies. The defender will earn the highest average net profit and have the most uncompromised accounts at the end of the two-year period if the balanced policy is used. When compared to the balanced policy, the aggressive policy brings in a similar amount of revenue, but incurs much higher costs (due to the frequency of defensive actions taken with this policy), so the defender earns a significantly lower profit. The defender, using an aggressive policy, successfully thwarts the adversary's attempts to compromise accounts, but the defender's frequent actions annoy some users, driving them to abandon the service. The aggressive policy leads to slightly earlier threat detection than the balanced policy. The passive policy costs the defender much less than the aggressive or balanced policies. However, revenue is not as high, so average net profit is also low. The abandonment of so many user accounts explains the low revenue. We investigated why so many more users abandoned their accounts here than for the other two policies. We found that the media never publicize the hack if the defender uses the aggressive or balanced policies (because not enough accounts are compromised for a long enough time to be newsworthy). However, because the passive defender does so little to counter the

attacker, many accounts are compromised, which triggers the media publication of the hack, which alarms users and drives them to abandon their accounts.

## 5 Conclusion

In this paper we argue that cyber security models must explicitly incorporate all relevant agents to provide an accurate view of the overall system behavior, and that the agents must be modeled in a realistic manner. Formalisms that only model adversary behavior or simple adversary-defender conflict behavior do not accurately reflect the reality found in cyber systems that are manipulated and used by many different human entities. To solve this problem, we propose a new, easy-to-use modeling framework, the General Agent Model for the Evaluation of Security. This framework allows the modeler to construct different agent sub-models, which may be composed together and executed to calculate metrics that give insight into system behavior. Each submodel consists of the agent's view of the state, the actions available to the agent, and the agent's customizable decision algorithm. We demonstrated the richness of the formalism with an example, which incorporated a number of different agents with different goals and policies. The GAMES formalism is a significant step forward in the quest to give analysts the ability to create realistic security models of cyber systems.

## References

1. Baiardi, F., Corò, F., Tonelli, F., Bertolini, A., Bertolotti, R., Guidi, L.: Security stress: evaluating ICT robustness through a monte carlo method. In: Panayiotou, C.G.G., Ellinas, G., Kyriakides, E., Polycarpou, M.M.M. (eds.) CRITIS 2014. LNCS, vol. 8985, pp. 222–227. Springer, Cham (2016). [https://doi.org/10.1007/978-3-319-31664-2\\_23](https://doi.org/10.1007/978-3-319-31664-2_23)
2. Buratowski, M.: The DNC server breach: who did it and what does it mean? *Netw. Secur.* **2016**(10), 5–7 (2016)
3. Eloff, J., Labuschagne, L., Badenhorst, K.: A comparative framework for risk analysis methods. *Comput. Secur.* **12**(6), 597–603 (1993). [https://doi.org/10.1016/0167-4048\(93\)90056-B](https://doi.org/10.1016/0167-4048(93)90056-B)
4. Gorodetski, V., Kottenko, I., Karsaev, O.: Multi-agent technologies for computer network security: attack simulation, intrusion detection and intrusion detection learning. *Int. J. Comput. Syst. Eng.* **18**(4), 191–200 (2003)
5. Grossklags, J., Christin, N., Chuang, J.: Secure or unsure? A game-theoretic analysis of information security games. In: Proceedings of the 17th International Conference on World Wide Web, WWW 2008, pp. 209–218. ACM, New York (2008)
6. Herley, C., v. Oorschot, P.C.: SoK: science, security and the elusive goal of security as a scientific pursuit. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 99–120, May 2017
7. Izmalkov, S., Micali, S., Lepinski, M.: Rational secure computation and ideal mechanism design. In: 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2005), pp. 585–594, October 2005. <https://doi.org/10.1109/SFCS.2005.64>

8. Kordy, B., Mauw, S., Radomirović, S., Schweitzer, P.: Attack-defense trees. *J. Logic Comput.* **24**(1), 55–87 (2012). <https://doi.org/10.1093/logcom/exs029>
9. Kordy, B., Piètre-Cambacédès, L., Schweitzer, P.: Dag-based attack and defense modeling: don't miss the forest for the attack trees. *CoRR abs/1303.7397* (2013). <http://arxiv.org/abs/1303.7397>
10. Kramer, J.: Attack-defence graphs: on the formalisation of security-critical systems. Master's thesis, Saarland University (2015). [https://www-old.cs.uni-paderborn.de/uploads/tx\\_sibibtex/main\\_01.pdf](https://www-old.cs.uni-paderborn.de/uploads/tx_sibibtex/main_01.pdf)
11. Langner, R.: Stuxnet: dissecting a cyberwarfare weapon. *IEEE Secur. Priv.* **9**(3), 49–51 (2011)
12. Lee, R.M., Assante, M.J., Conway, T.: Analysis of the cyber attack on the Ukrainian power grid. *SANS Industrial Control Systems* (2016)
13. LeMay, E.: Adversary-driven state-based system security evaluation. Ph.D. thesis, University of Illinois at Urbana-Champaign (2011). [https://www.perform.illinois.edu/Papers/USAN\\_papers/11LEM02.pdf](https://www.perform.illinois.edu/Papers/USAN_papers/11LEM02.pdf)
14. Manshaei, M.H., Zhu, Q., Alpcan, T., Başçar, T., Hubaux, J.P.: Game theory meets network security and privacy. *ACM Comput. Surv.* **45**(3), 25:1–25:39 (2013)
15. Marsan, M.A., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: *Modelling with Generalized Stochastic Petri Nets*, 1st edn. Wiley, New York (1994)
16. Meyer, J.F., Movaghar, A., Sanders, W.H.: Stochastic activity networks: Structure, behavior, and application. In: *Proceedings of the International Conference on Timed Petri Nets*, Torino, Italy, pp. 106–115, July 1985
17. Roy, S., Ellis, C., Shiva, S., Dasgupta, D., Shandilya, V., Wu, Q.: A survey of game theory as applied to network security. In: *2010 43rd Hawaii International Conference on System Sciences (HICSS)*, pp. 1–10. IEEE (2010)
18. Salter, C., Saydjari, O.S., Schneier, B., Wallner, J.: Toward a secure system engineering methodology. In: *Proceedings of the 1998 Workshop on New Security Paradigms, NSPW 1998*, pp. 2–10. ACM, New York (1998)
19. Sanders, W.H., Meyer, J.F.: Reduced base model construction methods for stochastic activity networks. *IEEE J. Sel. Areas Commun.* **9**(1), 25–36 (1991). Special issue on Computer-Aided Modeling, Analysis, and Design of Communication Networks
20. Sanders, W.H., Meyer, J.: A unified approach for specifying measures of performance, dependability, and performability. In: Avizienis, A., Kopetz, H., Laprie, J. (eds.) *Dependable Computing for Critical Applications, Dependable Computing and Fault-Tolerant Systems*, vol. 4, pp. 215–237. Springer, Vienna (1991). [https://doi.org/10.1007/978-3-7091-9123-1\\_10](https://doi.org/10.1007/978-3-7091-9123-1_10)
21. Straub, D.W., Welke, R.J.: Coping with systems risk: security planning models for management decision making. *MIS Q.*, 441–469 (1998)
22. Verendel, V.: Quantified security is a weak hypothesis: a critical survey of results and assumptions. In: *Proceedings of the 2009 Workshop on New Security Paradigms Workshop, NSPW 2009*, pp. 37–50. ACM, New York (2009)
23. Wagner, N., Lippmann, R., Winterrose, M., Riordan, J., Yu, T., Streilein, W.W.: Agent-based simulation for assessing network security risk due to unauthorized hardware. In: *Proceedings of the Symposium on Agent-Directed Simulation, ADS 2015*, pp. 18–26. Society for Computer Simulation International, San Diego (2015)
24. You, X.Z., Shiyong, Z.: A kind of network security behavior model based on game theory. In: *Proceedings of the 4th International Conference on Parallel and Distributed Computing, Applications and Technologies, PDCAT 2003*, pp. 950–954, August 2003. <https://doi.org/10.1109/PDCAT.2003.1236458>



# Approximate Time Bounded Reachability for CTMCs and CTMDPs: A Lyapunov Approach

Mahmoud Salamati<sup>1</sup>(✉), Sadegh Soudjani<sup>2</sup>(✉), and Rupak Majumdar<sup>1</sup>(✉)

<sup>1</sup> Max Planck Institute for Software Systems,  
Kaiserslautern and Saarbrücken, Germany  
{msalamati, rupak}@mpi-sws.org

<sup>2</sup> School of Computing, Newcastle University, Newcastle upon Tyne, UK  
sadegh.soudjani@ncl.ac.uk

**Abstract.** Time bounded reachability is a fundamental problem in model checking continuous-time Markov chains (CTMCs) and Markov decision processes (CTMDPs) for specifications in continuous stochastic logics. It can be computed by numerically solving a characteristic linear dynamical system, which is computationally expensive. We take a control-theoretic approach and propose a reduction technique that finds another dynamical system of lower dimension (number of variables), such that numerically solving the reduced dynamical system provides an approximation to the solution of the original system with guaranteed error bounds. Our technique generalises lumpability (or probabilistic bisimulation) to a quantitative setting. Our main result is a Lyapunov function characterisation of the difference in the trajectories of the two dynamics that depends on the initial mismatch and exponentially decreases over time. In particular, the Lyapunov function enables us to compute an error bound between the two dynamics as well as a convergence rate. Finally, we show that the search for the reduced dynamics can be computed in polynomial time using a Schur decomposition of the transition matrix. This enables us to efficiently solve the reduced dynamical system using exponential of upper-triangular matrices. For CTMDPs, we generalise the approach to computing a piecewise quadratic Lyapunov functions for a switched affine dynamical system. We synthesise a policy for the CTMDP via its reduced-order switched system in order to have time bounded reachability probability above a threshold. We provide error bounds that depend on the minimum dwell time of the policy. We show the efficiency of the technique on examples from queueing networks, for which lumpability does not produce any state space reduction and which cannot be solved without reduction.

## 1 Introduction

Continuous-time Markov chains (CTMCs) and Markov decision processes (CTMDPs) play a central role in the modelling and analysis of performance and

dependability analysis of probabilistic systems evolving in real time. A CTMC combines probabilistic behaviour with real time: it defines a transition system on a set of states, where the transition between two states is delayed according to an exponential distribution. A CTMDP extends a CTMC by introducing non-deterministic choice among a set of possible actions. Both CTMCs and CTMDPs have been used in a large variety of applications — from biology to finance.

A fundamental problem in the analysis of CTMCs and CTMDPs is *time bounded reachability*: given a CTMC, a set of states, a time bound  $T$ , and a number  $\theta \in [0, 1]$ , it asks whether the probability of reaching the set of states within time  $T$  is at least  $\theta$ . In CTMDPs we are interested in finding a policy that resolves non-determinism for satisfying this requirement. Time bounded reachability is the core technical problem for model checking stochastic temporal logics such as Continuous Stochastic Logic [1, 3], and having efficient implementations of time bounded reachability is crucial to scaling formal analysis of CTMCs and CTMDPs.

Existing approaches to the time bounded reachability problem are based on discretisation or uniformisation, and in practice, are expensive computational procedures, especially as the time bound increases. The standard state-space reduction technique is probabilistic bisimulation [14]: a probabilistic bisimulation is an equivalence relation on the states that allows “lumping” together the equivalence classes without changing the value of time bounded reachability properties, or indeed of any CSL property [3]. Unfortunately, probabilistic bisimulation is a strong notion and small perturbations to the transition rates can change the relation drastically. Thus, in practice, it is often of limited use.

In this paper, we take a control-theoretic view to state space reductions of CTMCs and CTMDPs. Our starting point is that the forward Chapman-Kolmogorov equations characterising time bounded reachability define a linear dynamical system for CTMCs and a switched affine dynamical system for CTMDPs; moreover, one can transform the problem so that the dynamics is stable. Our first observation is a generalisation of probabilistic bisimulation to a quantitative setting. We show that probabilistic bisimulation can be viewed as a projection matrix that relates the original dynamical system with its bisimulation reduction. We then relax bisimulation to a quantitative notion, using a *generalised projection* operation between two linear systems.

**CTMCs.** The generalised projection does not maintain a linear relationship between the original and the reduced linear systems. However, our second result shows how the difference between the states of the two linear dynamical systems can be bounded as an exponentially decreasing function of time. The key to this result is finding an appropriate Lyapunov function on the difference between the two dynamics, which demonstrates an exponential convergence over time. We show that the search for a suitable Lyapunov function can be reduced to a system of matrix inequalities, which have a simple solution, and which leads to an error bound of the form  $L_0 e^{-\kappa t}$ , where  $L_0$  depends on the matrices defining the dynamics, and  $\kappa$  is related to the eigenvalues of the dynamics. Clearly, the error goes to zero exponentially as  $t \rightarrow \infty$ . Hence, by solving the reduced linear system, one can

approximate the time bounded reachability probability in the original system, with a bound on the error that converges to zero as a function of reachability horizon. For reducible CTMCs, we show that the same approach is applicable by preprocessing the structure of CTMC and eliminating those bottom strongly connected components that do not influence the reachability probability.

The Lyapunov approach suggests a systematic procedure to reduce the state space of a CTMC. If the original dynamical system has dimension  $n$ , we show, using Schur decomposition, that we can compute an  $m$ -dimensional linear system for each  $m \leq n$  as well as a Lyapunov-based bound on the error between the dynamics. Thus, for a given tolerance  $\varepsilon$ , one can iterate this procedure to find an appropriate  $m$ . This  $m$ -dimensional system can be solved using existing techniques, e.g., using exponential of upper-triangular matrices. The results in the literature (e.g. in [9]) rely on computing solutions of matrix inequalities, which is not scalable for dynamical systems of large dimension. In our paper, we characterized the required nonlinear matrix (in)equalities and provided a solution based on Schur decomposition of the generator matrices, which can be computed in polynomial-time.

**CTMDPs.** For CTMDPs, we generalise the approach for CTMCs using Lyapunov stability theorems for switched systems. Once again, the objective is to use multiple Lyapunov functions as a way to demonstrate stability, and derive an error bound from the multiple Lyapunov functions. For this we construct a piecewise quadratic Lyapunov function for a switched affine dynamical system. Then we synthesise a policy for the CTMDP via its reduced-order switched system in order to have time bounded reachability probability above a threshold. We provide error bounds that depend on the minimum dwell time of the policy.

The notion of *behavioral pseudometrics* on stochastic systems as a quantitative measure of dissimilarity between states have been studied extensively [2, 6], but mainly for discrete time Markov models and mostly for providing an upper bound on the difference between *all* formulas in a logic; by necessity, this makes the distance too pessimistic for a single property. In contrast, our approach considers a notion of distance for a specific time-bounded reachability property, and provides a time-varying error bound.

We have implemented our state space reduction approach and evaluated its performance on a queueing system benchmark. Fixing time horizon and error bound, our reduction algorithm computes a reduced order system, which takes less time to run. We show that, as the time horizon increases, we get significant reductions in the dimension of the linear system while providing strong bounds on the quality of the approximation.

## 2 Continuous-Time Markov Chains

**Definition 1.** A continuous-time Markov chain (CTMC)  $\mathcal{M} = (S_{\mathcal{M}}, R, \alpha)$  consists of a finite set  $S_{\mathcal{M}} = \{1, 2, \dots, |S_{\mathcal{M}}|\}$  of states, a rate matrix  $R: S_{\mathcal{M}} \times S_{\mathcal{M}} \rightarrow \mathbb{R}_{\geq 0}$ , and an initial probability distribution  $\alpha: S_{\mathcal{M}} \rightarrow [0, 1]$  satisfying  $\sum_{s \in S_{\mathcal{M}}} \alpha(s) = 1$ .

Intuitively,  $R(s, s') > 0$  indicates that a transition from  $s$  to  $s'$  is possible and that the timing of the transition is exponentially distributed with rate  $R(s, s')$ . If there are several states  $s'$  such that  $R(s, s') > 0$ , the chain can transition to any one of them in the following way. A state  $s \in S_{\mathcal{M}}$  is called *absorbing* if and only if  $R(s, s') = 0$  for all  $s' \in S_{\mathcal{M}}$ . Denote the total rate of taking an outgoing transition from state  $s$  by  $E(s) = \sum_{s' \in S_{\mathcal{M}}} R(s, s')$ . A transition from a non-absorbing state  $s$  into  $s'$  happens within time  $t$  with probability

$$\mathbf{P}(s, s', t) = \frac{R(s, s')}{E(s)} \cdot (1 - e^{-E(s)t}).$$

Intuitively,  $1 - e^{-E(s)t}$  is the probability of taking an outgoing transition at  $s$  within time  $t$  (exponentially distributed with rate  $E(s)$ ) and  $R(s, s')/E(s)$  is the probability of taking transition to  $s'$  among possible next states at  $s$ . Thus, the probability of moving from  $s$  to  $s'$  in one transition, written  $\mathbf{P}(s, s')$  is  $\frac{R(s, s')}{E(s)}$ . For an absorbing state, we have  $E(s) = 0$  and no transitions are enabled.

A right continuous step function  $\rho : \mathbb{R}_{\geq 0} \rightarrow S_{\mathcal{M}}$  is called an infinite path. For a given infinite path  $\rho$  and  $i \in \mathbb{N}$ , we denote by  $\rho_S[i]$  the state before the  $(i + 1)$ -th step, and by  $\rho_T[i]$  the time spent at  $\rho_S[i]$ , i.e., the length of the step segment starting with  $\rho_S[i]$ . Let  $\Pi_{\mathcal{M}}$  denote the set of all infinite paths, and  $\Pi_{\mathcal{M}}(s)$  denote the subset of those paths starting from  $s \in S_{\mathcal{M}}$ . Let  $I_0, \dots, I_{k-1}$  be nonempty intervals in  $\mathbb{R}_{\geq 0}$ . The cylinder set  $Cyl(s_0, I_0, s_1, I_1, \dots, s_{k-1}, I_{k-1}, s_k)$  is defined by:

$$\{\rho \in \Pi_{\mathcal{M}} \mid \forall 0 \leq i \leq k. \rho_S[i] = s_i \wedge \forall 0 \leq i < k. \rho_T[i] \in I_i\}.$$

Let  $\mathcal{F}(\Pi_{\mathcal{M}})$  denote the smallest  $\sigma$ -algebra on  $\Pi_{\mathcal{M}}$  containing all cylinder sets. The probability measure  $\text{Prob}_{\alpha}$  on  $\mathcal{F}(\Pi_{\mathcal{M}})$  is the unique measure defined by induction on  $k$  with  $\text{Prob}_{\alpha}(Cyl(s_0)) := \alpha(s_0)$  and

$$\begin{aligned} \text{Prob}_{\alpha}(Cyl(s_0, I_0, \dots, s_k, [a, b], s')) := \\ \text{Prob}_{\alpha}(Cyl(s_0, I_0, \dots, s_k)) \cdot \mathbf{P}(s_k, s')(e^{-E(s_k)a} - e^{-E(s_k)b}). \end{aligned}$$

The *transient state probability*, written  $\bar{\pi}_{\alpha}^{\mathcal{M}}(t)$ , is defined as a row vector with elements  $\text{Prob}_{\alpha}\{\rho \mid \rho(t) = s'\}$ ,  $s' \in S_{\mathcal{M}}$ . The transient probabilities of  $\mathcal{M}$  are characterised by the forward Chapman-Kolmogorov differential equation [4], which is the system of linear differential equations

$$\frac{d}{dt} \bar{\pi}_{\alpha}^{\mathcal{M}}(t) = \bar{\pi}_{\alpha}^{\mathcal{M}}(t) \bar{\mathbf{Q}}, \quad \bar{\pi}_{\alpha}^{\mathcal{M}}(0) = \alpha. \tag{1}$$

where  $\bar{\mathbf{Q}}$  is the *infinitesimal generator* matrix of  $\mathcal{M}$  defined as  $\bar{\mathbf{Q}} = R - \text{diag}_s(E(s))$ . Note that  $\sum_{s'} \bar{\mathbf{Q}}(s, s') = 0$  for any  $s \in S_{\mathcal{M}}$ .  $\pi_s^{\mathcal{M}}(t)(s')$  indicates the probability that  $\mathcal{M}$  starts at initial state  $s$  and is at state  $s'$  at time  $t$ . Therefore,

$$\frac{d}{dt} \bar{\pi}_s^{\mathcal{M}}(t) = \bar{\pi}_s^{\mathcal{M}}(t) \bar{\mathbf{Q}}, \quad \bar{\pi}_s^{\mathcal{M}}(0) = \mathbf{1}(s) \tag{2}$$



where  $\bar{\pi}_s^{\mathcal{M}}(t) \in \mathbb{R}^{S_{\mathcal{M}}}$  is a row vector containing transient state probabilities ranging over all states in  $S_{\mathcal{M}}$ . We equate the row vector with  $n$  co-ordinates with a function from  $S$  to reals. The initial value of differential equation (2) is a vector indicating the initial probability distribution that assigns the entire probability mass to the state  $s$ , that is,  $\mathcal{M}: \bar{\pi}^{\mathcal{M}}(s, 0) = \mathbf{1}(s)$ , a vector whose element associated to  $s$  is one, and zero otherwise.

Let  $\mathcal{M} = (S \uplus \{\mathbf{good}, \mathbf{bad}\}, R, \alpha)$  be a CTMC with two absorbing states **good** and **bad**, where,  $|S| = n$  and let  $T \in \mathbb{R}_{\geq 0}$  be a time bound. We write  $\text{Prob}^{\mathcal{M}}(\mathbf{1}(s), T) = \bar{\pi}_s^{\mathcal{M}}(T)(\mathbf{good})$ . The *time-bounded reachability problem* asks to compute this probability. Note that, for all  $T$ ,  $\text{Prob}^{\mathcal{M}}(\mathbf{1}(\mathbf{good}), T) = 1$  and  $\text{Prob}^{\mathcal{M}}(\mathbf{1}(\mathbf{bad}), T) = 0$ . In general, we are interested in finding the probability for a given subset  $S_0 \subseteq S$  of states. We denote solution to this problem as a  $n_0 \times 1$  vector  $\text{Prob}^{\mathcal{M}}(C, T)$ , where  $C$  is a  $n_0 \times (n + 2)$  matrix with  $n_0 = |S_0|$  ones on its main diagonal, corresponding to the states in  $S_0$ . If  $S_0 = S_{\mathcal{M}}$ , then  $C$  is the  $(n + 2) \times (n + 2)$  identity matrix.

### 3 Time-Bounded Reachability on CTMCs

#### 3.1 From Reachability to Linear Dynamical Systems

Let  $\mathcal{M} = (S \uplus \{\mathbf{good}, \mathbf{bad}\}, R, \alpha)$  be a CTMC, with  $|S| = n$ , and absorbing states **good** and **bad**. The solution to the time-bounded reachability problem for a projection matrix  $C$  can be obtained by rewriting (2) as:

$$\begin{cases} \frac{d}{dt}Z(t) = QZ(t), & Z(0) = \mathbf{1}(\mathbf{good}), \\ \text{Prob}^{\mathcal{M}}(C, t) = CZ(t) \end{cases} \tag{3}$$

where  $Z(t) \in \mathbb{R}^{n+2}$  is a column vector with elements  $Z_i(t) = \text{Prob}^{\mathcal{M}}(\mathbf{1}(s_i), t)$ . Notice that in this formulation, we have let time “run backward”: we start with an initial vector which is zero except for corresponding element to the state **good** and compute “backward” up to the time  $T$ . By reordering states, if necessary, the generator matrix  $Q$  in (3) can be written as:

$$Q = \begin{bmatrix} A & \vdots & \chi & \vdots & \beta \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \vdots & \mathbf{0} & \vdots & \mathbf{0} \end{bmatrix} \tag{4}$$

with  $A \in \mathbb{R}^{n \times n}$ ,  $\chi \in \mathbb{R}^{n \times 1}$ , and  $\beta \in \mathbb{R}^{n \times 1}$ . Vectors  $\chi$  and  $\beta$  contain the rates of jumping to the states **bad** and **good**, respectively. With this reordering of the states, it is obvious that in (3),  $Z(t)(\mathbf{bad}) = 0$  and  $Z(t)(\mathbf{good}) = 1$ , thus we assume states **good** and **bad** are not included in  $C$ . We write  $Z_S(t)$  for the vector (in  $\mathbb{R}^n$ ) restricting  $Z$  to states in  $S$ . These variables should satisfy

$$\begin{cases} \frac{d}{dt}Z_S(t) = AZ_S(t) + \beta, & Z_S(0) = 0 \\ \text{Prob}^{\mathcal{M}}(C_S, t) = C_S Z_S(t), \end{cases} \tag{5}$$

where  $C_S \in \mathbb{R}^{n_0 \times n}$  is the matrix obtained by omitting the last two columns of  $C$ .

Equation (5) can be seen as model of a linear dynamical system with unit input. Our aim here is to find solution of (5) using reduction techniques from control theory while providing guarantees on the accuracy of the computation and to interpret the solution as the probability for time bounded reachability.

Let  $\gamma := \max_{i=1:n} |a_{ii}|$ , the maximal diagonal element of  $A$ , and define matrix  $H$  as  $H = \frac{A}{\gamma} + \mathbb{I}_n$ , where  $\mathbb{I}_n$  is the  $n \times n$  identity matrix. We fix the following assumption.

**Assumption 1.**  *$H$  is an irreducible matrix, i.e., its associated directed graph is strongly connected. Moreover,  $\beta + \alpha \neq 0$ . That is, either **good** or **bad** is reachable from some state in  $S$ .*

*Remark 1.* The above assumption is “WLOG.” First, if there is no edge from  $S$  to **good** or **bad**, the problem is trivial. Second, the general case, when  $H$  is not irreducible can be reduced to the assumption in polynomial time. Thus, the assumption restricts attention to the core technical problem.

Recall that a matrix  $A$  is stable if every eigenvalue of  $A$  has negative real part. The spectral radius of a matrix is the largest absolute value of its eigenvalues. Assumption 1 will imply the following proposition.

**Proposition 1.** *Assumption 1 implies that matrix  $A$  is invertible and stable.*

Since the input to (5) is fixed, we try to transform it to a set of differential equations without input but with initial value. Let us take a transformation that translates  $Z_S(t)$  by the offset vector  $A^{-1}\beta$ :

$$X(t) := Z_S(t) + A^{-1}\beta. \tag{6}$$

The evolution of  $X(\cdot)$  is:

$$\begin{cases} \frac{d}{dt}X(t) = AX(t), & X(0) = A^{-1}\beta \\ \text{Prob}^{\mathcal{M}}(C_S, t) = C_S X(t) + d. \end{cases} \tag{7}$$

where  $d = -C_S A^{-1}\beta$ . The *dimension* (number of variables) of dynamical system (7) is  $n$ , the size of the state space  $S$ .

*Remark 2.* Under Assumption 1, the solution of infinite horizon reachability problem is  $-A^{-1}\beta$ , which can be computed efficiently as the solution of a system of linear equations. Elements of  $X(t)$  defined in (6) contains the values of finite-horizon reachability in compare with the infinite-horizon values.

In the following, we show how the solution of this dynamical system can be approximated by a dynamical system of lower dimension. Our approach relies on stability property of matrix  $A$ , and gives an upper bound on the approximation error that converges exponentially to zero as a function of time. Thus our approach is beneficial for long time horizons when previous techniques fail to provide tight bounds.

### 3.2 Bisimulation and Projections

Probabilistic bisimulation or lumpability is a classical technique to reduce the size of the state space of a CTMC. For CTMC  $\mathcal{M} = (S_{\mathcal{M}}, R, \alpha)$  with space  $S_{\mathcal{M}} = S \uplus \{\mathbf{good}, \mathbf{bad}\}$ , a bisimulation on  $\mathcal{M}$  is an equivalence relation  $\cong$  on  $S_{\mathcal{M}}$  such that **good** and **bad** are singleton equivalence classes and for any two states  $s_1, s_2 \in S$ ,  $s_1 \cong s_2$  implies  $R(s_1, \Theta) = R(s_2, \Theta)$  for every equivalence class  $\Theta$  of  $\cong$ , where  $R(s, \Theta) := \sum_{s' \in \Theta} R(s, s')$ . Given a bisimulation relation  $\cong$  on  $\mathcal{M}$ , we can construct a CTMC  $\bar{\mathcal{M}} = (S_{\bar{\mathcal{M}}}, \bar{R}, \bar{\alpha})$  of smaller size such that probabilities are preserved over paths of  $\mathcal{M}$  and  $\bar{\mathcal{M}}$ . In particular,  $s_1 \cong s_2$ , implies that

$$\text{Prob}^{\mathcal{M}}(\mathbf{1}(s_1), t) = \text{Prob}^{\bar{\mathcal{M}}}(\mathbf{1}(s_2), t), \quad \forall t \in \mathbb{R}_{\geq 0}.$$

The CTMC  $\bar{\mathcal{M}}$  has the quotient state space  $\{[s]_{\cong} \mid s \in S\} \uplus \{\mathbf{good}, \mathbf{bad}\}$ , where  $[s]_{\cong}$  is the equivalence class of  $s \in S$ , rate function  $\bar{R}([s]_{\cong}, \Theta) = R(s, \Theta)$  for any  $\Theta \in S_{\bar{\mathcal{M}}}$ , and initial distribution  $\bar{\alpha}([s]_{\cong}) = \sum_{s' \in [s]_{\cong}} \alpha(s')$ .

We now show how the differential equation (7) for  $\mathcal{M}$  and  $\bar{\mathcal{M}}$  relate. Assume that the state space of  $\bar{\mathcal{M}}$  is  $\bar{S} \cup \{\mathbf{good}, \mathbf{bad}\}$ , where  $|\bar{S}| = m$ . We have

$$\begin{cases} \frac{d}{dt} \bar{X}(t) = \bar{A} \bar{X}(t), & \bar{X}(0) = \bar{A}^{-1} \bar{\beta}, \\ \text{Prob}^{\bar{\mathcal{M}}}(\bar{C}_S, t) = \bar{d} + \bar{C}_S \bar{X}(t). \end{cases} \tag{8}$$

where  $\bar{A}$  and  $\bar{\beta}$  are computed similarly to that of  $\mathcal{M}$  according to the generator matrix of  $\bar{\mathcal{M}}$ . Note that  $\bar{A}$  is an  $m \times m$  matrix. Matrix  $\bar{C}_S$  is  $n_0 \times m$  constructed according to  $S_0$ , with  $|S_0|$  ones corresponding to the quotient states  $\{[s]_{\cong} \mid s \in S_0\}$ . We now define a *projection matrix*  $P_{\cong} \in \mathbb{R}^{n \times m}$  as  $P_{\cong}(i, j) = 1$  if  $s_i \in [j]$ , i.e.,  $s_i$  belongs to the equivalence class  $[j] \in \bar{S}$ , and zero otherwise. This projection satisfies  $C_S P_{\cong} = \bar{C}_S$ , and together with the definition of  $\cong$  implies the following proposition.

**Proposition 2.** *For every bisimulation  $\cong$ , The projection matrix  $P_{\cong}$  satisfies the following*

$$AP_{\cong} = P_{\cong} \bar{A}, \quad \beta = P_{\cong} \bar{\beta}. \tag{9}$$

*Conversely, every projection matrix satisfying (9) defines a bisimulation relation. In particular,*

$$X(t) = P_{\cong} \bar{X}(t), \quad \forall t \in \mathbb{R}_{\geq 0}. \tag{10}$$

*Example 1.* As an example, consider the CTMC in Fig. 1 and assume first that all  $\epsilon_{ij} = 0$ . We have omitted state **bad** which is unreachable. We are interested in measuring probability of reaching state **good**, which is made absorbing by removing its outgoing links. It is easy to see that the bisimulation classes are  $\{s_1, s_2\}$ ,  $\{s_3, s_4\}$ , and  $\{\mathbf{good}\}$ . The bisimulation reduction and the corresponding projection matrix  $P_{\cong}$  are shown on the right-hand side. The differential equation for the reduced CTMC has dimension 2.

Unfortunately, as is well known, bisimulation is a strong condition, and small perturbations in the rates can cause two states to not be bisimilar. Consider a

perturbed version of the CTMC with  $Q$  as generator matrix, given below. Here,  $\varepsilon \neq 0$  for some links, and the CTMC on the right-hand side of Fig. 1 is not a bisimulation reduction. Let us also consider a perturbed version of the CTMC on the right-hand side of Fig. 1 with the following generator matrix and updated projection matrix:

$$Q = \begin{bmatrix} -2.95 & 0 & 0.95 & 0 & 2 \\ 0 & -3.05 & 1.05 & 0 & 2 \\ 0 & 1.5 & -1.5 & 0 & 0 \\ 0 & 1.5 & 0 & -1.5 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, Q_r = \begin{bmatrix} -3.05 & 1.05 & 2 \\ 1.5 & -1.5 & 0 \\ 0 & 0 & 0 \end{bmatrix}, P = \begin{bmatrix} .8285 & .0552 & .1162 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Clearly these two perturbed CTMCs are not bisimilar according to the usual definition of bisimulation relation, but the above choices satisfies the equality  $QP = PQ_r$ . Note that  $P$  is no longer a projection matrix, but has entries in  $[0, 1]$ , which sum up to 1 for each row. This particular  $P$  satisfies  $AP = P\bar{A}$  but not  $\beta = P\bar{\beta}$  (see (9)). Thus the original dynamics of  $X(t)$  and their lower-dimensional version  $\bar{X}(t)$ , reduced with  $P$ , do not satisfy the equality (10).

However, since  $A$  is a stable matrix, we expect the trajectories of the original and the reduced dynamics to converge, that is, the error between the trajectories to go to zero as time goes to infinity. In the next section, we generalise projection matrices as above, and formalise this intuition.

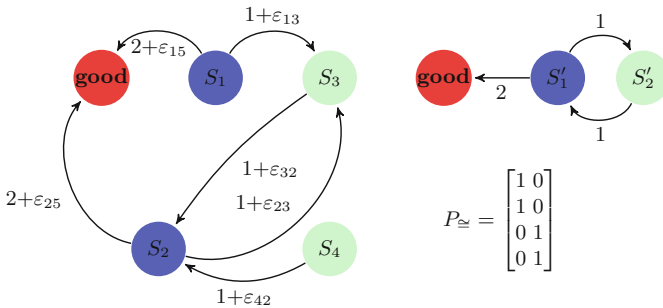


Fig. 1. Full state  $\varepsilon$  perturbed CTMC (left), reduced order CTMC (right), and projection matrix (right, below) computed for the perturbation free case.

### 3.3 Generalised Projections and Reduction

Suppose we are given CTMCs  $\mathcal{M}$  and  $\bar{\mathcal{M}}$ , with corresponding dynamical systems (7) and (8), and a matrix  $P$  with entries in  $[0, 1]$  whose rows add up to 1, such that  $AP = P\bar{A}$ . We call such a  $P$  a *generalised projection*. Define vector  $\bar{C}_S = C_S P$ . In general, the equality  $\beta = P\bar{\beta}$  does not hold for generalised projections. In the following we provide a method based on Lyapunov stability theory to quantify an upper bound  $\varepsilon(t)$  such that

$$\left| Prob^{\mathcal{M}}(C_S, t) - Prob^{\bar{\mathcal{M}}}(\bar{C}_S, t) \right| \leq \varepsilon(t), \tag{11}$$

for all  $t \geq 0$ , where  $\varepsilon(t)$  depends linearly on the mismatch  $\beta - P\bar{\beta}$  and decays exponentially with  $t$ .

First, we recall some basic results for linear dynamical systems (see, e.g., [7])

$$\frac{d}{dt}Y(t) = AY(t), \quad Y(t) \in \mathbb{R}^n, \quad Y(0) = Y_0. \tag{12}$$

We call the system stable if  $A$  is a stable matrix.<sup>1</sup> A continuous scalar function  $V : \mathbb{R}^n \rightarrow \mathbb{R}$  is called a *Lyapunov function* for dynamical system (12) if  $V(0) = 0$ ;  $V(y) > 0$  for all  $y \in \mathbb{R}^n \setminus \{0\}$ ; and  $dV(Y(t))/dt < 0$  along trajectories of the dynamical system  $Y(t) \neq 0$ .

A matrix  $M \in \mathbb{R}^{n \times n}$  is symmetric if  $M^T = M$ . A symmetric matrix  $M$  satisfying the condition  $Y^TMY > 0$  for all  $Y \in \mathbb{R}^n \setminus \{0\}$  is called *positive definite*, and written  $M \succ 0$ . Any symmetric matrix  $M$  satisfying  $Y^TMY \geq 0$  for all  $Y \in \mathbb{R}^n$  is called *positive semi-definite*, written  $M \succeq 0$ . Similarly, we can define *negative definite* matrices  $M \prec 0$  and *negative semi-definite* matrices  $M \preceq 0$ . The following is standard.

**Theorem 1.** [13] *Linear dynamical system (12) is stable iff there exists a quadratic Lyapunov function  $V(Y) = Y^TMY$  such that  $M \succ 0$  and  $A^TM + MA \prec 0$ . Moreover, for any constant  $\kappa > 0$  such that  $A^TM + MA + 2\kappa M \preceq 0$ , we have*

$$\|Y(t)\|_2 \leq Le^{-\kappa t}\|Y_0\|_2, \quad \forall Y_0 \in \mathbb{R}^n, \forall t \in \mathbb{R}_{\geq 0},$$

for some constant  $L \geq 0$ , where  $\|\cdot\|_2$  indicates the two-norm of a vector.

Note that in our setting, we are not interested in the study of asymptotic stability of systems, but we are given two dynamical systems (7) and (8), and we would like to know how close their trajectories are as a function of time. In this way we can use one of them as an approximation of the other one with guaranteed error bounds. For this reason, we define Lyapunov function  $V : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$  of the form

$$V(X, \bar{X}) = (X - P\bar{X})^T M(X - P\bar{X}), \tag{13}$$

where  $M \succ 0$  is a positive definite matrix. The value of  $V(X(t), \bar{X}(t))$  at  $t = 0$  can be calculated as

$$\begin{aligned} V(X(0), \bar{X}(0)) &= (A^{-1}\beta - P\bar{A}^{-1}\bar{\beta})^T M(A^{-1}\beta - P\bar{A}^{-1}\bar{\beta}) \\ &= (\beta - P\bar{\beta})^T A^{-1T} M A^{-1} (\beta - P\bar{\beta}). \end{aligned} \tag{14}$$

Using Lyapunov function (13) we can establish the following theorem.

**Theorem 2.** *Consider dynamical systems (7) and (8) with invertible matrix  $A$ , and let  $P$  be a generalised projection satisfying  $AP = P\bar{A}$ . If there exist matrix  $M$  and constant  $\kappa > 0$  satisfying the following set of matrix inequalities:*

$$\begin{cases} M \succ 0 \\ C_S^T C_S \preceq M \\ MA + A^T M + 2\kappa M \preceq 0, \end{cases} \tag{15}$$

<sup>1</sup> In this case, it is known that  $\lim_{t \rightarrow \infty} Y(t) = 0$  for any initial state  $Y_0 \in \mathbb{R}^n$ .

then we have  $|\text{Prob}^M(C_S, t) - \text{Prob}^{\bar{M}}(\bar{C}_S, t)| \leq \varepsilon(t)$ , for all  $t \geq 0$ , with

$$\varepsilon(t) = \xi \| \Gamma \| e^{-\kappa t}, \tag{16}$$

where  $\Gamma := \beta - P\bar{\beta}$  is the mismatch induced by the generalised membership functions and  $\xi^2 = \lambda_{\max}(A^{-1T}MA^{-1})$ .

The error in (16) is exponentially decaying with decay factor  $\kappa$  and increases linearly with mismatch  $\Gamma$ . Matrix inequalities (15) in Theorem 2 are bilinear in terms of unknowns (entries of  $M$  and constant  $\kappa$ ) due to the multiplication between  $\kappa$  and  $M$ , thus difficult to solve. In the following we show how to obtain a solution efficiently.

**Theorem 3.** *Under the conditions of Theorem 2, suppose additionally that  $A$  is stable. Then there exists  $M$  and  $\kappa$  such that (15) is satisfied. Further, for each  $r \leq n$ , there is an  $n \times r$  matrix  $P$  and an  $r \times r$  matrix  $\bar{A}$ , computable in polynomial time in  $n$ , such that  $AP = P\bar{A}$ .*

Once  $\kappa$  is fixed, constraints (15) become matrix inequalities that are linear in terms of entries of  $M$  and can be solved using convex optimisation [8] and developed tools for linear matrix inequalities [11, 15]. In particular,  $M = \mathbb{I}_n$  is a valid solution to the LMI. However, when  $C_S$  is not full rank, which is the case when  $S_0 \neq S$ , solving the LMI for  $M$  can result in better error bounds.

Notice that  $V(0) = (X(0) - P\bar{X}(0))^T M (X(0) - P\bar{X}(0))$  and using (7), we have  $X(0) = A^{-1}\beta$ . Therefore, it is important to find  $\bar{X}(0)$  that results in the least  $V(0)$ . We can compute  $\bar{X}(0)$  taking  $M$  into account:

$$\bar{X}(0) = (P^T M P)^{-1} P^T M (A^{-1}\beta), \tag{17}$$

which provides a tighter initial error bound. Knowing  $\bar{A}$  and  $\bar{X}(0)$ , one can find  $\bar{\beta} = \bar{A}\bar{X}(0)$ .

Theorem 3 gives an algorithm to find lower dimensional approximations to the dynamical system (7) and Theorem 2 provides a quantitative error bound for the approximation. Given a time-bounded reachability problem and an error bound  $\varepsilon$ , we iteratively compute reduced order dynamical systems of dimension  $r = 1, \dots, n-1$  using Theorem 3. Then, we check if the error bound in Theorem 3 is at most  $\varepsilon$ . If so, we solve the dynamical system of dimension  $m$  (using, e.g., exponential of an upper-triangular matrix) to compute an  $\varepsilon$ -approximation to the time bounded reachability problem. If not, we increase  $r$  and search again.

## 4 Time-Bounded Reachability on CTMDPs

First, we define continuous-time Markov decision processes (CTMDPs) that include non-deterministic choice of actions on top of probabilistic jumps.

**Definition 2.** *A continuous-time Markov decision process (CTMDP)  $\mathcal{N} = (S_{\mathcal{N}}, \mathcal{D}, R_d)$  consists of a finite set  $S_{\mathcal{N}} = \{1, 2, \dots, |S_{\mathcal{N}}|\}$  of states, a finite set of possible actions  $\mathcal{D}$ , and action-dependent rate matrices  $R_d$ , where  $d \in \mathcal{D}^{|S_{\mathcal{N}}|}$  is a decision vector containing actions taken at different states,  $d := \{d(s) \mid s \in S_{\mathcal{N}}\}$ .*

Note that some of the actions may not be available at all states. Denote the set of possible decision vectors by  $\mathbf{D} \subseteq \mathcal{D}^{|S_{\mathcal{N}}|}$ . Similar to CTMCs, we assign an initial distribution  $\alpha$  to the CTMDP  $\mathcal{N}$ . For any fixed  $d \in \mathbf{D}$ ,  $\mathcal{N}_d = (S_{\mathcal{N}}, R_d, \alpha)$  forms a CTMC, for which we can define infinitesimal generator  $\mathbf{Q}_d := R_d - \text{diag}_s(E_d(s))$  with total exit rates at state  $s$ ,  $E_d(s) := \sum_{s' \in S_{\mathcal{N}}} R_d(s, s')$ .

Path  $\omega$  of a CTMDP  $\mathcal{N}$  is a (possibly infinite) sequence including transitions of the form  $s_i \xrightarrow{d_i, t_i} s_{i+1}$ , for  $i = 0, 1, 2, \dots$ , where  $t_i \in \mathbb{R}_{\geq 0}$  is the sojourn time in  $s_i$  and  $d_i \in \mathcal{D}$  is a possible action taken at  $s_i$ . A policy provides a mapping from the paths to actions of the model, in order to resolve the nondeterminism that occurs in the states of a CTMDP for which more than one action is possible.

Let  $\mathcal{N} = (S \uplus \{\mathbf{good}, \mathbf{bad}\}, \mathcal{D}, R_d)$  be a CTMDP with two absorbing states **good** and **bad**, where,  $|S| = n$  and let  $T \in \mathbb{R}_{\geq 0}$  be a time bound and  $\theta \in (0, 1)$  a probability threshold. We are interested in synthesising a policy  $\pi$  such that probability of reaching state **good** and avoiding state **bad** within time interval  $[0, T]$  is at least  $\theta$ :

$$\text{Prob}^{\mathcal{N}(\pi)}(\mathbf{1}(s), T) = \bar{\pi}_s^{\mathcal{N}(\pi)}(T)(\mathbf{good}) \geq \theta, \tag{18}$$

where  $\text{Prob}^{\mathcal{N}(\pi)}$  is the probability measure induced on paths of  $\mathcal{N}$  by resolving non-determinism via policy  $\pi$ . Synthesising such a policy can be done by maximising the left-hand side of (18) on the set of policies and then compare the optimal value with  $\theta$ . Such an optimal policy is shown to be in the class of time-dependent Markov policies and can be characterised as follows [5]. We partition any generator matrix  $Q_d$  corresponding to decision vector  $d \in \mathbf{D}$ , as

$$Q_d = \begin{bmatrix} A_d & \vdots & \chi_d & \vdots & \beta_d \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \vdots & \mathbf{0} & \vdots & \mathbf{0} \end{bmatrix} \tag{19}$$

with  $A_d \in \mathbb{R}^{n \times n}$ ,  $\chi_d \in \mathbb{R}^{n \times 1}$ , and  $\beta_d \in \mathbb{R}^{n \times 1}$ . Then for a CTMDP  $\mathcal{N}$  with matrix  $C$  indicating a subset of initial states  $S_0 \subseteq S$  for which we would like to satisfy (18),  $\max_{\pi} \text{Prob}^{\mathcal{N}(\pi)}(C, T)$  can be characterised backward in time as the solution of the following set of nonlinear differential equations

$$\begin{cases} \frac{d}{dt} W(t) = \max_{d(t) \in \mathbf{D}} Q_{d(t)} W(t), & W(0) = \mathbf{1}(\mathbf{good}), \\ \max_{\pi} \text{Prob}^{\mathcal{N}(\pi)}(C, T) = C W(t), \end{cases} \tag{20}$$

where  $W(t)$  is a column vector containing probabilities  $\max_{\pi} \text{Prob}^{\mathcal{N}(\pi)}(\mathbf{1}(s), T)$  as a function of initial state  $s$ .

With respect to the partitioning (19), it is obvious that in (20),  $W(t)(\mathbf{bad}) = 0$  and  $W(t)(\mathbf{good}) = 1$  for all  $t \in \mathbb{R}_{\geq 0}$ . The remaining state variables  $W_S(t)$  should satisfy

$$\begin{cases} \frac{d}{dt} W_S(t) = \max_{d(t) \in \mathbf{D}} (A_{d(t)} W_S(t) + \beta_{d(t)}), & W_S(0) = 0, \\ \max_{\pi} \text{Prob}^{\mathcal{N}(\pi)}(C_S, t) = C_S W_S(t). \end{cases} \tag{21}$$

The optimal policy is the one maximising the right-hand side of differential equation in (21),  $\pi^* = \{d(t) \in \mathbf{D} \mid t \in \mathbb{R}_{\geq 0}\}$ , thus it is time-dependent and is only a function of state of the CTMDP at time  $t$ . Finding the optimal policy is computationally expensive particularly for CTMDPs with large number of states. For instance [16] has proposed an approach based on breaking time interval  $[0, T]$  into smaller intervals of length  $\delta$ , and then computing (approximate) optimal decisions in each interval of length  $\frac{T}{\delta}$  sequentially. Thus, a set of linear differential equations must be solved in each interval, which is computationally expensive.

In the following, we will develop a new way of synthesising a policy that satisfies (18) by approximating the solution of (21) via generalised projections and reductions. We treat (21) as a *switched affine system* [10]. We are given a collection of  $|\mathbf{D}|$  affine dynamical systems, characterised by the pairs  $(A_d, \beta_d)$ , and the role of any policy  $\pi = \{d(t) \in \mathbf{D}, t \geq 0\}$  is to switch from one dynamical system to another by picking a different pair. The main underlying idea of our approximate computation is to consider the reduced order version of these dynamical systems and find a switching policy  $\pi$ . We provide guarantees on the closeness to the exact reachability probability when this policy is applied to the original CTMDP. For this we require the following assumption.

**Assumption 2.** *Matrices  $\{A_d, d \in \mathbf{D}\}$  are all stable.*

Note that this assumption is satisfied if for each setting of actions, the resulting CTMC is irreducible (Proposition 1) and the time-bounded reachability problem does not have a trivial solution.

Under Assumption 2, we can find matrix  $M_d$  and constant  $\kappa_d > 0$ , for any  $d \in \mathbf{D}$ , such that the following matrix inequalities hold:

$$\begin{cases} M_d \succ 0 \\ C_S^T C_S \preceq M_d \\ M_d A_d + A_d^T M_d + 2\kappa_d M_d \preceq 0, \end{cases} \tag{22}$$

We need the following lemma that gives us a bound on the solution of reduced order systems.

**Lemma 1.** *Suppose generalised projections  $P_d$  and matrices  $\bar{A}_d$  satisfy  $A_d P_d = P_d \bar{A}_d$  for any  $d \in \mathbf{D}$ . Then  $V(\bar{X}_d) = \bar{X}_d^T \bar{M}_d \bar{X}_d$  with  $\bar{M}_d = P_d^T M_d P_d$  and  $M_d$  satisfying (22), is a Lyapunov function for  $d\bar{X}_d(t)/dt = \bar{A}_d \bar{X}_d(t)$ . Moreover,*

$$\|\bar{X}_d(t_1)\|_{\bar{M}_d} \leq \|\bar{X}_d(t_0)\|_{\bar{M}_d} e^{-\kappa_d(t_1-t_0)}, \quad \forall t_1 \geq t_0, \tag{23}$$

where  $\|Y\|_G := \sqrt{Y^T G Y}$  is the weighted two-norm of a vector  $Y$ .

Consider an arbitrary time-dependent Markov policy  $\pi = \{d(t) \in \mathbf{D}, t \geq 0\}$ . Then there is a sequence of decision vectors  $(d_0, d_1, d_2, \dots)$  with switching times  $(t_0, t_1, t_2, \dots)$  such that actions in  $d_i$  are selected over time interval  $[t_{i-1}, t_i]$  depending on the state of  $\mathcal{N}$ , for any  $i = 0, 1, 2, \dots$  with  $t_{-1} = 0$ . We first study



time-bounded reachability for  $\mathcal{N}$  under policy  $\pi$ , which can be characterised as the switched system:

$$\frac{d}{dt}W_S(t) = A_{d_i}W_S(t) + \beta_{d_i}, \forall t \in [t_{i-1}, t_i), i = 0, 1, \dots \quad (24)$$

Similar to our discussion on CTMC, we prefer to move constant inputs  $\beta_{d_i}$  in (24) into initial states. Therefore, we define the following piecewise translation

$$X(t) := W_S(t) + A_{d_i}^{-1}\beta_{d_i}, \forall t \in [t_{i-1}, t_i), i = 0, 1, 2, \dots \quad (25)$$

that depends also on  $\pi$ . Thus the evolution of  $X(t)$  becomes

$$\frac{d}{dt}X(t) = A_{d_i}X(t), \forall t \in [t_{i-1}, t_i), i = 0, 1, 2, \dots, \quad (26)$$

with jumps happening at switching times

$$\Delta X(t_i) := X(t_i) - X(t_i^-) = A_{d_{i+1}}^{-1}\beta_{d_{i+1}} - A_{d_i}^{-1}\beta_{d_i}. \quad (27)$$

This quantity is exactly the difference between unbounded reachability probability if one of decision vectors  $d_i$  and  $d_{i+1}$  is taken independent of time. Similarly, we define

$$\Delta_{ij} := A_{d_j}^{-1}\beta_{d_j} - A_{d_i}^{-1}\beta_{d_i}, \quad (28)$$

which will be used later in Theorem 4. Now we construct the reduced order switched system

$$\frac{d}{dt}\bar{X}(t) = \bar{A}_{d_i}\bar{X}(t), \forall t \in [t_{i-1}, t_i), i = 0, 1, 2, \dots, \quad (29)$$

with  $\bar{A}_d$  satisfying  $A_d P_d = P_d \bar{A}_d$  for all  $d \in \mathbf{D}$ . We choose the values of jumps  $\Delta \bar{X}(t_i) := \bar{X}(t_i) - \bar{X}(t_i^-)$  so that the behaviour of (29) is as close as possible to (26). For this, we have

$$\bar{X}(t_i) := \arg \min_{\bar{X}} \|\Delta X(t_i) - P_{d_{i+1}}\bar{X} + P_{d_i}\bar{X}(t_i^-)\|_{M_{d_{i+1}}}, \quad (30)$$

which can be computed for any value of  $\bar{X}(t_i^-)$ .

Define the *dwell time* of a policy  $\pi$  by  $\tau = \min_i(t_i - t_{i-1})$ , i.e., the minimum time between two consecutive switches of decision vectors in  $\pi$ . Next theorem quantifies the error between the two switched system using dwell time.

**Theorem 4.** *Given a CTMDP  $\mathcal{N}$ , a policy  $\pi$ , and bounded-time reachability over  $[0, T]$  characterised via (26). Suppose there exist  $M_d, \kappa_d$  satisfying (22), constant  $\mu$  satisfies  $M_d \preceq \mu M_{d'}$  for all  $d, d' \in \mathbf{D}$ , and matrices  $\bar{A}_d, P_d$  are computed such that  $A_d P_d = P_d \bar{A}_d$ . Then it holds that*

$$\|X(T) - P_{d_{n+1}}\bar{X}(T)\|_{M_{d_{n+1}}} \leq \left[ \frac{\eta(1 - g^n)}{1 - g} + g^n \varepsilon_0 \right] e^{-\kappa(T - t_n)}, \quad (31)$$

where  $g := \mu e^{-\kappa\tau}$  with  $\kappa = \min_d \kappa_d$  being the minimum decay rate and  $\tau$  dwell time of  $\pi$ . Constant  $\eta = \max \eta_{ij}$  with

$$\eta_{ij} := \min_{\bar{X}_j} \max_{\bar{X}_i} \|\Delta_{ij} + P_{d_i} \bar{X}_i - P_{d_j} \bar{X}_j\|_{M_{d_j}}, \tag{32}$$

and  $\Delta_{ij}$  defined in (28). Finally,  $\varepsilon_0 := \|A_{d_0}^{-1} \beta_{d_0} - P_{d_0} \bar{X}(0)\|_{M_{d_0}}$ ,  $\bar{X}(0)$  is computed using weighted least square method similar to (17), and  $t_n$  is the last switching time before time bound  $T$ .

Note that  $\eta_{ij}$ 's in Theorem 4 are well-defined due to the fact that  $\bar{X}_i, \bar{X}_j$  are states of the reduced order systems and are bounded as stated in Lemma 1.

*Remark 3.* (1) The precision of the bound in (31) can be increased in two ways. First, the bound will be lower (smaller  $g$ ) for policies with larger dwell time  $\tau$ . Second, if we increase the order of reduced system,  $\eta$  and  $\varepsilon_0$  will become smaller.

(2) The gain  $g$  solely depends on the CTMDP  $\mathcal{N}$  and dwell time of policy  $\pi$ . In order to have a meaningful error bound, dwell time should satisfy  $\tau > \frac{\log \mu}{\kappa}$ . This condition is already true if we find a *common Lyapunov function* for the CTMDP  $\mathcal{N}$ , i.e., if there is one matrix  $M$  independent of the decision vector  $d$  satisfying (22). In that case,  $\mu = 1$  and dwell time can be freely selected. As we saw for CTMCs, we can always select  $M$  as the identity matrix (and thus, the square of the  $L_2$ -norm as the common Lyapunov function) and ensure this property. However, this choice of  $M$  may have a larger initial error.

So far we discussed reduction and error computation for a given policy  $\pi$ . Notice that the statement of Theorem 4 holds for any policy as long as it has a dwell time at least  $\tau$ . Therefore, we can find a policy using reduced system and apply it to the original CTMDP  $\mathcal{N}$  with the goal of increasing reachability probability. For a given CTMDP  $\mathcal{N}$ , time horizon  $T$ , probability threshold  $\theta$ , and error bound  $\epsilon$ , we select a dwell time  $\tau$  and order of the reduced system such that  $\varepsilon_n \leq \epsilon$  with  $n = T/\tau$ . Then we construct a policy  $\pi$  using the reduced order system (29) by setting  $d_0 = \arg \max_d A_d X(0)$ . The next selection of policies are done by respecting dwell time and  $d_{i+1} = \arg \max_d P_d \bar{A}_d \bar{X}(t)$  for  $t \geq t_i + \tau$  with  $t_i$  being the previous switching time. If the computed interval for reachability probability is not above  $\theta$ , we go back and improve the results by increasing the order of the reduced system.

## 5 Simulation Results

In this section, we use our results for reachability analysis of *tandem network* [12], which is a queuing network shown in Fig. 2 and consists of a  $M/Co2/1$  queue composed with a  $M/M/1$  queue.

Both queuing stations have a capacity of **cap**. The first queuing station has two phases for processing jobs while the second queuing station has only one

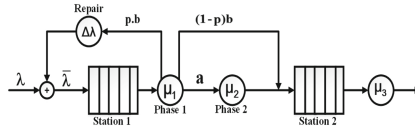


Fig. 2. A typical tandem network.

phase. Processing phases are indicated by circles in Fig. 2. Jobs arrive at the first queuing station with rate  $\bar{\lambda}$  and are processed in the first phase with rate  $\mu_1$ . After this phase, jobs are passed through the second phase with probability  $a$ , which are then processed with rate  $\mu_2$ . Alternatively, jobs will be sent directly to the second queuing station with probability  $b$ , a percent of which will have to undergo a repair phase and will go back to the first station with rate  $\Delta\lambda$  to be processed again. Processing in the second station has rate  $\mu_3$ .

The tandem network can be modelled as a CTMC with a state space of size determined by **cap**. We find the probability of reaching to the configurations in which both stations are at their full capacity (blocked state) starting from a configuration in which both stations are empty (empty state). We consider **cap** = 5 which results in a CTMC with 65 states. We have chosen values  $\mu_1 = \mu_2 = 2$ ,  $\mu_3 = \lambda = 4$ ,  $a = 0.1$ ,  $b = 0.9$  and  $\Delta\lambda = 0$ . Matrix inequalities (15) are satisfied with  $M$  being identity and  $\kappa = 0.001$ . Using the reduction technique of Sect. 3, we can find approximate solution of reachability with only 3 state variables. Figure 3 gives the error bound as a function of time horizon of reachability and order of the reduced system. As discussed, the error goes to zero exponentially as a function of time horizon. It also converges to zero by increasing the order of reduced system. Figure 5 (left) shows reachability probability computed over the tandem network and the reduced order system together with the error bound as a function of time horizon. The error has the initial value 0.36, computed via the choice of initial reduced state in (17), and converges to zero exponentially with rate 0.001. Our experiments show that for a fixed error bound 0.2 and time horizon 100 s, we get reductions of 43% in dimension and 53% in computation time.

Now consider a scenario that the network can operate in *fast* or *safe* modes. In fast mode, less jobs are sent through the second phase (corresponding to a

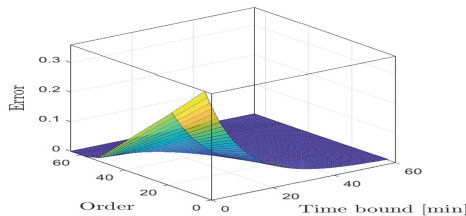
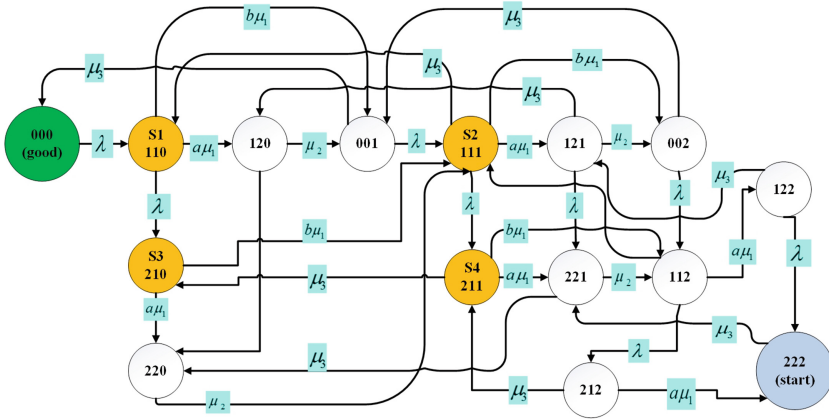


Fig. 3. Error bound as a function of time horizon and order of the reduced system.

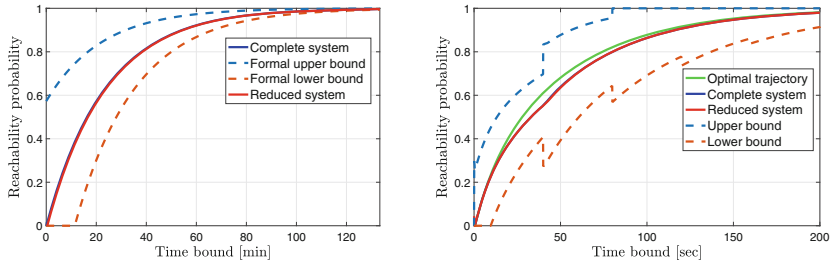


**Fig. 4.** State diagram of a CTMDP with 16 states and 16 decision vectors corresponding to a tandem network with capacity 2. States  $S_1, S_2, S_3, S_4$  have two modes with rates  $a \in \{0.6, 0.7\}$ .

smaller value of  $a$ ); this, in turn, increases the probability that jobs which did not pass second phase, need to be processed again. We model influence of returned jobs as an increase in  $\Delta\lambda$ .

We consider the case that there are two possible rates  $a \in \{0.6, 0.7\}$  corresponding respectively to fast and safe modes. If fast mode is chosen, 10% of jobs will be returned ( $p = 0.1$ ) with rate  $\Delta\lambda = 0.05$ . In the safe mode, only 5% of jobs ( $p = 0.05$ ) will be returned with the same rate  $\Delta\lambda$ . We set  $\lambda = 6$  and other rates are kept the same as defined above.

A tandem network with capacity  $\mathbf{cap} = 2$  and these two modes can be modeled as a CTMDP with 16 states and 16 decision vectors. Figure 4 depicts state diagram of this CTMDP with states  $S_1, S_2, S_3, S_4$  having two modes with the corresponding value of rate  $a$ . We assume the tandem network is initially at blocked state and consider synthesising a strategy with respect to the probability of having both queuing stations being empty. We have implemented the approach of Sect. 4 and obtained a reduced system of order 6 with  $\varepsilon_0 = 0.06$  for given time bound  $T = 200\text{s}$  and dwell time  $\tau = 40\text{s}$ . Figure 5 (right) demonstrates reachability probabilities as a function of time for both tandem network and its reduced counterpart together with the error bound and the optimal trajectory. Intuitively, choosing fast mode in the beginning will result in faster progress of the tasks, especially when queues are more loaded; however, continue of such a selection will result in high number of returned jobs which is not desired. This behaviour is observed depending on the state and three switches happens in states  $S_2, S_3, S_4$ . Jumps in the error bound are in the order of  $10^{-4}$  thus not noticeable in Fig. 5 (right).



**Fig. 5. Left:** approximate reachability probability for tandem network as a function of time horizon with guaranteed error bounds; **Right:** approximate reachability probability for tandem network with 16 decision vectors including guaranteed error bounds.

## 6 Discussions

We have taken a control-theoretic view on the time bounded reachability problem for CTMCs and CTMDPs. We show the dynamics associated with the problems are stable, and use this as the basis for state space reduction. We define reductions as generalised projections between state spaces and find a Lyapunov characterisation of the error between the original and the reduced dynamics. This provides a formal error bound on the solution which decreases exponentially over time. Our experiments on queueing systems demonstrate that, as the time horizon grows, we can get significant reductions in state (and thus, model checking complexity).

## References

1. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model-checking continuous-time Markov chains. *ACM Trans. Comput. Logic* **1**(1), 162–170 (2000)
2. Bacci, G., Bacci, G., Larsen, K.G., Mardare, R.: On the total variation distance of semi-Markov chains. In: Pitts, A. (ed.) *FoSSaCS 2015*. LNCS, vol. 9034, pp. 185–199. Springer, Heidelberg (2015). [https://doi.org/10.1007/978-3-662-46678-0\\_12](https://doi.org/10.1007/978-3-662-46678-0_12)
3. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Softw. Eng.* **29**(6), 524–541 (2003)
4. Boyd, S., Ghaoui, L.E., Feron, E., Balakrishnan, V.: *Linear matrix inequalities in system and control theory*. In: SIAM, vol. 15 (1994)
5. Buchholz, P., Hahn, E.M., Hermanns, H., Zhang, L.: Model checking algorithms for CTMDPs. In: Gopalakrishnan, G., Qadeer, S. (eds.) *CAV 2011*. LNCS, vol. 6806, pp. 225–242. Springer, Heidelberg (2011). [https://doi.org/10.1007/978-3-642-22110-1\\_19](https://doi.org/10.1007/978-3-642-22110-1_19)
6. Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Metrics for labelled Markov processes. *Theor. Comput. Sci.* **318**(3), 323–354 (2004)
7. Doyle, J., Francis, B., Tannenbaum, A.: *Feedback Control Theory*. Macmillan Publishing Co., New York (1990)
8. Feller, W.: *An Introduction to Probability Theory and Its Applications*. Wiley, Hoboken (1968)

9. Girard, A., Pappas, G.J.: Approximate bisimulation relations for constrained linear systems. *Automatica* **43**(8), 1307–1317 (2007)
10. Girard, A., Pola, G., Tabuada, P.: Approximately bisimilar symbolic models for incrementally stable switched systems. *IEEE Trans. Autom. Control* **55**(1), 116–126 (2010)
11. Grant, M.C., Boyd, S.P.: Graph implementations for nonsmooth convex programs. In: Blondel, V.D., Boyd, S.P., Kimura, H. (eds.) *Recent Advances in Learning and Control*. Lecture Notes in Control and Information Sciences, vol 371. Springer, London (2008). [https://doi.org/10.1007/978-1-84800-155-8\\_7](https://doi.org/10.1007/978-1-84800-155-8_7)
12. Hermanns, H.: Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In: *Numerical Solution of Markov Chains (NSMC 1999)*, pp. 188–207 (1999)
13. Khalil, H.K.: *Nonlinear Systems*. Prentice Hall, Upper Saddle River (1996)
14. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. *Inf. Comput.* **94**(1), 1–28 (1991)
15. Lofberg, J.: YALMIP: a toolbox for modeling and optimization in MATLAB. In: *IEEE International Conference on Robotics and Automation*. IEEE (2004)
16. Rabe, M.N., Schewe, S.: Finite optimal control for time-bounded reachability in CTMDPs and continuous-time Markov games. *Acta Informatica* **48**(5–6), 291–315 (2011)



# On Saturation Effects in Coupled Speed Scaling

Maryam Elahi and Carey Williamson<sup>(✉)</sup>

Department of Computer Science, University of Calgary, Calgary, Canada  
carey@cpsc.ucalgary.ca

**Abstract.** In coupled speed scaling systems, the speed of the CPU is adjusted dynamically based on the number of jobs present in the system. In this paper, we use Markov chain analysis to study the autoscaling properties of an M/GI/1/PS system. In particular, we study the saturation behaviour of the system under heavy load. Our analytical results show that the mean and variance of system occupancy are not only finite, but tightly bounded by polynomial functions of the system load and the speed scaling exponent. We build upon these results to study the speed, utilization, and mean busy period of the M/GI/1/PS. Discrete-event simulation results confirm the accuracy of our analytical models.

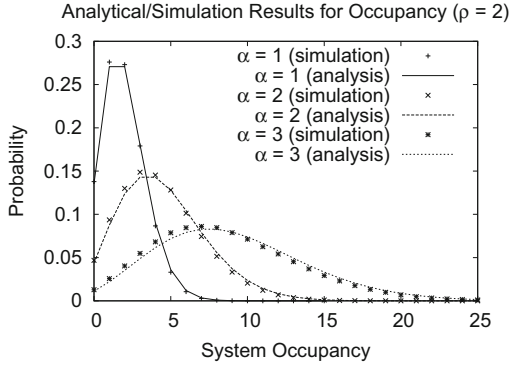
## 1 Introduction

Coupled speed scaling systems adjust the CPU speed dynamically based on the number of jobs in the system. These dynamic speed scaling systems provide tradeoffs between response time and energy consumption [1, 2]. Specifically, running the CPU faster improves the response time, but consumes more energy.

Within a speed scaling system, the two most important considerations are the scheduler and the speed scaling function. The scheduler determines which job is executed next, and the speed scaling function determines the speed at which that job is executed. A popular approach for the latter is *job-count-based* speed scaling, in which the service rate is a function of the current system occupancy [3, 8, 22]. We refer to this as *coupled* speed scaling, since the service rate is coupled to the system occupancy.

In this paper, we focus on the autoscaling properties of coupled speed scaling systems under heavy load. In particular, we consider sustained offered loads that drive the system toward *saturation*, in which the utilization becomes arbitrarily close to unity (i.e.,  $U \rightarrow 1$ ). Note that if there is no limit to the maximum service rate, then the system will automatically adjust (i.e., autoscale) its service rate to accommodate whatever load is presented to it.

Our current paper is motivated by some of our own prior work on the autoscaling properties of coupled speed scaling systems [10]. In particular, our prior work used discrete-event simulation to show that the mean system occupancy remained finite in coupled speed scaling systems under heavy load (see Fig. 1). Furthermore, the mean occupancy was estimated as  $E[N] \approx \rho^\alpha$  [10].



**Fig. 1.** Distribution of system occupancy (based on Fig. 2b in [10])

In our current paper, we present analytical results that bound the mean and variance of occupancy under heavy load. Specifically, we use Markov chain analysis to study the dynamics of a Processor Sharing (PS) speed scaling system, and derive tight bounds on system performance. We then extend our model to analyze the mean busy period for PS under coupled speed scaling. Finally, we use discrete-event simulation to verify the accuracy of our analytic model, and to extend our observations to Shortest Remaining Processing Time (SRPT) systems.

The main insights from our work are the following. First, we show that the mean and variance of the system speed are bounded, even under heavy (but finite) offered load. Second, we show that the mean and variance of system occupancy are tightly bounded, and are polynomial functions of  $\rho$  and  $\alpha$ . Third, we show that the mean busy period in a PS-based coupled speed scaling system grows at least exponentially with offered load. Finally, we show that the mean busy period for an SRPT-based system grows much faster than that for the corresponding PS-based system.

The rest of this paper is organized as follows. Section 2 reviews prior literature on speed scaling systems. Section 3 presents our system model. Section 4 presents our analytical and numerical results. Section 5 presents simulation results. Finally, Sect. 6 concludes the paper.

## 2 Background and Related Work

Prior research on speed scaling systems appears in two different research communities: theory and systems. Theoretical work typically focuses on the optimality of speed scaling systems under some simplifying assumptions (e.g., unbounded service rates, known job sizes). Systems work typically focuses on “good” solutions, rather than optimal ones [7, 8], and especially those that are robust to unknown job sizes, scheduling overheads, as well as finite and discrete system



speeds. In this literature review, we focus primarily on the theoretical work as relevant background context for our paper.

In speed scaling systems, there are many tradeoffs between service rate, response time, and energy consumption. Yao *et al.* [24] analyzed dynamic speed scaling systems in which jobs have explicit deadlines, and the service rate is unbounded. Bansal *et al.* [5] considered an alternative approach that minimizes system response time, within a fixed energy budget. Others have focused on finding the optimal fixed rate at which to serve jobs in a system with dynamically-settable speeds [11, 22, 23].

Several studies indicate that energy-proportional speed scaling is nearly optimal [3, 6]. In this model, the power consumption  $P(s)$  of the system depends only on the speed  $s$ , which itself depends on the number of jobs in the system. Bansal *et al.* [6] showed that SRPT with the speed scaling function  $P^{-1}(n + 1)$  is 3-competitive for an arbitrary power function  $P$ . Andrew *et al.* [3] showed that the optimal policy is SRPT with a job-count-based speed scaling function of the form  $s = P^{-1}(n\beta)$ .

Fairness in dynamic speed scaling systems is also an important consideration. In particular, speed scaling systems induce tradeoffs between fairness, robustness, and optimality [3]. PS is always fair, providing the same expected slowdown for all jobs, even under speed scaling. However, the unfairness of SRPT is magnified under speed scaling, since large jobs tend to run only when the system is nearly empty, and hence at lower speeds. While PS is good for fairness, it is suboptimal for both response time and energy [3].

## 3 System Model

### 3.1 Model Overview and Assumptions

We consider a single-server system with dynamically adjustable service rates. Service rates are changed only when the system occupancy changes (i.e., at job arrival and departure points). There is no cost incurred for changing the service rate, and no limit on the maximum possible service rate. (Prior work by others has considered bounded service rates [11].)

The workload presented to the server is a sequence of jobs with random arrival times and sizes. We assume that the arrival process is Poisson, with mean arrival rate  $\lambda$ . The size (work) of a job represents the time it takes to complete the job when the service rate is  $\mu = 1$ . We assume that job sizes are exponentially distributed and independent. Unless stated otherwise, we assume that the mean job size is  $E[X] = 1$ . Table 1 summarizes our model notation.

In this paper, we consider two specific work-conserving scheduling policies, namely PS and SRPT. PS shares the CPU service rate equally amongst all jobs present in the system, while SRPT works exclusively on the job with the least remaining work. We assume that the schedulers know all job sizes upon arrival, or can at least estimate them dynamically [8]. A job in execution may be preempted and later resumed without any context-switching overhead.

**Table 1.** Model notation

Symbol	Description
$\lambda$	Mean job arrival rate
$\mu$	Service rate
$\mu_n$	Service rate in state $n$
$E[X]$	Average size (work) for each job
$\rho$	Offered load $\rho = \lambda/\mu = \lambda E[X]$
$p_n$	Steady-state probability of $n$ jobs in the system (a.k.a. $\pi(n)$ )
$U$	System utilization $U = 1 - p_0$
$n$	Number of jobs
$\phi(n)$	CPU speed as a function of number of jobs
$t$	Time in seconds
$n(t)$	Number of jobs in system at time $t$
$s(t)$	CPU speed at time $t$
$P(s)$	Power consumption when running at speed $s$
$\alpha$	Exponent in power consumption function $P(s) = s^\alpha$

A speed scaling function,  $s(t)$ , specifies the speed of the system at time  $t$ . For coupled speed scaling, the speed at time  $t$  depends on the number of jobs in the system, denoted by  $n(t)$ , and thus is influenced by the scheduling policy. The best known policy uses the speed function  $s(t) = P^{-1}(n(t)\beta)$  [3]. In this paper, we assume  $\beta = 1$ . We also consider  $P(s) = s^\alpha$ , which is commonly used in the literature to model the power consumption of the CPU. Therefore, in the coupled speed scaling model, we use  $s(t) = \sqrt[\alpha]{n(t)} = n(t)^{1/\alpha}$ , where  $\alpha \geq 1$ . When time  $t$  is not relevant, we use  $\phi(n)$  to denote the CPU speed for  $n$  jobs.

In our work, we focus on the PS scheduling policy, which is an example of a symmetric scheduling policy [12]. Such policies do not prioritize based on job size, or any other job trait, but merely treat all arrivals equivalently. Symmetric policies have the important property that their departure process is stochastically identical to their arrival process when time is reversed. Therefore, in the M/GI/1 model, where arrivals are Poisson, the queue occupancy states form a birth-death process regardless of the form of the job size distribution. This result is formalized in the following theorem, the proof of which is given in [12]. A proof for the special case of PS scheduling appears in [16].

**Theorem 1** [12]. In an M/GI/1 queue with a symmetric scheduling policy, the limiting probability that the queue contains  $n$  jobs is:

$$\pi(n) = \frac{\rho^n}{\prod_{i=0}^{\infty} \phi(i)} \pi(0), \quad \text{for } n > 0,$$

where the probability  $\pi(0)$  of the system being empty is given by:

$$\pi(0) = \frac{1}{1 + \sum_{n=1}^{\infty} \frac{\rho^n}{\prod_{i=1}^n \phi(i)}}.$$

Theorem 1 indicates that all symmetric policies have the same occupancy distribution for the same  $\phi(n)$  function. Furthermore, this occupancy distribution is insensitive to the job size distribution, and depends only on the mean job size.

Although FCFS is not a symmetric policy, it is interesting to note that the occupancy distribution for M/M/1 FCFS is equivalent to the occupancy distribution for M/GI/1 symmetric policies with  $\phi(n) = 1$ . In fact, the occupancy distribution under all non-size-based policies is equivalent for a general  $\phi(n)$  [12, 23]. Therefore, in a single-server with some  $\phi(n)$  speed-scaling discipline, in order to study the occupancy distribution under M/GI/1 PS, it suffices to study the occupancy distribution under M/M/1 FCFS. In our work, we consider the special case of  $\phi(n) = n^{1/\alpha}$ , and derive results for the average speed, occupancy, and expected busy period length.

### 3.2 Markov Chain Model

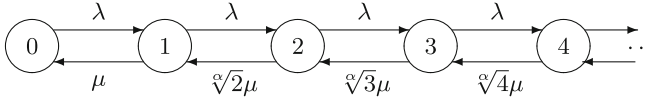
We consider the dynamics of a system with sub-linear speed scaling. Specifically, we consider running the system at speed  $s = n^{1/\alpha}$  when the system occupancy is  $n$  jobs. We consider  $1 \leq \alpha \leq 3$ , which is the relevant range of interest for Dynamic Voltage and Frequency Scaling (DVFS) on modern processors [20, 23]. Note that  $\alpha$  need not be an integer, but is treated as such in the discussion.

The parameter  $\alpha$  determines the set of distinct speeds available in our speed scaling system. For the special case  $\alpha = 1$ , the speeds scale linearly with occupancy, much like the M/M/ $\infty$  queue, which provides a natural validation point for our model. For  $\alpha = 2$ , speeds scale less than linearly with system occupancy, following the “square root speed scaling” approach recommended in the literature (i.e., the system speed when there are  $n$  jobs in the system is  $\sqrt{n} = n^{1/2}$ ). For  $\alpha = 3$ , speeds scale even more slowly with growing system occupancy: the system speed when there are  $n$  jobs in the system is  $\sqrt[3]{n} = n^{1/3}$ . In the limiting case of  $\alpha = \infty$ , the speeds scale so slowly that they are effectively constant (i.e., single-speed system). This provides another validation point for our model.

Figure 2 shows the Markov chain for our speed scaling system. The key difference from Kleinrock’s classic M/M/ $\infty$  model is the change in the service rates  $\mu_n = n^{1/\alpha}\mu$ . Analysis of this chain produces steady-state probabilities  $p_n$  that are analogous to those for the M/M/ $\infty$  chain, except for the effect of the  $1/\alpha$  exponent on all of the service rates.

## 4 Analytical and Numerical Results

In this section, we consider the M/M/1 queue with FCFS scheduling and  $\phi(n)$ -coupled speed scaling, where  $\phi(n) = n^{1/\alpha}$  for  $\alpha \geq 1$ .



**Fig. 2.** Markov chain for coupled speed scaling system model

In the context of the “dynamic service rate” control problem, the M/M/1 FCFS queue with adjustable service rates has been studied in the literature [4, 11, 14, 23], and elegant results for state-dependent speeds that optimize the linear combination of average occupancy and average energy consumption are presented in [11, 23]. However, the proof for the formulation of the occupancy distribution is not provided explicitly. For the sake of completeness, we briefly discuss here the special case of  $n^{1/\alpha}$ -coupled speed-scaling systems.

Consider an  $n^{1/\alpha}$ -coupled speed-scaling system for some  $\alpha > 0$ , and with non-preemptive, non-size-based scheduling. Assume inter-arrival times are exponentially distributed with rate  $\lambda$ , and job sizes are exponentially distributed with rate  $\mu$ . Let  $\rho = \lambda/\mu$ .

In this system, the queue occupancy evolves as a birth-death process since the time between transitions is exponentially distributed. The CTMC for this model is similar to the single-speed M/M/ $\infty$  in that transitions between states occur upon state-independent arrivals with rate  $\lambda$ , and state-dependent departures with rates  $\mu_n$ . Unlike the M/M/ $\infty$ , however, this is a single server model, with at most one job in service at any point in time. When in state  $n > 0$ , provided that no arrival occurs, the time to the next departure is the remaining work of the job in service divided by the service rate  $n^{1/\alpha}$ . Since the service requirements are exponentially distributed with rate  $\mu$ , and the exponential distribution is closed under scaling by a positive factor, the time until the next departure is also exponentially distributed with rate  $\mu_n = \mu n^{1/\alpha}$ . Therefore, the queue occupancy forms a birth-death process, and the limiting probabilities (if they exist) are:

$$\pi(n) = \pi(0) \prod_{i=0}^{n-1} \frac{\lambda_i}{\mu_{i+1}} = \pi(0) \prod_{i=0}^{n-1} \frac{\lambda/\mu}{(i+1)^{1/\alpha}} = \pi(0) \frac{\rho^n}{(n!)^{1/\alpha}}, \quad \text{for } n > 0,$$

where:

$$\pi(0) = \frac{1}{\sum_{i=0}^{\infty} \frac{\lambda_0 \lambda_1 \dots \lambda_{i-1}}{\mu_1 \mu_2 \dots \mu_i}} = \frac{1}{\sum_{i=0}^{\infty} \frac{\rho^i}{(i!)^{1/\alpha}}}.$$

To show that the limiting probabilities exist, and that the chain is ergodic, it suffices to show that the infinite sums converge. Based on the ratio test for convergence of an infinite series, the series  $\sum_{i=0}^{\infty} a_n$  converges if  $\lim_{n \rightarrow \infty} |\frac{a_{n+1}}{a_n}| < 1$ . This condition holds in our case, since  $\alpha > 0$  and  $\mu > 0$ . Specifically,

$$\lim_{n \rightarrow \infty} \frac{\rho^{n+1}/(n+1)^{1/\alpha}}{\rho^n/(n!)^{1/\alpha}} = \lim_{n \rightarrow \infty} \frac{\rho}{(n+1)^{1/\alpha}} < 1.$$

Note that our speed scaling system is just a special case of Theorem 1 with  $\phi(n) = n^{1/\alpha}$ . Therefore, M/GI/1 queues with  $n^{1/\alpha}$ -coupled speed-scaling and

with symmetric scheduling policies, including PS, have the same occupancy distribution as M/M/1 FCFS with  $n^{1/\alpha}$ -coupled speed-scaling.

Unfortunately, we do not have closed form expressions for the foregoing steady-state probabilities. However, it is possible to numerically evaluate the mean and higher moments of the occupancy distribution. In fact, we can derive bounds for the mean and variance of the occupancy distribution (see Sect. 4.2). In the remainder of this section, we make a few observations about the shape of the occupancy distribution.

The steady-state probability distribution in our system is a function of the average load  $\rho$  and the speed-scaling parameter  $\alpha$ . Recall that  $\rho$  is a function of the arrival rate  $\lambda$ , and the job size based on rate  $\mu$ . Note that increasing or decreasing the arrival rate, while adjusting the average job size to keep the average load constant, results in the same occupancy distribution. The parameter  $\alpha$  determines the set of distinct speeds available in the coupled speed-scaling system. For the special case  $\alpha = 1$ , the speeds scale linearly with occupancy, similar to the M/M/ $\infty$  queue. For  $\alpha > 1$ , speeds scale sub-linearly with system occupancy. For very large  $\alpha$ , the speeds scale so slowly that the system effectively behaves like a single-speed system.

The parameter  $\alpha$  has three main impacts on the occupancy distribution, as illustrated in prior work [10]. The first effect of increasing  $\alpha$  is to shift the occupancy distribution to the right (see Fig. 1). This is intuitively expected, since the slower service rates lead to a larger queue of jobs in the system. However, as the backlog of jobs grows, the service rate is also increased, which eventually stabilizes the system. This pendulum effect keeps the mode of the occupancy distribution close to  $\rho^\alpha$ , which determines the average speed ( $\rho$ ) required to serve the load arriving to the system. The second effect of  $\alpha > 1$  is the distortion of the Poisson distribution observed for system occupancy when  $\alpha = 1$ . While the structure of the distribution is similar to Poisson, the state probabilities degenerate, and the Coefficient of Variation (CoV) is greater than that for a Poisson distribution. The particular relationship observed is  $Var[N] \approx \alpha E[N]$  (see Fig. 4(b) for graphical evidence of this observation). In the limiting case of  $\alpha \rightarrow \infty$ , this distribution degenerates to an equal but negligible probability for all states, indicating an unstable (infinite) queue. The third effect that we observe when increasing  $\alpha$  is the decline in  $\pi(0)$ , which is the probability of having an idle system. We call this the *saturation effect*, which is our main focus in this paper. We explore the effect of  $\alpha$  on the utilization, and the expected busy period length, in Sects. 4.3 and 4.4, respectively.

#### 4.1 Mean and Variance of Speed

We first establish some fundamental results regarding the mean and variance of the system speed for coupled speed scaling systems. Let random variables  $S$  and  $N$  denote the speed of the server and the system occupancy, respectively. In the  $n^{1/\alpha}$ -coupled speed-scaling system,  $S = N^{1/\alpha}$ .

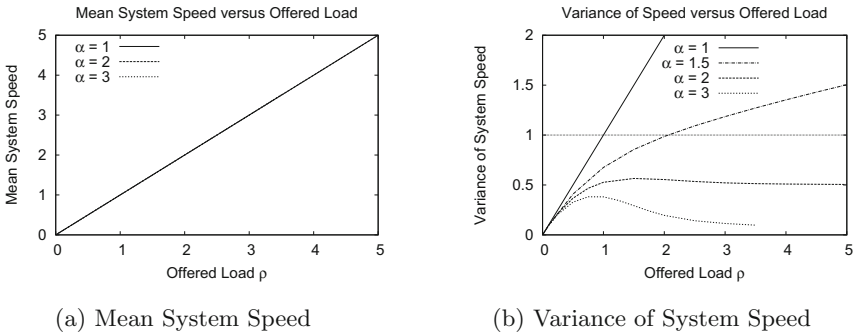
**Theorem 2.** In an M/M/1 queue with  $n^{1/\alpha}$ -coupled speed-scaling,  $E[S] = \rho$ . Furthermore,  $Var[S] < 1$  for any  $\alpha \geq 2$ .

The proof of the first half of Theorem 2 is fairly straightforward, based on the definition of  $E[S]$ . Intuitively, we expect the steady-state average speed to be equal to the incoming load for any stable system (assuming the speed is 0 when there are no jobs in the system). In [23], the general lower bound for the time average speed in all stable speed-scaling systems is argued to be  $\bar{S} \geq \rho$ . Furthermore,  $\bar{S} = \rho$  for systems that run at speed 0 when the system is empty. Our result involves a stochastic proof for the special case of M/M/1 with  $n^{1/\alpha}$ -coupled speed-scaling [9].

The proof for the second part of Theorem 2 is a bit more involved, but we sketch it here. The essence is to use the fact that  $Var[S] = E[S^2] - E[S]^2$ , and to derive a bound on  $E[S^2]$ , since  $E[S]^2 = \rho^2$ . The algebraic derivation culminates in  $Var[S] \leq 1 - \pi(0) < 1$  (see Chap. 4 of [9] for details).

Figure 3 shows the effects of  $\alpha$  and  $\rho$  on the system speed. Figure 3(a) shows the mean speed, which scales linearly with  $\rho$ , regardless of the value of  $\alpha$ . Figure 3(b) shows the variance of speed. When  $\alpha = 1$ ,  $Var[S] = \rho$ , since the speeds follow the Poisson distribution with rate  $\rho$  (analogous to occupancy). For  $\alpha \geq 2$ , the variance is always less than unity. The theoretical bound (indicated by the horizontal line) is not especially tight, but it is a provable bound [9].

For  $\alpha \geq 2$ , the variance of speed initially increases with  $\rho$  up to a point, before decreasing and seemingly converging to a value well below 1. The intuition behind this result is that for larger  $\alpha$ , the speed changes are quite gradual, especially when the occupancy is high. Thus the variance of the speed remains low even when the occupancy fluctuates a lot.



**Fig. 3.** Analytical results for system speed in coupled speed scaling systems

### 4.2 Mean and Variance of Occupancy

We next establish results concerning the mean and the variance of system occupancy. The following Theorem 3 shows that the upper bound for occupancy exceeds the lower bound by at most a polynomial function of  $\alpha$ , which we specify in Definition 1 below. Furthermore, Theorem 3 provides an upper bound on the variance of system occupancy.

**Definition 1.** Let  $f : \mathbb{N} \rightarrow \mathbb{N}$  be  $f(\alpha) = \alpha - 1$  for  $\alpha \in \{1, 2, 3\}$  and  $f(\alpha) = \alpha(\alpha - 1)$  for  $\alpha \geq 4$ .

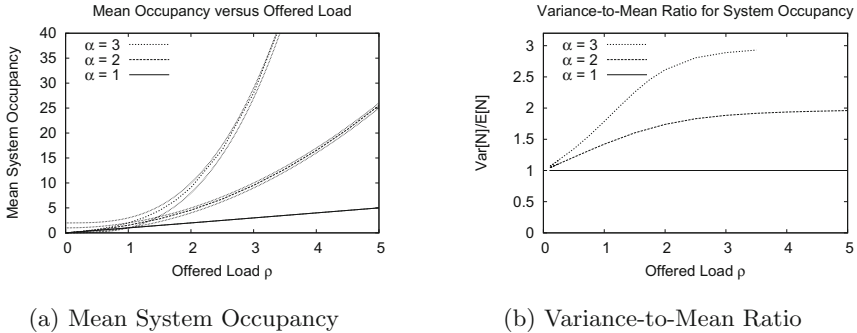
**Theorem 3.** Consider an M/M/1 system with  $n^{1/\alpha}$ -coupled speed-scaling, where  $\alpha \in \mathbb{N}$ . For  $f(\alpha)$  as defined above, the mean system occupancy  $E[N]$  satisfies:

$$\rho^\alpha \leq E[N] \leq \rho^\alpha + f(\alpha).$$

Furthermore, for  $\alpha \geq 2$ ,  $Var[N] \leq E[N](f(\alpha) + 2\alpha - 1)$ .

*Proof.* See Chap. 4 of [9] for details.

Figure 4 shows the effects of  $\alpha$  and  $\rho$  on the mean and variance of occupancy. For mean occupancy, the bounds are quite tight, as shown in Fig. 4(a). Furthermore, these  $E[N]$  values can be used in conjunction with Little’s Law to determine the mean time  $T$  in the system. For  $\alpha = 1$ , the occupancy distribution is Poisson, so the mean and variance are both equal to  $\rho$ . Unlike the variance of speed, which is less than 1 for  $\alpha \geq 2$ , the variance of occupancy is an increasing function of the average load (since  $E[N]$  increases with load). Numerical and simulation results show that, under heavy load,  $Var[N] \approx \alpha E[N]$ , which is even less than the given bound. Figure 4(b) illustrates this phenomenon, by plotting the variance-to-mean ratio for system occupancy as load is increased.



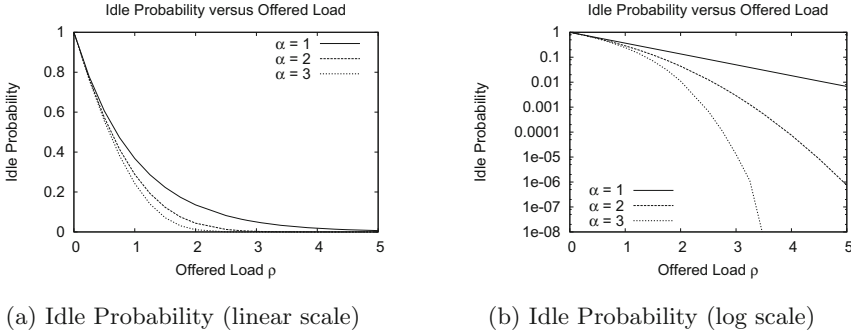
**Fig. 4.** Analytical results for system occupancy in coupled speed scaling systems

### 4.3 System Saturation

We next explore the saturation effect, in which system utilization  $U$  approaches unity. Conversely, the probability  $\pi(0)$  of an empty system approaches zero.

Figure 5 shows  $\pi(0)$  as a function of load, both on a linear scale (Fig. 5(a)) and a logarithmic scale (Fig. 5(b)). We see that with an increase in  $\alpha$ , the probability of the idle state decreases quickly. Recall that  $U = 1 - \pi(0)$  is the utilization of the system. For  $\rho > 1.6$ , systems with  $\alpha \geq 2$  are utilized more than 90% of the time, and for  $\rho \geq 4$ , the utilization exceeds 99.99% in these systems.

For an arbitrary small threshold  $\epsilon > 0$ , one can define a “saturation load”  $\rho$  at which  $\pi(0) \leq \epsilon$ . For example, when  $\epsilon = 10^{-4}$ , the saturation loads would be near 9.2 for  $\alpha = 1$ , 3.9 for  $\alpha = 2$ , and 2.75 for  $\alpha = 3$ . For  $\epsilon = 10^{-6}$ , the saturation loads would be 13.5 ( $\alpha = 1$ ), 4.9 ( $\alpha = 2$ ), and 3.2 ( $\alpha = 3$ ).



**Fig. 5.** Analytical results for saturation in coupled speed scaling systems

Recall that in single-speed systems, the mean occupancy under M/M/1 (FCFS or PS) is  $\frac{\rho}{1-\rho}$ . Furthermore, the average load  $\rho$  is equal to the utilization  $U$ . Therefore, when utilization approaches 1, the mean occupancy under FCFS (equivalently under PS) grows very quickly. In coupled speed-scaling systems, however, M/M/1 FCFS (equivalently PS) with  $n^{1/\alpha}$ -coupled speed-scaling maintains robust performance even when the utilization is close to 1. That is, the mean occupancy is always polynomial in  $\rho$  with degree  $\alpha$  (see Theorem 3).

#### 4.4 Mean Busy Period

In this section, we analyze the expected busy period length under M/M/1 with  $n^{1/\alpha}$ -coupled speed-scaling. Recall that the length of a busy period, denoted by  $B$ , is defined to be the time from when the system becomes busy until the next time that all jobs have left the system, and the system becomes idle. The length of an idle period is denoted by  $I$ .

Our main result is that the mean busy period grows at least exponentially with  $\rho$ . We achieve this result by first establishing the following Theorem 4, and then focusing on Corollary 1.

**Theorem 4.** Consider M/M/1 with  $n^{1/\alpha}$ -coupled speed-scaling with load  $\rho = \lambda/\mu$ , where  $\lambda$  is the arrival rate and  $\mu$  is the rate of the (exponential) job size distribution. Then, the expected busy period length exists, and it satisfies:

$$E[B] = \frac{1}{\lambda} \sum_{i=1}^{\infty} \frac{\rho^i}{(i!)^{1/\alpha}}$$



*Proof.* In a birth-death process, it is known that  $B$  and  $I$  form an alternating renewal process, for which the following equality holds [16]:

$$U = \frac{E[B]}{E[B] + E[I]}$$

where  $U$  is the limiting probability of the system being busy (i.e., utilization). Therefore, the expected busy period length can be derived as a function of the expected idle period length and the utilization as follows:

$$E[B] = \frac{UE[I]}{1 - U}$$

Note that the length of the idle period is the time until the next arrival. Since the arrival process is Poisson with rate  $\lambda$ ,  $E[I] = 1/\lambda$ . By definition,  $U = 1 - \pi(0)$ . Based on Theorem 1, the system is ergodic, and  $\pi(0) = \frac{1}{\sum_{i=0}^{\infty} \frac{\rho^i}{(i!)^{1/\alpha}}} > 0$ .

Therefore,

$$E[B] = \frac{1 - \pi(0)}{\pi(0)\lambda} = \frac{1}{\lambda} \left( \frac{1}{\pi(0)} - 1 \right) = \frac{1}{\lambda} \left( \sum_{i=0}^{\infty} \frac{\rho^i}{(i!)^{1/\alpha}} - 1 \right) = \frac{1}{\lambda} \sum_{i=1}^{\infty} \frac{\rho^i}{(i!)^{1/\alpha}}. \quad \square$$

**Corollary 1.** In an M/M/1 with  $n^{1/\alpha}$ -coupled speed-scaling, for any  $\alpha \geq 1$ ,  $E[B]$  satisfies:

$$E[B] \geq \frac{1}{\lambda} (e^\rho - 1)$$

*Proof.* It is known that  $\sum_{i=0}^{\infty} \frac{\rho^i}{i!} = e^\rho$ . The result then follows directly.  $\square$

This corollary shows that the mean busy period grows at least exponentially with load  $\rho$ , as stated earlier. Note, however, that the busy period duration is sensitive to *both* the arrival rate and the average load, while  $U$  is only a function of the average load.

Figure 6 illustrates the effects of  $\alpha$  and  $\rho$  on the expected busy period length. There are three pairs of lines in this graph, corresponding to  $\alpha = 3$  (highest pair),  $\alpha = 2$  (middle pair), and  $\alpha = 1$  (lowest pair), respectively. Within each pair of lines, the flatter one shows the expected busy period length when the load is changed via the arrival rate (i.e.,  $\rho = \lambda$ , since  $\mu = 1$ ), while the steeper line shows the expected busy period length when the load is changed via the mean of the job size distribution (i.e.  $\rho = E[X] = 1/\mu$ , since  $\lambda = 1$ ). As expected, the trend is similar in both cases, with the lines differing by a factor of  $\lambda$  (note the logarithmic vertical scale on the graph). The lines also differ at the leftmost edge of the graph, since  $E[B] \approx E[X]$  when the load is very light (i.e.,  $\rho \ll 1$ ).

The lower bound given by Corollary 1 is for the general case of  $\alpha \geq 1$ . It is tight for  $\alpha = 1$  (see LB points in Fig. 6), but very loose for  $\alpha \geq 2$ , for which  $E[B]$  grows much faster than  $e^\rho$ , since the system saturates sooner, and much more dramatically. (It is more like  $e^{\rho^\alpha/\alpha}$ , but we have no proof for this yet).

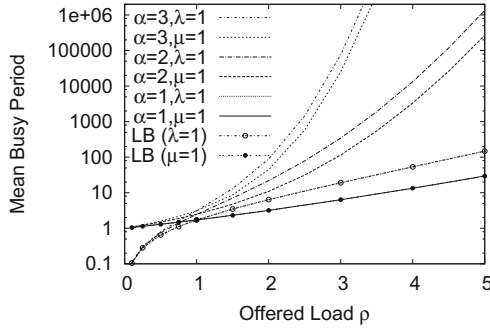


Fig. 6. Analytical results for busy period in coupled speed scaling systems

### 5 Simulation Results

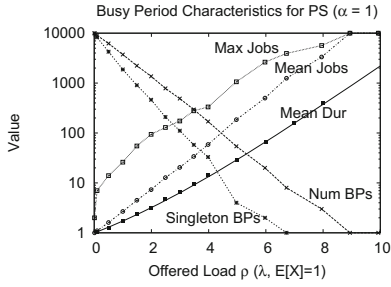
In this section, we use discrete-event simulation to explore saturation effects in coupled speed scaling systems. Our simulator supports different schedulers (e.g., FCFS, PS, SRPT) and speed scaling functions (e.g., coupled, decoupled), and reports results for speeds, response times, energy, and busy period structure [19]. We use this simulator to study the autoscaling dynamics of PS and SRPT.

In our first experiment, we use our simulator to explore the busy period structure of PS-based speed scaling systems. As the load offered to a speed scaling system is increased, the number of busy periods diminishes until there is a single massive busy period that includes all jobs. We refer to this phenomenon as *saturation*, since  $U \rightarrow 1$ .

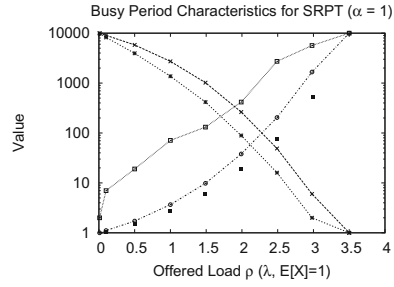
Figure 7(a) illustrates the saturation effect, based on simulation of a PS-based system with linear speed scaling (i.e.,  $\alpha = 1$ ). The horizontal axis shows the offered load based on the arrival rate  $\lambda$ , assuming that the mean job size  $E[X] = 1$ , while the vertical axis shows the value of different busy period metrics, on a logarithmic scale.

The downward-sloping diagonal line on the graph shows the number of busy periods observed, in a simulation run with a total of 10,000 jobs. Furthermore, the dashed line just beneath it shows the number of busy periods that have only a single job. At light load, there are thousands of busy periods, and most have just a single job. As the load increases, the number of busy periods decreases, as does the number of singleton busy periods. The straight-line behaviour on this log-linear plot indicates exponential decline, consistent with the mathematical model.

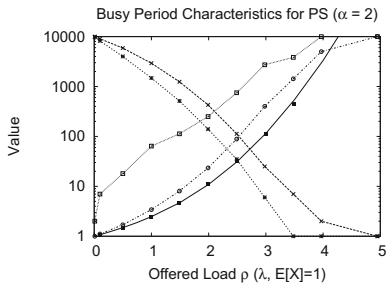
In Fig. 7(a), the upward-sloping dotted line shows the average number of jobs per busy period, while the line above it shows the maximum number of jobs observed in any of the busy periods seen. Both of these lines increase with load, and asymptotically approach a limit that reflects a single massive busy period containing all of the jobs. For  $\alpha = 1$ , this limit is near  $\lambda = 9$ , though the simulation results are somewhat noisy near this point.



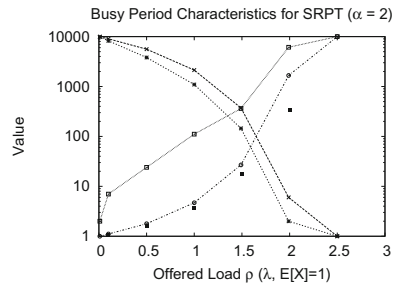
(a) PS ( $\alpha = 1$ )



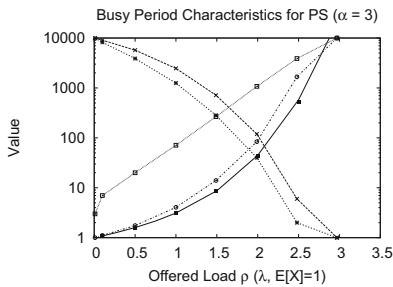
(d) SRPT ( $\alpha = 1$ )



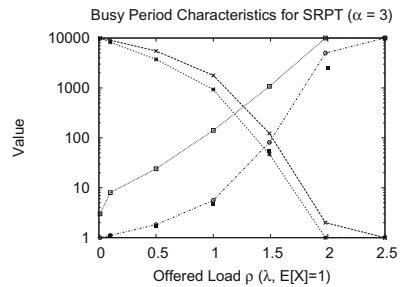
(b) PS ( $\alpha = 2$ )



(e) SRPT ( $\alpha = 2$ )



(c) PS ( $\alpha = 3$ )



(f) SRPT ( $\alpha = 3$ )

**Fig. 7.** Busy period characteristics for PS and SRPT scheduling (simulation)

Analytically, from the Poisson distribution, we know that  $p_0 = e^{-\lambda}$  when  $\alpha = 1$ . For some suitably chosen small  $\epsilon > 0$ , this formula can be used to determine the load  $\lambda$  at which  $p_0 \leq \epsilon$ . For example, for  $\epsilon = 0.0001$ , solving  $\lambda = -\ln(\epsilon)$  yields  $\lambda = 9.2$ , which closely matches the simulation results.

The solid line in Fig. 7(a) shows the mean busy period duration calculated using our analytical result from Theorem 4, while the black squares show the simulation results. The close agreement provides validation for our model.

Figures 7(b) and (c) show the results for PS when  $\alpha = 2$  and  $\alpha = 3$ , respectively. Note that the horizontal scales of these graphs differ from those in Fig. 7(a). The busy period dynamics in these graphs are structurally similar to Fig. 7(a), wherein light load has many very small busy periods, while heavier loads have fewer and larger busy periods. The primary differences from Fig. 7(a) are the distinct downward curvature for the lines showing the number of busy periods, implying a decrease that is faster than exponential as load is increased. Furthermore, the point at which saturation occurs, as load is increased, arises sooner when  $\alpha$  is larger. For example, the load levels at which saturation occurs in the simulation are  $\lambda = 5$  for  $\alpha = 2$ , and  $\lambda = 3$  for  $\alpha = 3$ . These closely match the predictions from our saturation analysis.

Despite the saturation of the utilization  $U$ , the speed scaling system still remains stable, even if the load is further increased. The probability of the system returning to the empty state becomes very small, but the system is still recurrent.

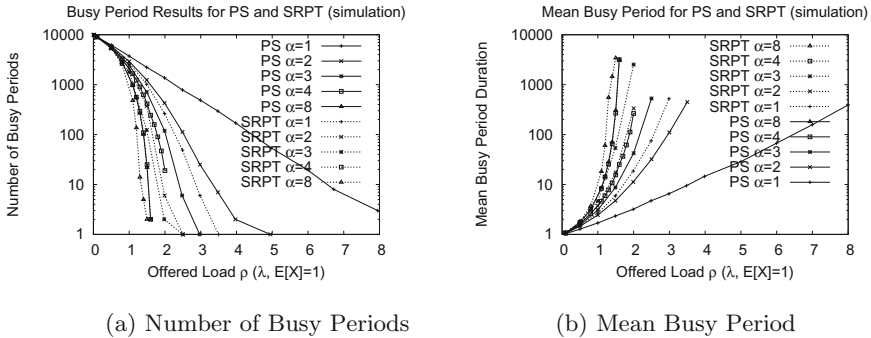


Fig. 8. Busy period comparison for PS and SRPT scheduling (simulation)

The right-hand side of Fig. 7 shows the busy period results from our second set of simulation experiments, for an SRPT-based speed scaling system. Note that on each row of graphs, the horizontal scales for PS and SRPT plots differ.

The main observation here is that the saturation load for SRPT is different than under PS scheduling. In particular, the SRPT system saturates sooner. One implication of this observation is that there exist load levels at which SRPT is beyond saturation, while PS is not. In such scenarios, there will be significant unfairness for large jobs under SRPT (i.e., starvation). That is, while the average number of jobs in the system is the same for both PS and SRPT, the SRPT system tends to retain the largest jobs, causing anomalously high response times [10].

Another anecdotal observation from the simulation results is that the busy period structure for an SRPT system with speed scaling exponent  $\alpha$  is qualita-

tively similar to that for a PS system with speed scaling exponent  $2\alpha$  (at least over the range of parameters considered here). Figure 8 illustrates this result, both for the number of busy periods in Fig. 8(a), and the busy period duration in Fig. 8(b). Furthermore, the saturation point in Fig. 8(a) asymptotically approaches 1 (as expected) when  $\alpha$  is increased from 1 to 8.

## 6 Conclusions

In this paper, we have used mathematical analysis and simulation to explore the autoscaling properties of dynamic speed scaling systems. We have assumed coupled (i.e., job-count-based) speed scaling, with PS as a representative symmetric scheduler. We focus particularly on heavy loads that cause the system to approach saturation (i.e.,  $U \rightarrow 1$ ).

The main conclusions from our work are the following. First, the mean and variance of the system speed are bounded, as long as the offered load is finite. Second, the mean and variance of system occupancy are tightly bounded by polynomial functions of  $\rho$  and  $\alpha$ . Third, the mean busy period in a PS-based coupled speed scaling system grows at least exponentially with offered load when  $\alpha = 1$ , and even faster than this when  $\alpha > 1$ . Finally, we show that SRPT-based systems saturate sooner than the corresponding PS-based system. While such a system remains stable (in terms of job occupancy), it can manifest extreme unfairness due to starvation of the largest jobs.

Our ongoing work is exploring tighter bounds for the mean busy period in both PS and SRPT systems.

**Acknowledgements.** The authors thank the QEST 2018 reviewers for their constructive and insightful comments on an earlier version of this paper, and Philipp Woelfel for many hours discussing and analyzing proofs. Financial support for this work was provided by Canada's Natural Sciences and Engineering Research Council (NSERC).

## References

1. Albers, S., Mueller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proceedings of ACM SPAA, pp. 289–298 (2007)
2. Albers, S.: Energy-efficient algorithms. *Commun. ACM* **53**(5), 86–96 (2010)
3. Andrew, L., Lin, M., Wierman, A.: Optimality, fairness, and robustness in speed scaling designs. In: Proceedings of ACM SIGMETRICS, pp. 37–48, June 2010
4. Ata, B., Shneorson, S.: Dynamic control of an M/M/1 service system with adjustable arrival and service rates. *Manag. Sci.* **52**(11), 1778–1791 (2006)
5. Bansal, N., Kimbrel, T., Pruhs, K.: Speed scaling to manage energy and temperature. *J. ACM* **54**, 1–39 (2007)
6. Bansal, N., Chan, H., Pruhs, K.: Speed scaling with an arbitrary power function. In: Proceedings of ACM-SIAM Symposium on Discrete Algorithms (2009)
7. Dell'Amico, M., Carra, D., Pastorelli, M., Michiardi, P.: Revisiting size-based scheduling with estimated job sizes. In: Proceedings of IEEE MASCOTS, Paris, pp. 411–420 (2014)

8. Dell'Amico, M., Carra, D., Pastorelli, M., Michiardi, P.: PSBS: Practical Size-Based Scheduling. *IEEE Trans. Comput.* **65**(7), 2199–2212 (2016)
9. Elahi, M.: Optimality and fairness in speed-scaling systems. Ph.D. dissertation, Department of Computer Science, University of Calgary, September 2017
10. Elahi, M., Williamson, C.: Autoscaling effects in speed scaling systems. In: Proceedings of IEEE MASCOTS, London, pp. 307–312 (2016)
11. George, J., Harrison, J.: Dynamic control of a queue with adjustable service rate. *Oper. Res.* **49**(5), 720–731 (2001)
12. Kelly, F.: Reversibility and Stochastic Networks. Wiley, New York (1979)
13. Kleinrock, L.: Queueing Systems, Volume 1: Theory. Wiley, New York (1975)
14. Low, D.: Optimal dynamic pricing policies for an M/M/s queue. *Oper. Res.* **22**(3), 545–561 (1974)
15. Lu, D., Shen, H., Dinda, P.: Size-based scheduling policies with inaccurate scheduling information. In: Proceedings of IEEE/ACM MASCOTS, Volendam, Netherlands, pp. 31–38 (2004)
16. Ross, S.: Stochastic Processes. Wiley, New York (1983)
17. Schrage, L.: A proof of the optimality of the shortest remaining processing time discipline. *Oper. Res.* **16**, 678–690 (1968)
18. Schroeder, B., Harchol-Balter, M.: Web servers under overload: how scheduling can help. *ACM Trans. Internet Technol.* **6**(1), 20–52 (2006)
19. Skrenes, A., Williamson, C.: Experimental calibration and validation of a speed scaling simulator. In: Proceedings of IEEE MASCOTS, London, UK, pp. 105–114 (2016)
20. Snowdon, D., Le Sueur, E., Petters, S., Heiser, G.: Koala: a platform for OS-level power management. In: Proceedings of ACM EuroSys, pp. 289–302 (2009)
21. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. In: Proceedings of USENIX OSDI (1994)
22. Wierman, A., Andrew, L., Tang, A.: Power-aware speed scaling in processor sharing systems. In: Proceedings of IEEE INFOCOM, April 2009
23. Wierman, A., Andrew, L., Tang, A.: Power-aware speed scaling in processor sharing systems: optimality and robustness. *Perform. Eval.* **69**, 601–622 (2012)
24. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proceedings of ACM FOCS, pp. 374–382 (1995)

## Author Index

- Ábrahám, Erika 20  
Aichernig, Bernhard K. 36  
Arming, Sebastian 53
- Bacci, Giorgio 339  
Bacci, Giovanni 71  
Bartocci, Ezio 53, 323  
Biagi, Marco 87  
Bonakdarpour, Borzoo 20  
Bortolussi, Luca 157, 323  
Bucur, Doina 306
- Cardelli, Luca 104  
Carnevali, Laura 87  
Chatterjee, Krishnendu 53  
Chatzikokolakis, Konstantinos 1
- Elahi, Maryam 257, 407
- Fawaz, Ahmed 373  
Fu, Hongfei 122
- Gainer, Paul 140  
Großmann, Gerrit 157
- Hahn, Ernst Moritz 140  
Hansen, Mikkel 71  
Hayhoe, Mary 207  
Hillston, Jane 356
- Ivanov, Dmitry 173
- Jansen, Nils 207  
Jing, Yaping 190  
Junges, Sebastian 207
- Katoen, Joost-Pieter 53, 207  
Keefe, Ken 373  
Kwiatkowska, Marta 223  
Kyriakopoulos, Charalampos 157
- Lal, Ratan 240  
Larsen, Kim Guldstrand 71, 173, 339  
Li, Jianlin 122  
Li, Yi 122
- Majumdar, Rupak 389  
Mardare, Radu 339  
Marin, Andrea 257, 273  
Milios, Dimitrios 289  
Miner, Andrew S. 190  
Mitrani, Isi 257
- Nauta, Meike 306  
Nenzi, Laura 323  
Norman, Gethin 223
- Parker, David 223  
Pedersen, Mathias Ruggaard 339  
Piho, Paul 356  
Prabhakar, Pavithra 240
- Rausch, Michael 373  
Rossi, Sabina 273
- Salamati, Mahmoud 389  
Sanders, William H. 373  
Sanguinetti, Guido 289  
Santoni, Francesco 87  
Santos, Gabriel 223  
Schewe, Sven 140  
Schnoerr, David 289  
Schumi, Richard 36  
Schupp, Sibylle 173  
Silvetti, Simone 323  
Sokolova, Ana 53  
Sottana, Matteo 273  
Soudjani, Sadegh 389  
Srba, Jiří 173  
Stoelinga, Mariëlle 306
- Topcu, Ufuk 207  
Tribastone, Mirco 104  
Tschaikowski, Max 104
- Vandin, Andrea 104  
Vicario, Enrico 87
- Williamson, Carey 257, 407  
Wolf, Verena 157
- Zhang, Ruohan 207