Towards Computer-Aided Quantitative Synthesis

Habilitation Thesis

RNDr. Milan Češka Ph.D.

Brno 2020

Abstract

Computer-aided synthesis is an emerging paradigm in system design that automatically transforms a formal specification into a system that is correct by construction. The synthesis mitigates the cost of the classical design loop consisting of implementation, followed by the verification phase. Recently, new challenges in system design have arisen from applications requiring quantitative reasoning, which include, e.g., synthesis of probabilistic programs/models, approximate computing, or construction of biochemical models. The existing synthesis methods do not, however, sufficiently support quantitative reasoning.

In this thesis, we summarise our contribution towards scalable methods for quantitative synthesis including theoretical foundations, prototype tools and rigorous experimental evaluation using practically relevant application domains. We demonstrate that our work considerably extend capabilities of existing synthesis methods and advance engineering processes towards automated system design.

Keywords

Quantitative formal methods; syntax-guided synthesis; inductive synthesis; automated decision procedures; evolutionary optimisation; approximation techniques; system design automation; probabilistic systems; computational biochemical models

Acknowledgment

First, I would like to thank my advisers and mentors: Luboš Brim for starting my research and academic career, Marta Kwiatkowska for providing me the world-leading expertise in the area of probabilistic verification and Tomáš Vojnar for helping me to establish my own research group. Second, I would like to thank all my co-authors, especially David Šafránek, Nicola Paoletti, Alessandro Abate, Radu Calinescu, Joost-Pieter Katoen, Ondřej Lengál, Lukáš Holík, and Jan Křetínský. Third, I would like to thank all Ph.D. students I had the chance to work with especially Sven Dražan, Luca Laurenti, Simos Gerasimou, Jiří Matyáš, Vojta Havlena and Sebastian Junges. Last but not least, I would like to thank Gabina for supporting me during my entire research career and my parents for inspiring me to take this career.

Over the time, I has been supported by number of projects, in particular, by Czech Science Foundation projects, the Czech IT4Innovations Centre of Excellence project, EU ECSEL projects and the ERC Advanced Grant VERIWARE project.

Contents

Part I

COMMENTARY

Chapter 1

Introduction

Computer-aided synthesis is a traditional paradigm in system design automation that has recently gained a lot of traction, resulting in better applicability. It has been successfully integrated into the development processes of hardware, software, or even biochemical systems. The aim of the computer-aided synthesis is to alleviate the costly and time-demanding classical system-design loop between implementation (which is usually performed by a system designer or a programmer) and verification (which checks the correctness of the produced system). In contrast, a synthesis procedure automatically transforms a formal specification into a system that is *correct by construction*.

Recent advances in machine learning, evolutionary optimisation, and computational reasoning, including powerful *Boolean satisfiability* (SAT) and *satisfiability modulo theories* (SMT) solvers, have led to the development of scalable synthesis methods such as game-based reactive synthesis [BJP$^+$12], syntax-guided synthesis [ABD$^+$15, HD18], example-driven synthesis [OZ15], or inductive learning [JS17]. These methods mitigate the high theoretical complexity of synthesis problems and provide a significant shift in the system design paradigm. They have been successfully applied in several practically relevant scenarios, e.g., design of bit-streaming programs [SLRBE05] and spreadsheet macros [OZ15], or strategy synthesis for robot motion planning [ZL18]. These methods have significantly pushed the horizon of feasible synthesis and provide opportunities to attack new challenging problems in system design as well as to use the synthesis in new application domains.

Recently new challenges have arisen from applications that require *quantitative reasoning* either due to *quantitative semantics* of the systems under study or due to *quantitative requirements*. Prominent examples of such applications include (i) synthesis of *probabilistic programs and models*, which are more and more often used in hardware and software engineering to quantify and minimise the probability of encountering an anomaly or an unexpected behaviour, (ii) *approximate computing*, representing a modern design paradigm that aims at reducing system resources by introducing a controlled inaccuracy in the developed system, or (iii) synthesis of *computational biological models*, which play an important role in system and synthetic biology.

In this thesis, we summarise our results in the area of quantitative synthesis. We have proposed novel techniques that support efficient quantitative reasoning about many alter-

native designs and system parameters and allow us to rigorously assess their quantitative attributes (e.g., reliability, performance, cost, robustness, etc.). As such, the techniques help designers make informed decisions during the engineering process and thus mitigate its complexity and time-demands. The achieved results include theoretical foundations underlying new scalable methods for quantitative synthesis as well as prototype tools and their detailed experimental evaluations.

The following subsections summarise our results in the different areas of quantitative synthesis and also provide a landscape of the most related work.

Synthesis Techniques for Probabilistic Systems

Randomisation is key to research fields such as dependability under uncertain system components, symmetry breaking in distributed computing, planning under unpredictable environments, and probabilistic programming. Families of alternative designs differing in the structure and system parameters are ubiquitous. Software dependability has to cope with configuration options, in distributed computing the available memory per process is highly relevant, in planning the observability of the environment is pivotal, and program synthesis is all about selecting correct program variants. The automated analysis of such families has to face a formidable challenges – in addition to the state-space explosion affecting each family member, the family size typically grows exponentially in the number of features, options, or observations. This affects many application domains such as quantitative analysis of software product lines [VtBLL18, CDKB18], strategy synthesis in planning under partial observability [CCD16, NPZ17], and probabilistic program synthesis [CP17, GCT18].

We focus on Markov chains (MCs) with parameters that are widely used to describe configurable probabilistic systems. We distinguish two types of parameters: 1) parameters (typically with continuous domains) affecting *transition probabilities or rates* and 2) parameters (typically with finite but very large discrete domains) determining the *system topology*. MCs including only the former type of parameters are known as parametric MCs introduced in parameter synthesis [HKM08, HHZ11] and model repair [BGK$^+$11] problems. The parameters determining the topology go beyond the class of parametric MCs and are essential in many aforementioned applications.

The existing synthesis techniques for parametric probabilistic systems can be divided into the following categories:

Parameter synthesis

Parameter synthesis techniques consider models with uncertain parameters associated to transition probabilities or rates, and analyse how the system behaviour depends on the parameter values. In [BČDŠ13], we have proposed the first technique effectively computing safe probability bounds on the temporal system behaviour over the given parameter space[1]. The technique has been further extended to a more general class of probabilistic

[1] We originally proposed this technique for a subclass of MCs describing stochastic biochemical systems.

systems [ČPP+16, QDJ+16] and implemented in the state-of-the-art probabilistic model checkers PRISM [KNP11] and STORM [DJKV17]. An alternative approach based on building rational functions for the satisfaction probability [HHZ11, DJJ+15] supports also different classes of probabilistic models and synthesis problems, e.g., the model repair problem [CHH+13, PÁJ+15]. However, for the synthesis against time bounded properties, our technique provides a superior scalability and offers efficient data-parallelisation leading to additional speed-up on modern many-core architectures [ČPP+16]. We emphasise that, the aforementioned synthesis techniques build on a numerical parametric analysis allowing for a full exploration of the continuous parameter space (in contrast to search-based techniques discuss below).

Topology synthesis

For parameters over finite domains, topology synthesis problems are NP-complete with respect to the number of parameters [Cho17], and can naively be solved by analysing all individual family members. An alternative is to model the family by a single Markov decision process (MDP) and use standard MDP model-checking algorithms. This approach has been implemented in tools such as ProFeat [CDKB18] but is infeasible for large systems as well as the naive solution.

In our recent work, we have proposed two alternative approaches that significantly improves the scalability of the synthesis process: 1) *abstraction-refinement* scheme over the MDP representation [ČJJK19], and 2) *counter-example guided inductive synthesis* (CEGIS) for MCs [ČHJK19]. The key of the abstraction is to *forget* in which family member the MDP operates. The resulting *quotient* MDP has a single representative for every reachable state in a family member and typically provides a very compact representation allowing an efficient analysis. The CEGIS approach adopts the idea used for syntax-guided synthesis of deterministic programs [SLTB+06, ABD+15] to finite-state probabilistic models and programs. It starts with a sketch, a program with holes, and iteratively searches for good—or even optimal—instantiations of these holes. Rather than checking all instantiations, the design space is pruned by constructing counter-examples of the rejected candidates and deriving their generalisations that potentially rule out many instantiations at once.

An alternative problem, sketching for probabilistic programs that fit a given data, is considered in [NORV15], together with a synthesis algorithm that builds on stochastic search and approximate likelihood computation.

Searched-based synthesis

Searched-based techniques can be used for the general class of MC families (with both types of parameters). However, they typically do not ensure an exhausted exploration of the parameter space. In contrast, they leverage various evolutionary-based optimisation algorithms to drive the search towards feasible or (sub-)optimal solutions [HMZ12]. In [GCT18], the authors adopt searched-based synthesis to probabilistic models. They apply multi-objective optimisation and genetic algorithms to a design template that captures

alternative system designs, and approximate Pareto-optimal set of Markov models associated with the quality optimisation criteria of software systems. Likewise, the approach from [MKBR10] employs evolutionary algorithms to search the configuration space of Palladio Component Models.

We have recently extended the work of [GCT18] towards synthesis of robust systems [CČG+18] ensuring that perturbations in the system parameters cause only small changes in the system behaviour. Apart from a novel approach for the synthesis of robust designs, the proposed synthesis algorithm uniquely integrates searched-based techniques with our parameter analysis [BČDŠ13] to effectively compute robustness over the perturbation of the rate parameters.

Analyses and Synthesis of Chemical Reaction Networks

Chemical Reaction Networks (CRNs) are a versatile language widely used for *modelling and analysis* of biochemical systems [CBHB09] as well as for high-level *programming* of molecular devices [SSW10, Car13]. They provide a compact formalism equivalent to Petri nets [Mur89], Vector Addition Systems (VAS) [KM69] and distributed population protocols [AAER07]. Formal verification methods are now commonly embodied in the design process of biological systems [BL16, GGG+15, HKN+08, LPC+12] in order to reason about their correctness and performance. However, there is still a costly gap between the design and verification process, exacerbated in cases where stochasticity must be considered – this is typically the case for molecular computation. The effective synthesis of rate parameters (determining the speed of the reactions) such that the CRN satisfies a given temporal behaviour is the first step narrowing the gap. We developed precise parameter synthesis algorithms for CRNs [ČDP+17], that combine a computation of probability bounds over given parameter space [BČDŠ13] with a refinement and sampling of the parameter space. Our methods significantly improve on existing approximate techniques that employ discretisation [HKM08]. For the synthesis against linear-time specifications, statistical methods such as Gaussian Process regression have been used [JL11, BMS16]. In contrast to our approach, the statistical estimation cannot provide guaranteed results.

In our recent work, we have considered a more general synthesis problems where also the topology of the CRN (i.e. particular reactions) ensuring the required temporal behaviour is synthesised [CČF+17]. We have proposed a sketching language for CRNs that concisely captures syntactic constraints on the network topology and allows its underspecification. To ensure computational feasibility of the synthesis process, we employ Linear Noise Approximation [VK92, EK09] of CRNs. This approximation allows us to encode the synthesis problem as a SMT problem over a set of parametric ordinary differential equations (ODEs). We have designed and implemented a novel algorithm for the optimal synthesis of CRNs that employs almost complete refutation procedure for SMT over reals and ODEs [EFH08, GAC12, GKC13] and exploits a meta-sketching abstraction controlling the search strategy [BTGC16].

Syntax-guided synthesis has also been employed for data-constrained synthesis, as in [KPS+13, PYH+14, DMY+14], where (deterministic) biological models are derived

from gene expression data. Synthesis of CRNs from input-output functional specifications is considered in [DMPY15, MPP+18], via a SMT-based generation of qualitative CRN models and the consequent parameter estimation. As this approach requires solving an optimisation problem for each qualitative model whose dimension is exponential in the number of molecules, the synthesis is feasible only for small numbers of molecules.

In our work, we also focus on approximation techniques for CRNs. These techniques simplify the underlying system dynamics (e.g. reduce the state space) while preserving the important system behaviour. Therefore, they can improve the scalability of the synthesis process by accelerating the candidate design evaluation as well as by reducing the number of numerical components in the underlying SMT encoding. In [ABČK15], we proposed an adaptive aggregation scheme for CRNs that significantly reduces the state space while providing formal error bounds[2]. The main idea is to dynamically (re-)cluster states having small probability. Comparing to state truncation techniques [MWDH10, MK06], our approach provides better performance for systems with more complicated dynamics.

In some cases, formal bounds on the approximation error can be relaxed and thus a simpler dynamics including hybrid models [HWKT14, CKL16] can be used. Very recently, we have proposed a principally novel approach for scalable analysis of CRNs that leverages a semiquantitative analysis [ČK19] aiming at quantitative precision only in orders of magnitude. It first builds a compact understandable model, which is then crudely analysed. As demonstrated on complex CRNs from literature, our approach reproduces known results, but in contrast to the state-of-the-art methods, it runs with virtually no computational cost and thus offers unprecedented scalability.

Design of Approximate Circuits and Automata

Approximate circuits are digital circuits that trade functional correctness (precision of computation) for various other design objectives such as chip area, performance, or power consumption. Methods allowing one to develop such circuits are currently in high demand as many applications require low-power circuits, and approximate circuits offer a viable solution. Prominent examples of such applications include image and video processing [VM17, VMS17], or architectures for neural networks [MAFL10, MSS+16]. As shown in [YC16, CSGD16a], many applications favour *provable error bounds* on resulting approximate circuits, which makes automated design of such circuits a very challenging and computationally-demanding task. Simulating the circuit on all possible inputs does not scale beyond circuits with more than 12-bit operands even when exploiting modern computing architectures [MSS+16]. To solve this problem, various *formal verification methods* have been used in the circuit optimisation [VS11, CYB+15, SAGK+16] as well as in the circuit approximation including binary decision diagrams (BDDs) [VMS17], boolean satisfiability (SAT) solving [VARR11], model checking [CS+16], or symbolic computer algebra [FGD18]. However, these approaches did still not scale beyond approximation of multipliers with 12-bit operands and adders with 16-bit operands.

[2] This work is not included in the thesis.

In our work, we have proposed a new approximation technique that integrates formal methods, namely SAT solving, into evolutionary-based approximation [ČMM+17]. The key distinguishing idea of our approach is simple, but it makes our approach dramatically more scalable comparing to previous approaches. Namely, we *restrict the resources* (running time) available to the SAT solver when evaluating a candidate solution. If no decision is made within the limit, a minimal score is assigned to the candidate circuit. This approach leads to a *verifiability-driven search strategy* that drives the search towards promptly verifiable approximate circuits. Experimental evaluation demonstrates that, comparing to existing approximation techniques, our approach is able to discover circuits that have much better trade-offs between the precision and energy savings. We have implemented this approach in ADAC [ČMM+18], our tool for automated design of approximate circuits, that is now able to effectively approximate complex arithmetic circuits such as 32-bit multipliers, multiply-and-accumulate circuits, and dividers.

Apart from circuit approximation, we have also developed techniques for quantitative automata reductions. In particular, we focused on *nondeterministic finite automata* used for regex matching in hardware-accelerated network intrusion detection systems. For a given probability distribution of packets in network traffic, our goal is to design approximate automata having the best tradeoffs between the probability that a packet is misclassified and the automaton size. In [ČHH+18, ČHH+19a], we proposed approximate reduction techniques that employ a novel error state labelling. The labelling provides safe bounds on the error introduced by removing a given state. The technique achieves a great size reduction (much beyond the state-of-the-art language-preserving techniques) with a controlled and small error. Their practical usefulness is, however, limited by the size of the automaton to be reduced, i.e. by the complexity of regular expression the automaton represents. Therefore, we have recently proposed lightweight reduction techniques providing only statistical guarantees on the reduction error. The resulting automata have been integrated into a novel FPGA architecture for regex matching [ČHH+19b] that is able to process network traffic beyond 100 Gbps for complex sets of regular expression from SNORT database [Sno]. Our approach significantly improves on the performance of the state-of-the-art hardware-accelerated network intrusion detection systems such as [ARS15, MKP16a, YJB+18].

Related Activities

In 2019, the author of this thesis obtained a Czech Research Foundation grant *CAQtuS: Computer-Aided Quantitative Synthesis* on the research topics presented in this thesis. He was invited to present the achieved results on several seminars including Dagstuhl seminar and faculty seminars at Oxford University, University of Tokyo, TU Muchen, RWTH Aachen University. The proposed synthesis techniques were implemented in several publicly available tools including a tool for automated design of approximate circuits that has received Bronze Human Competitive Awards in Genetic and Evolutionary Computation (Humies) in 2018.

The author has been very active in the research community. He was co-chairing the 16th International Conference on Computational Methods in Systems Biology (CMSB)

in 2018 and the 6th International Workshop on Hybrid Systems and Biology (HSB) in 2019. He was also a co-chair of ETAPS (The European Joint Conferences on Theory and Practice of Software) workshops in 2019. He regularly serves in program committees including conferences and workshops such as CMSB, HSB, QEST, FORMATS, or CIBCB. He was also a guest editor of a special issue in IEEE/ACM Transactions on Computational Biology and Bioinformatics in 2019.

Note on the Author's Contribution

We would like to emphasise that the papers underlying this thesis (see Sections 2.5, 3.5 and 4.4) use the alphabetical ordering of the authors that does not reflect authors' contribution. The exception are the papers [ČŠDB14] and [ČDP+17] where the authors' ordering emphasises the major contribution of the first and second author. In Table 1.1, the author of this thesis tries to describe his contribution to the papers. As there is no agreement on a metric allowing an qualitative evaluation, he focuses on the contribution to commonly accepted parts of the process of creating a paper in computer science.

	topic	approach	proofs	implementation	experiments	writing
[BČDŠ13]						
[ČŠDB14]						
[ČDKP14]						
[ABČK15]						
[ČPP+16]						
[ČDP+17]						
[ČMM+17]						
[CČG+17a]						
[CČG+17b]						
[CČF+17]						
[ČMM+18]						
[CČG+18]						
[ČHH+18]						
[ČHH+19a]						
[ČHH+19b]						
[ČK19]						
[ČJJK19]						
[ČHJK19]						
[ČMM+20]						

Table 1.1: The contributions of the author of this thesis to the selected papers related to the thesis. Black denotes an essential contribution, grey denotes an important contribution, white denotes minor or no contribution, and crosses denote non-applicability. The highlighted papers are attached to this thesis.

Focus and Structure of the Thesis

In the following chapters, we will discuss into more details some of the key ideas underlying the aforementioned results[3]. In particular, we motivate and formalise the problems under study, present the key steps of the solution, and demonstrate its performance and applicability on selected case studies. Chapter 2 presents our results in the area of synthesis techniques for probabilisitic systems and focuses on the parameter synthesis, inductive methods for the topology synthesis, and on the synthesis of robust systems. Chapter 3 presents parameter synthesis techniques for CRNs and optimal syntax-guided synthesis of CRNs under linear noise approximation. We also discus our novel results on semi-quantitave abstraction for CRNs that open new directions for the synthesis of CRNs. Chapter 5 presents our results in the area of automated design of approximate circuits and automata. In particular, it focuses on verifiability-driven search strategy and on automata reduction for regex matching in deep network packet inspection.

The second part of this thesis lists selected papers on which the discussed results are based on.

[3]The results have been achieved since mid-2012, when the author of this thesis obtained the Ph.D.

Chapter 2

Synthesis of Probabilistic Systems

This chapter presents our three most important results in the area of probabilistic system synthesis: 1) parameter synthesis for probabilistic systems [BČDŠ13, ČPP+16, ČDP+17], 2) topology synthesis for probabilistic systems [ČJJK19, ČHJK19] and 3) synthesis of robust stochastic systems [CČG+17a, CČG+18].

2.1 Parameter synthesis for probabilistic systems

Traditionally, probabilisitic model checking techniques assume that model parameters – namely, the transition probability and rate constants – are known a priori. This is often not the case and one has to consider ranges of parameter values instead, for example, when the parameters result from imprecise measurements, or when designers are interested in finding parameter values such that the model fulfils a given specification. Such problems can be effectively formulated in the framework of parameter synthesis for Markov models [HKM08, HHZ11]: given a formula in a suitable (probabilistic) logic [ASSB96, HJ94, ASB+95] and a model whose transition rates/probabilities are functions of the parameters, find parameter values such that the satisfaction probability of the formula meets a given threshold, is maximised, or minimised.

In this section, we present our results [BČDŠ13, ČPP+16, ČDP+17] including synthesis algorithms for *parametric continuous-time Markov chains (pCTMCs)* and *time-bounded continuous stochastic logic (CSL)*. Note that we originally introduced the synthesis algorithms for stochastic biochemical systems, in particular for parametric chemical reaction networks (discussed in Section 3.1.1). Our approach can be, however, straightforwardly applied for general parametric stochastic system with the pCTMC semantics as we shown e.g. in [ČPP+16]. Our results were further generalised for discrete-time models and time unbounded properties using the notation of *parameter lifting* [QDJ+16].

pCTMCs allow transition rates to depend on model parameters. We assume a set K of model parameters. The domain of each parameter $k \in K$ is given by a closed real interval of possible values, i.e, $[k^\perp, k^\top] \subseteq \mathbb{R}$. The *parameter space* $\mathcal{P}$ induced by K is defined as the Cartesian product of the individual intervals, $\mathcal{P} = \bigtimes_{k \in K} [k^\perp, k^\top]$. The key concept of pCTMCs is the *parametric rate matrix* $\mathbf{R} : S \times S \to \mathbb{R}[K]$ where S is the set of states and $\mathbb{R}[K]$ denotes the set of polynomials over the reals $\mathbb{R}$ with variables $k \in K$.

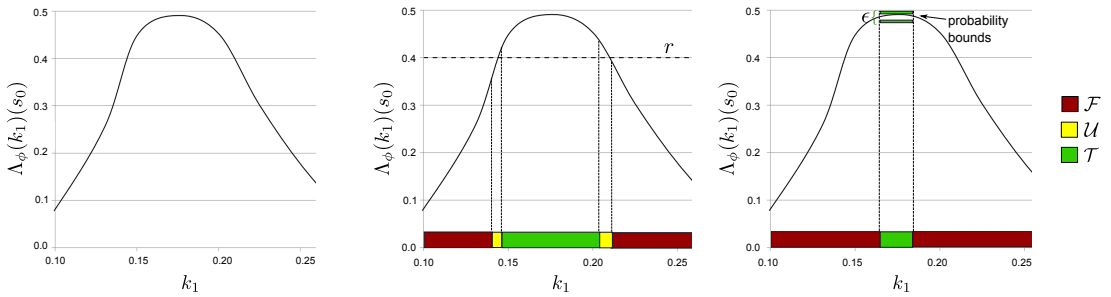


Figure 2.1: **Left:** Example of a satisfaction function. **Centre:** Threshold synthesis for $\mathrm{P}_{\geq 0.4}[\phi]$ and with volume tolerance $\varepsilon = 5\%$. **Right:** Max-synthesis with probability tolerance $\epsilon = 2\%$.

Given a pCTMC and a parameter space $\mathcal{P}$, we denote with $\mathcal{C}_\mathcal{P}$ the set $\{\mathcal{C}_p \mid p \in \mathcal{P}\}$ where $\mathcal{C}_p = (S, \pi, \mathbf{R}_p, L)$ is the instantiated CTMC obtained by replacing the parameters in $\mathbf{R}$ with their valuation in p. The definition restricts the rates to be polynomials, which are sufficient to describe a wide class of stochastic systems.

Problem formulation To introduce the synthesis problems, we introduce a *satisfaction function* to capture how the satisfaction probability of a given property relates the parameters and the initial state[1].

Definition 1 (Satisfaction function) *Let ϕ be a CSL path formula, $\mathcal{C}_\mathcal{P}$ be a pCTMC over a space $\mathcal{P}$ and $s \in S$. We denote with $\Lambda_\phi : \mathcal{P} \to S \to [0,1]$ the satisfaction function such that $\Lambda_\phi(p)(s) = Pr(\omega \in Path(s) \mid \omega \vDash \phi)$ in $\mathcal{C}_p$, i.e. the probability of the set of paths starting in s and satisfying ϕ.*

Figure 2.1 illustrates two synthesis problems, i.e. decomposition of the parameter space $\mathcal{P}$. Given a threshold $\sim r$, where $\sim \in \{<, \leq, >, \geq\}$, and a CSL path formula ϕ, the *threshold synthesis* problem asks for the parameter regions where the probability of ϕ meets $\sim r$ and the regions that violate $\sim r$. The *max synthesis* problem determines the parameter region where the probability of the input formula attains its maximum, together with probability bounds approximating that maximum. Solutions to the threshold synthesis problem admit parameter points left undecided, while, in the max synthesis problem, the actual set of maximising parameters is contained in the synthesised region. The min synthesis problem is defined and solved in a symmetric way to the max case. Formal definition can be found in [ČDP+17]. Note that ϕ allows nested probabilistic operators, and thus the satisfaction function is, in general, not continuous.

2.1.1 Synthesis algorithms

The core part of the synthesis algorithm an efficient parameter exploration procedure we introduced in [BČDŠ13] and extended in [ČDP+17]. The procedure takes a pCTMC $\mathcal{C}_\mathcal{P}$

[1] For simplicity, we define the function and further describe parameter synthesis only for the probabilistic operator. Our approach can also handle various time-bounded rewards operators (see [ČDP+17]).

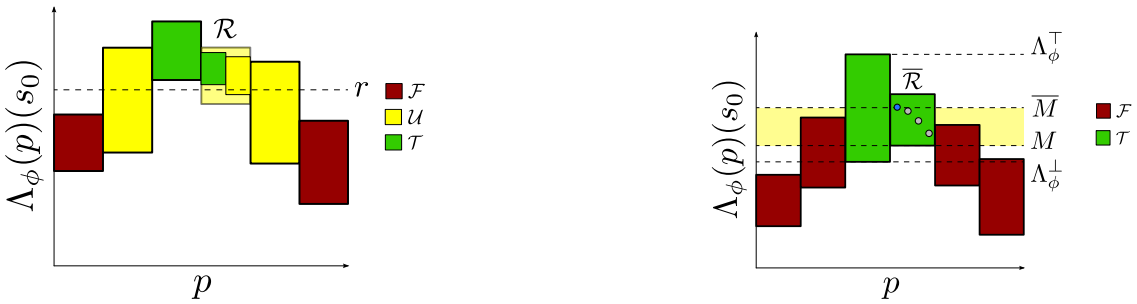


Figure 2.2: **Left:** Refinement in threshold synthesis with $\geq r$. Parameter values are on the x-axis, probabilities on the y-axis. Each box describes a parameter region (width), and its probability bounds (height). The refinement of $\mathcal{R}$ yields regions in $\mathcal{T}$ and in $\mathcal{U}$. **Right:** Refinement in max synthesis. The two outermost regions (in red) cannot contain the maximum, as their upper bound is below the maximum lower bound (M) found at region $\overline{\mathcal{R}}$. The maximum lower bound is improved by sampling several points $p \in \overline{\mathcal{R}}$ and taking the highest value ($\overline{M}$) of the satisfaction function $\hat{\Lambda}_\phi(p)(s_0)$. The yellow area highlights the improvement.

and CSL path formula ϕ, and provides safe under- and over-approximations for the minimal and maximal probability that $\mathcal{C}_\mathcal{P}$ satisfies ϕ, that is, lower and upper bounds satisfying, for all $s \in S$,

$$\Lambda_{\phi,\min}(s) \leq \inf_{p\in\mathcal{P}} \Lambda_\phi(p)(s) \qquad \text{and} \qquad \Lambda_{\phi,\max}(s) \geq \sup_{p\in\mathcal{P}} \Lambda_\phi(p)(s). \tag{2.1}$$

The accuracy of these approximations is improved by partitioning the parameter space $\mathcal{P}$ into subspaces and re-computing the corresponding bounds, which forms the basis of the synthesis algorithms.

The key idea of parameter exploration is to replace the parametric choice at each state and at each time step by a fresh variable with the same domains as the original parameters. This replacement allows us to safely approximate the computationally intractable global optimisation over $\mathcal{P}$ by a chain of simple optimisations. The local optimisation boils down to an optimisation of *multivariate polynomial function*, where the degree of the polynomial depends solely on the degree of the rate functions f_τ. Moreover, for *multi-affine* rate functions, the values $\Lambda_{\phi,\min}(s)$ and $\Lambda_{\phi,\max}(s)$ can be obtained by considering only the extremal values of the variables.

Iterative refinement Having the procedure returning the bounds on the probability for a given parameter space $\mathcal{P}$, the synthesises algorithms are obtained as iterative refinement of $\mathcal{P}$. The key idea of the refinement for the threshold synthesis is illustrated in Figure 2.2 (left). For the max synthesis we employ a bit more involved refinement strategy illustrated in Figure 2.2 (right). When analysing a subspace $\overline{\mathcal{R}}$, the refinement algorithm additionally samples a set of parameters $\{p_1, p_2, \ldots\}$ (the dots) and computes the highest value $\overline{M}$ over $\Lambda_\phi(p_i)(s_0)$ using the standard verification procedure. $\overline{M}$ improves the under-approximation to the maximum of the satisfaction function. As a result, the bound rules out more regions, and fewer refinements are required in the next iteration (compare the bounds M and $\overline{M}$ in the figure).

2.1.2 Data-parallelisation

The complexity of the proposed synthesis algorithms depends mainly on the size of the underlying model and on the number of parameter regions to analyse in order to achieve the desired precision. However, existing parameter synthesis techniques usually do not sufficiently scale with the model size and the dimensionality of the parameter space. For instance, as reported in our earlier work [ČDKP14], the synthesis of two parameters for a model with 5.1K states requires the analysis of 5K parameter regions and takes 3.6 hours.

In the last decade, many-core graphical processing units (GPUs) have been utilised as general purpose, high-performance processing resources in computationally-intensive scientific applications. Relevant applications include data-parallel algorithms for matrix-vector multiplication [BG08] and probabilistic model checking [WB12, BESW10]. In the light of this development, we have redesigned the synthesis algorithms using matrix-vector operations [ČPP+16] to enable an efficient data-parallel processing and acceleration of the synthesis procedures on many-core architectures.

The novelty of our approach is a two-level parallelisation scheme that distributes the workload for the processing of the state space and the parameter space, in order to optimally utilise the computational power of the GPU. The state space parallelisation builds on a sparse-matrix encoding of the underlying parametric CTMC. The parameter space parallelisation exploits the fact that our synthesis algorithms require the analysis of a large number of parameter regions during the parameter space refinement.

The proposed data-parallel synthesis algorithms as well as a number of optimisations of the sequential algorithms have been implemented in our tool PRISM-PSY[2] [ČPP+16] that employs the front-end of the probabilistic model-checker PRISM [KNP11]. Our experiments on several case studies show that the data-parallel synthesis achieves on a single GPU up to a 31-fold speedup with respect to the optimised sequential implementation and that our algorithms provide good scalability with respect to the size of the model and the number of parameter regions to analyse. As a result, PRISM-PSY enables the application of precise parameter synthesis methods to more complex problems, i.e. larger models and higher-dimensional parameter spaces.

2.2 Complete Methods for Topology Synthesis

In this section, we present two orthogonal approaches for topology synthesis for discrete-time MCs over finite design spaces[3]. In contrast to search-based techniques [GCT18], we aim at complete algorithms that explore the entire design space without enumerating and verifying every candidate design individually. The completeness is indeed essential if the optimality or non-existence of the candidate has to be proved as well as if all valid candidates have to be found. Using a straightforward adaption of existing results (e.g. results for augmented interval Markov chains [Cho17, Theorem 3]), it can be shown that the synthesis problem is NP-complete with respect to the number of parameters.

[2] `http://www.prismmodelchecker.org/psy/`

[3] Extension of towards continuous-time MCs is under current investigation.

To mitigate this complexity, we propose techniques that, in many cases, allow efficient quantitative reasoning about a set alternative designs. This is essential in many application domains such as software product lines [CDKB18], strategy synthesis in planning under partial observability [GDF14], or probabilistic program synthesis [GCT18], where a very large set of system topologies has to be explore. Reasoning about alternative topologies poses a significantly harder problem comparing to the parameter synthesis. This is due to the shape of the satisfaction function describing how the probability of a given system behaviour depends on the parameters. In the case of the transition parameters (discussed in the previous section), the function is (piece-wise) continuous typically with a small number of local extrema and thus safe and useful bounds on the function can be computed [ČDP+17]. In the case of parameters affecting the system topology, the function is usually very discontinuous and chaotic and thus alternative techniques are required.

We first define *families* of Markov chains that compactly represent a set of system topologies. We then introduce a MDP-based abstraction of a given family and iterative refinement scheme leading to a novel synthesis algorithm [ČJJK19]. Afterwards, we present a counter-example driven synthesis [ČHJK19]. Finally, we briefly discuss how the synthesis methods can be lifted to the syntax-guided synthesis that operates at the language level rather at the state level.

2.2.1 Families of Markov chains

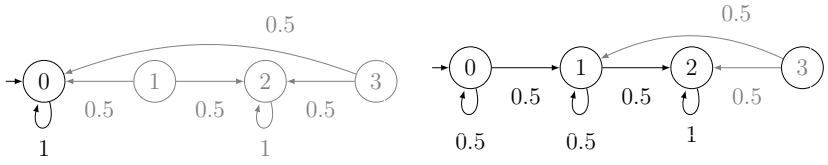
We present our approaches on the basis of an explicit representation of a *family of MCs* using a parametric transition probability function.

Definition 2 (Family of MCs) *A* family of MCs *is defined as a tuple* $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ *where* S *is a finite set of states,* $s_0 \in S$ *is an initial state,* K *is a finite set of discrete parameters such that the domain of each parameter* $k \in K$ *is* $T_k \subseteq S$*, and* $\mathfrak{P}\colon S \to Distr(K)$ *is a family of transition probability matrices.*

Note that the transition probability function of standard MCs maps states to distributions over successor states. For families of MCs, this function maps states to distributions over parameters. Instantiating each of these parameters with a value from its domain yields a "concrete" MC, called a *realisation*. Let $\mathcal{R}^{\mathfrak{D}}$ denote the *set of all realisations* for $\mathfrak{D}$.

Recall that the parametric MCs, defined in the prevision section, use a parametric transition probability function that maps states to parametric distributions over successor states. The parameters affect the transition rate/probability, not the target states. Different system topology is expressed in the the parametric MCs using discrete parameters whose valuation leads to different transition probability functions. The notation of the family allows us a more compact description of the alternative system topologies.

As a family $\mathfrak{D}$ of MCs is defined over finite parameter domains, the number of family members (i.e. realisations from $\mathcal{R}^{\mathfrak{D}}$) of $\mathfrak{D}$ is finite, i.e., $|\mathfrak{D}| := |\mathcal{R}^{\mathfrak{D}}| = \prod_{k \in K} |T_k|$, but exponential in $|K|$. Subsets of $\mathcal{R}^{\mathfrak{D}}$ induce so-called *subfamilies* of $\mathfrak{D}$. While all these MCs share the same state space, their *reachable* states may differ, as demonstrated by the following example.



(a) D_{r_1} with $r_1(k_1) = 0, r_1(k_2) = 2$ (b) D_{r_2} with $r_2(k_1) = 1, r_2(k_2) = 2$

Figure 2.3: The two different realisations of $\mathfrak{D}$.

Example 1 (Family of MCs) *Consider a family of MCs $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ where $S = \{0, 1, 2, 3\}$, $s_0 = 0$, and $K = \{k_0, k_1, k_2\}$ with domains $T_{k_0} = \{0\}, T_{k_1} = \{0, 1\}$, and $T_{k_2} = \{2, 3\}$. The parametric transition function $\mathfrak{P}$ is defined by:*

$$\mathfrak{P}(0) = 0.5 \colon k_0 + 0.5 \colon k_1 \qquad \mathfrak{P}(1) = 0.5 \colon k_1 + 0.5 \colon k_2$$
$$\mathfrak{P}(2) = 1 \colon k_2 \qquad \mathfrak{P}(3) = 0.5 \colon k_1 + 0.5 \colon k_2$$

$\mathfrak{D}$ induces four realisation. Fig. 2.3 shows the two MCs that result from the realisations $\{r_1, r_2\} = \mathcal{R}^{\mathfrak{D}}$. States that are unreachable from the initial state are greyed out.

Specifications. To simplify the presentation, we consider only unbounded reachability and expected reward specifications, however, our approaches may be extended to richer logics like arbitrary PCTL [HJ94], PCTL* [ASB+95], or ω-regular properties.

Synthesis problems. We consider the following formulations of synthesis problems for families of MCs: 1) *feasibility* synthesis: Does some member in $\mathfrak{D}$ satisfy the specification? 2) *threshold* synthesis: Which members of $\mathfrak{D}$ satisfy the specification? 3) *optimal* synthesis: Which family members satisfy φ optimally, e.g., with the highest probability?

Example 2 (Synthesis problems) *Recall the family of MCs $\mathfrak{D}$ from Example 1. For the specification $\varphi = \mathbb{P}_{\geq 0.1}(\Diamond\{1\})$, the solution to the threshold synthesis problem is $T = \{r_2, r_3\}$ and $F = \{r_1, r_4\}$, as the goal state 1 is not reachable for D_{r_1} and D_{r_4}. For $\varphi = \Diamond\{1\}$, the solution to the max synthesis problem on $\mathfrak{D}$ is r_2 or r_3, as D_{r_2} and D_{r_3} have probability one to reach state 1.*

2.2.2 Synthesis via refinement of MDP-based abstraction

Our first synthesis approach builds on an abstraction of the family $\mathfrak{D}$ in the form of Markov Decision Process (MDP). The abstraction is iteratively refined until it provides sufficient precision to solve the required synthesis problem over $\mathfrak{D}$. To build the abstraction, we first consider a single MDP (so-called *all-in-one MDP* [GS13, RAN+15, CDKB18]) that subsumes all individual MCs of a family $\mathfrak{D}$, and is equipped with an appropriate action and state labelling to identify the underlying realisations from $\mathcal{R}^{\mathfrak{D}}$.

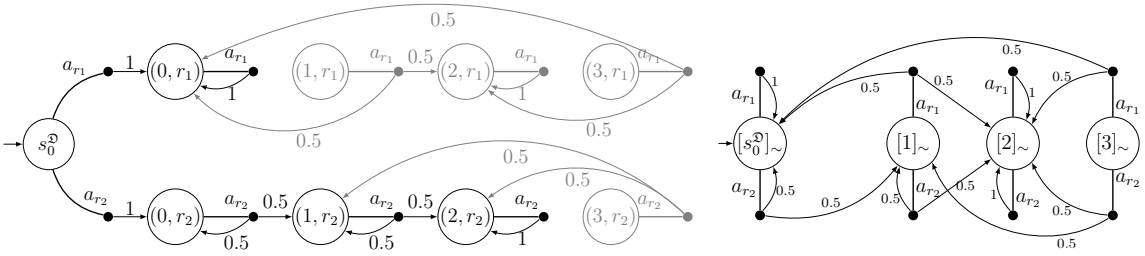


Figure 2.4: **Left:** Reachable fragment of the all-in-one MDP $M^{\mathfrak{D}}$ for realisations r_1 and r_2. **Right:** The quotient MDP $M^{\mathfrak{D}}_{\sim}$ for realisations r_1 and r_2.

Definition 3 *The* all-in-one MDP *of a family $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ of MCs is given as $M^{\mathfrak{D}} = (S^{\mathfrak{D}}, s_0^{\mathfrak{D}}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}})$ where $S^{\mathfrak{D}} = S \times \mathcal{R}^{\mathfrak{D}} \cup \{s_0^{\mathfrak{D}}\}$, $Act^{\mathfrak{D}} = \{a^r \mid r \in \mathcal{R}^{\mathfrak{D}}\}$, and $\mathcal{P}^{\mathfrak{D}}$ is defined as follows:*

$$\mathcal{P}^{\mathfrak{D}}(s_0^{\mathfrak{D}}, a^r)((s_0, r)) = 1 \quad \textit{and} \quad \mathcal{P}^{\mathfrak{D}}((s, r), a^r)((s', r)) = \mathfrak{P}(r)(s)(s').$$

Example 3 (All-in-one MDP) *Fig. 2.4 (left) shows the all-in-one MDP $M^{\mathfrak{D}}$ for the family $\mathfrak{D}$ of MCs from Example 1 – for the sake of readability, we only include the transitions and states that correspond to realisations r_1 and r_2. Again, states that are not reachable from the initial state $s_0^{\mathfrak{D}}$ are marked grey.*

Model checking the all-in-one MDP determines max or min probability (or expected reward) for all states, and thereby for all realisations, and thus provides a solution to both synthesis problems. Clearly, the MDP may be too large for realistic problems. Therefore, we define a predicate abstraction that at each state of the MDP *forgets* in which realisation we are, i.e., abstracts the second component of a state (s, r).

Definition 4 (Forgetting) *Let $M^{\mathfrak{D}} = (S^{\mathfrak{D}}, s_0^{\mathfrak{D}}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}})$ be an all-in-one MDP.* Forgetting *is an equivalence relation $\sim_f \subseteq S^{\mathfrak{D}} \times S^{\mathfrak{D}}$ satisfying*

$$(s, r) \sim_f (s', r') \iff s = s' \textit{ and } s_0^{\mathfrak{D}} \sim_f (s_0^{\mathfrak{D}}, r)\ \forall r \in \mathcal{R}^{\mathfrak{D}}.$$

Let $[s]_\sim$ denote the equivalence class wrt. $\sim_f$ containing state $s \in S^{\mathfrak{D}}$.

Forgetting induces the quotient MDP *$M^{\mathfrak{D}}_{\sim} = (S^{\mathfrak{D}}_{\sim}, [s_0^{\mathfrak{D}}]_\sim, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}}_{\sim})$, where*

$$\mathcal{P}^{\mathfrak{D}}_{\sim}([s]_\sim, a_r)([s']_\sim) = \mathfrak{P}(r)(s)(s').$$

Fig. 2.4 (right) illustrates the quotient MDP under forgetting for our example. Recall that in the quotient MDP the available actions allow to switch realisations and thereby create induced MCs different from any MC in $\mathfrak{D}$. We can naturally formalise the notion of a consistent scheduler with respect to the parameters, i.e., a scheduler does not allow to switch realisations. Therefore, enumerating all consistent schedulers for $M^{\mathfrak{D}}_{\sim}$ and analysing the induced MC provides a solution to both synthesis problems. However, our experiments demonstrate that this is also very inefficient.

Refinement Loop. The key observation leading to the refinement-based synthesis is that model checking of $M_{\sim}^{\mathfrak{D}}$ still provides useful information for the analysis of the family $\mathfrak{D}$. Consider a feasibility synthesis problem for $\varphi = \mathbb{P}_{\leq\lambda}(\varphi)$. If $\texttt{Prob}^{\max}(M_{\sim}^{\mathfrak{D}}, \varphi) \leq \lambda$, then all realisations of $\mathfrak{D}$ satisfy φ. On the other hand, $\texttt{Prob}^{\min}(M_{\sim}^{\mathfrak{D}}, \varphi) > \lambda$ implies that there is no realisation satisfying φ. If λ lies between the $\min$ and $\max$ probability, and the scheduler inducing the $\min$ probability is not consistent, we cannot conclude anything yet, i.e., the abstraction is too coarse. A natural countermeasure is to refine the abstraction $M_{\sim}^{\mathfrak{D}}$, in particular, split the set of realisations leading to two synthesis sub-problems.

The splitting operation is the core of the proposed abstraction-refinement algorithms for all synthesis problems. Note that, we avoid rebuilding the quotient MDP in each iteration, which is crucial for the overall performance. Instead, we only restrict the actions of the MDP to the particular subfamily. The particular refinement algorithms leverage the similar ideas as the refinement strategies for parameter synthesis described in Section 2.1.1. More details can be found in synthesis [ČJJK19] including heuristics for finding effective splitting strategy that reduces the number of model-checking calls.

2.2.3 Counter-example driven synthesis

In this section, we present an alternative approach for topology synthesis that adopts counter-example driven synthesis [SLTB+06, ABD+15, SLRBE05, ADK+18] to probabilistic domains. We follow the typical separation of concerns as in oracle-guided inductive synthesis [ASFS18, GCT18, GPS17]: a *synthesiser* selects single realisations r that have not been considered before, and a *verifier* checks whether the MC D_r satisfies the specification φ (cf. Fig. 2.5). If D_r violates the specification, the verifier derives a counterexample (CE) in the form of a *conflict* representing the core part of D_r causing the violation. Rather than checking all instantiations, the conflict is used to potentially rule out many instantiations (dashed area) at once and thus to prune the design space.

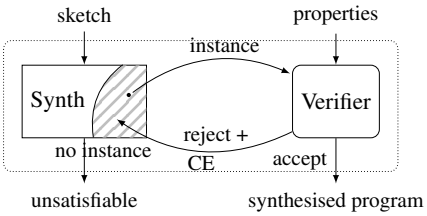
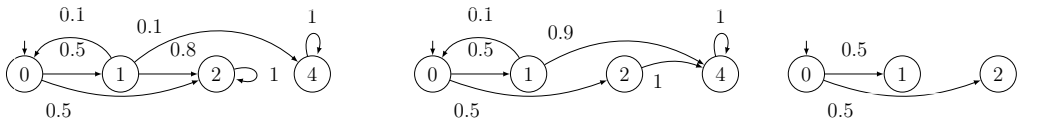


Figure 2.5: CEGIS for synthesis.

Definition 5 (Conflict) *Let $r \in \mathcal{R}^{\mathfrak{D}}$ be a realisation with $D_r \not\models \varphi$. A partial realisation $\bar{r}_\varphi \subseteq r$ is a* conflict *for the property φ iff $D_{r'} \not\models \varphi$ for each realisation $r' \supseteq \bar{r}_\varphi$. A set of conflicts is called a* conflict set.

To explore all realisations for the feasibility synthesis, the synthesiser starts with Q representing all realisations $\mathcal{R}^{\mathfrak{D}}$. It picks some realisation $r \in Q$. It then calls a verifier to decide whether $D_r \models \varphi$. If do so, r retuned as feasible realisation. Otherwise, the verifier computes a counter-example in the form a conflict. Then the conflict is generalised to a set of realisations that violate φ and Q is pruned by removing all these realisations. If Q is empty, we are done: each realisation violates φ. Note that the more complicate synthesis algorithms use the conflicts for different properties: The threshold synthesis algorithm uses both φ and $\neg\varphi$ to effectively identify all realisations satisfying and violating φ, respectively. The optimal synthesis algorithm iteratively updates φ based on the current optimal value achieved for φ.



(a) D_{r_1} ($r_1(k_2) = r_1(k_3) = 2$) (b) D_{r_2} ($r_2(k_2) = 2, r_2(k_3) = 4$) (c) Fragment of D_{r_1}

Figure 2.6: Fragment and corresponding sub-MC that suffices to refute φ

The verifier typically uses an off-the-shelf probabilistic model-checking procedure to determine if φ is violated and to computes a critical set of states C of D_r that induce *sub-MCs* (denoted as $D_r \downarrow C$) violating φ. The critical sets for safety properties can be obtained via standard methods [ÁBD+14], and support for liveness properties is discussed in [ČHJK19]. The essential property [WJÁ+12] of the sub-MCs is:

If a sub-MC of a MC D refutes a safety property φ, then D refutes φ too.

Finally, the verifier translates the obtained critical set C for realisation r to a conflict *Conflict*$(C, r) \subseteq r$ and stores it in the conflict set that is used to prune the design space. The following examples illustrate the key idea behind our approach.

Example 4 *Consider the family of MCs $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ where $S = \{0, \ldots, 4\}$, $s_0 = 0$, and $K = \{k_0, \ldots, k_5\}$ with $T_{k_0} = \{0\}$, $T_{k_1} = \{1\}$, $T_{k_2} = \{2, 3\}$, $T_{k_3} = \{2, 4\}$, $T_{k_4} = \{3\}$ and $T_{k_5} = \{4\}$, and $\mathfrak{P}$ given by:*

$$\mathfrak{P}(0) = 0.5 : k_1 + 0.5 : k_2 \quad \mathfrak{P}(1) = 0.1 : k_0 + 0.8 : k_3 + 0.1 : k_5 \quad \mathfrak{P}(2) = 1 : k_3$$
$$\mathfrak{P}(3) = 1 : k_4 \quad \mathfrak{P}(4) = 1 : k_5$$

and property $\varphi := \mathbb{P}_{\leq 2/5}(\Diamond\{2\})$. Assume the synthesiser picks realisation r_1 with $r_1(k_2) = 2, r_1(k_3) = 2$. The verifier builds D_{r_1}, depicted in Fig. 2.6(a), and determines $D_{r_1} \not\models \varphi$. Observe that the verifier does not need the full realisation D_{r_1} to refute φ. In fact, the paths in the fragment of D_{r_1} in Fig. 2.6(c) (ignoring the outgoing transitions of states 1 and 2) suffice to show that the probability to reach state 2 exceeds $2/5$.

Consider realisation r_2 with $r_2(k_2) = 2, r_2(k_3) = 4$. Observe that $D_{r_1} \downarrow C$ is part of D_{r_2} too. Formally, the sub-MC of $D_{r_2} \downarrow C$ is isomorphic to $D_{r_1} \downarrow C$ and therefore also violates φ – it can be pruned without constructing and verifying D_{r_2}.

2.2.4 Syntax-guided synthesis

Probabilistic models are typically specified by means of a program-level modelling language, such as PRISM [KNP11], JANI [BDH+17], or MODEST [BDHK06]. We propose a *sketching language* based on the PRISM modelling language. A sketch, a syntactic template, defines a high-level structure of the model and represents a-priori knowledge about the system under development. It effectively restricts the size of the design space and also allows to concisely add constraints and costs to its members. The proposed language is easily supported by model checkers and in particular by methods for generating CEs [DJW+14, WJV+15].

```
hole X either { XA is 1 cost 3, 2}
hole Y either { YA is 1, 3 }
hole Z either { 1, 2 }
constraint  !(XA && YA);
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=X + 0.5: s'=Y;
s = 1 -> s'=s+Z;
s >= 2 -> s'=s;
endmodule
```

(a) Program sketch $\mathfrak{S}_H$

```
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=1 + 0.5: s'=3;
s = 1 -> s'=3;
s >= 2 -> s'=s;
endmodule
```

(b) Realisation R ($\{X{\mapsto}1, Z{\mapsto}2, Y{\mapsto}3\}$)

Figure 2.7: Running example

In this section, we describe the sketching language and briefly discuss how to adapt the aforementioned synthesis techniques from state level to program level.

A program sketching language Recall that the PRISM program consists of one or more reactive modules that may interact with each other[4]. A program has a set of bounded variables spanning its state space. Transitions between states are described by guarded commands of the form:

$$\texttt{guard} \;\rightarrow\; p_1 : \texttt{update}_1 + \ldots\ldots + p_n : \texttt{update}_n$$

The guard is a Boolean expression over the module's variables of the model. If the guard evaluates to true, the module can evolve into a successor state by updating its variables. An update is chosen according to the probability distribution given by expressions $p_1, \ldots, p_n$. In every state enabling the guard, the evaluation of $p_1, \ldots, p_n$ must sum up to one. Overlapping guards yield non-determinism and are disallowed here.

Roughly, a program $\mathfrak{P}$ thus is a tuple (Var, E) of variables and commands. For a program $\mathfrak{P}$, the *underlying MC* $[\![\,\mathfrak{P}\,]\!]$ are $\mathfrak{P}$'s semantics. We lift specifications: Program $\mathfrak{P}$ satisfies a specification Φ, iff $[\![\,\mathfrak{P}\,]\!] \models \Phi$, etc.

A sketch $\mathfrak{S}_H$ is a program that contains *holes* $h \in H$. Holes are the program's open parts and can be replaced by one of finitely many options. Each option can *optionally* be named and associated with a cost. They are declared as:

$$\texttt{hole}\; h \;\texttt{either}\; \{\, x_1 \;\texttt{is}\; \texttt{expr}_1 \;\texttt{cost}\; c_1, \ldots, x_k \;\texttt{is}\; \texttt{expr}_k \;\texttt{cost}\; c_k \,\}$$

where h is the hole identifier, x_i is the option name, $\texttt{expr}_i$ is an expression over the program variables describing the option, and c_i is the cost, given as expressions over natural numbers. A hole h can be used in commands in a similar way as a constant, and may occur multiple times within multiple commands, in both guards and updates. The option names can be used to describe constraints on realisations. These propositional formulae over option names restrict realisations.

The *sketch realisation* R is a function that satisfies all constraints and that yields a program (without holes) in which each hole $h \in H$ i is replaced by $R(h)$.

[4]To simply the presentation, we consider only single module programs – this is not a restriction, every PRISM program can be flattened into this form.

```
const int X = 1, Y = 3;
...
module rex
s : [0..3] init 0;
s=0 -> 0.5: s'=X + 0.5: s'=Y;
endmodule
```

(a) CE for upper bound

```
...
module rex
s : [0..3] init 0;
s=0 -> 0.5:s'=X + 0.5:s'=Y;
s=3 -> s'=3
endmodule
```

(b) CE for lower bound

Figure 2.8: CEs for (a) $\mathbb{P}_{\leq 0.4}[F\ \mathtt{s=3}]$ and (b) $\mathbb{P}_{>0.6}[F\ \mathtt{s=2}]$.

Example 5 *We consider a small running example to illustrate the main concepts of the sketching language. Fig. 2.7(a) depicts the program sketch $\mathfrak{S}_H$ with holes $H = \{X, Y, Z\}$. For X, the options are $\{1, 2\}$. The constraint forbids XA and YA both being one; it ensures a non-trivial random choice in state s=0. Fig. 2.7(b) shows realisation $R = \{X \mapsto 1, Z \mapsto 2, Y \mapsto 3\}$.*

Program-level MDP-based synthesis Using the sketching language for the MDP-based synthesis is quite straightforward as the key concepts in the families of MC and sketches are analogous. In particular: holes and parameters are similar, parameter domains are options, and family realisations and sketch realisations both yield concrete instances from a family/sketch. In this case, the main benefit of the sketching language is a high-level and compact representation of the design space.

Program-level CEGIS To properly benefit from the program-level CEGIS, we need to first introduce *program-level counter-examples* and introduce adequate encoding and pruning of the program-level design space.

Definition 6 *For program $\mathfrak{P} = (\mathrm{Var}, E)$ and specification φ with $\mathfrak{P} \not\models \varphi$, a program-level CE $E' \subseteq E$ is a set of commands, such that for all (non-overlapping) programs $\mathfrak{P}' = (\mathrm{Var}, E'')$ with $E'' \supseteq E'$ (i.e, extending $\mathfrak{P}'$), $\mathfrak{P}' \not\models \varphi$.*

Example 6 *Reconsider $\varphi = \{\mathbb{P}_{\leq 0.4}[\Diamond\ s{=}3]\}$. Figure 2.8(a) shows a CE for realisation R from Fig. 2.7(b). The probability to reach s=3 in the underlying MC is $0.5 > 0.4$. Fig. 2.8(b) shows a CE for the lower bound property $\mathbb{P}_{>0.6}[F\ s{=}2]$.*

For safety properties, program-level CEs coincide with the notation of high-level CEs proposed in [WJV+15], their extension to liveness properties are discussed in [ČHJK19]. The program-level CEs are computed using the MaxSat approach from [DJW+14].

Program level synthesiser As before, the synthesiser stores and queries the set of realisations not yet pruned. These remaining realisations are represented by (the satisfying assignments of) the first-order formula Ψ over hole-assignments. The synthesiser first constructs Ψ such that it represents *all* sketch realisations that satisfy the constraints in the sketch $\mathfrak{S}_H$. Then it iteratively extendes Ψ with conjunctions representing the conflicts and thus prunes the remaining design space. The synthesiser exploits an SMT-solver for linear (bounded) integer arithmetic to obtain a realisation R consistent with Ψ, or `Unsat`

if no such realisation exists. As long as a new realisations is found, the verifier analyses it and returns a conflict set C, if φ is violated, that is used to extend Ψ. More details as well as the concrete encoding schemes can be found in [ČHJK19].

2.2.5 Experimental evaluation for the topology synthesis

A detailed experimental evaluation of the proposed approaches for the topology synthesis can be found in the original papers [ČJJK19, ČHJK19]. Below, we try to summarise the key observations assessing both the weakness and the strengths of the approaches.

Benchmarks description. We consider the following case studies: *Maze* is a planning problem typically considered as POMDP, e.g. in [NPZ17]. *Grid* is another classical benchmark for solving partially observable POMDPs [KLC98]. *Pole* considers balancing a pole in a noisy and unknown environment (motivated by [ABC$^+$18, CCM$^+$12]). *Herman* is an asynchronous encoding of the distributed Herman protocol for self-stabilising rings [Her90, KNP12]. *DPM* considers a partial information scheduler for a disk power manager motivated by [BBPM00, GCT18]. *BSN* (Body sensor network, [RAN$^+$15]) describes a network of connected sensors that identify health-critical situations – the largest software product line benchmark used in [CDKB18]. *Intrusion* describes a network (adapted from [KNPV09]), in which the controller tries to infect a target node via intermediate nodes. The benchmarks adequately cover various synthesis problems ranging from design spaces including few hundreds realisations (e.g. *Herman* or *BSN*) to several millions (e.g. *Intrusion*). Also the complexity of the realisations varies significantly: the size of the underlying MCs ranges from few hundred states (e.g. *Intrusion* or *Maze*) to hundred thousands states (e.g. *Grid*).

We primarily compare the performance of the proposed approaches with respect to the enumerative approach that linearly depends on the number of realisations, and the underlying MCs' size. We also consider synthesis via verification of *all-in-one* MDPs [GS13, RAN$^+$15, CDKB18] (recall Def. 3), however, in many cases building the MDP is not feasible due to it prohibited size. We focus on synthesis problems where all realisations are explored, as relevant for the threshold and optimal synthesis, since enumerative methods perform mostly independent of the order of enumerating realisations.

General observations. *Sketching.* Families are simpler objects than sketches, but their explicit usage of states make them inadequate for modelling. Moreover, additional features like program-level conflicts significantly ease the modelling process. Consider *intrusion*: Without constraints, the number of realisations grows to $6{\cdot}10^{11}$. Put differently, the constraint allows to discard over 99.99% of the realisations up front. Moreover, constraints can exclude realisations that would yield unsupported programs, e.g, programs with infinite state spaces. While modelling concise sketches with small underlying MCs, it may be hard to avoid such invalid realisations without the use of constraints.

Specification. The performance of both proposed approaches significantly depends on the specification, namely, on the thresholds appearing in the properties. The abstraction-

refinement synthesis benefits from thresholds being closer to the optima as the abstraction requires a smaller number of iterations, which directly improves the performance. Contrary, for CEGIS we observe the strong dependency between performance and "unsatisfiability" – this is not surprising as more unsatisfiable the property is, the smaller the conflicts exist (as in [DJW+14]) which prune more realisations and also they are typically found faster than large ones. Note that CEGIS generally performs better with specifications that have multiple (conflicting) properties: Different realisations can be effectively pruned by conflicts for different properties.

Structure of the design space significantly affects other important aspects of the synthesis process. For the abstraction-refinement synthesis, these aspects include: the number of iterations in the refinement loop, the size of the abstraction compared to the average size of the realisation, and optimality of the refinement strategy. For CEGIS, the locality of the holes are essential: it performs significantly better on sketches with holes that lie in local regions of the MC. Holes relating to states all-over the MC are harder to prune.

The size of the underlying MCs has similar effects on all considered synthesis algorithms and thus it does not effect their relative performance.

Speedup with respect to the enumerative approach. As explain above, the performance of the proposed two approaches significantly depends on the thresholds appearing in the specification. This dependency is orthogonal with respect to the abstraction-refinement synthesis and CEGIS. The following table presents ranges of speedups we achieved using both approaches for different thresholds. We can see that, in some cases, we achieved enormous speedup over 1000 which fundamentally pushes forward the frontiers of practically feasible synthesis. In other cases, the speedup is more modest. For *Grid*, the MCs' topology and the few commands make pruning hard. Moreover, the speedup is mostly independent of the threshold and thus abstraction-refinement also performs poorly. The run time for *BSN*, with a small $|\mathfrak{D}|$ is actually significantly affected by the initialisation of various data structures; thus only a small speedup is achieved. The slowdown for *Herman* is caused by mainly by sub-optimality of the refinement strategy and non-local holes.

problem	speedup
Maze	[46, 26K]
Grid	[2, 3]
Pole	[77, 2.2K]
Herman	[0.2, 0.3]
DPM	[3-100]
BSN	[1-3]
Intrusion	[5, 2.2K]

2.3 Synthesis of Robust Systems via Parametric Analysis

Robustness is a key characteristic of both natural [Kit04] and human-made [Pha95] systems. Despite significant advances in software performance and reliability engineering (see e.g. [Bon14]), the quality attributes of software systems are typically analysed for point estimates of stochastic system parameters such as component service rates or failure probabilities. Even the techniques that assess the sensitivity of quality attributes to parameter changes (e.g. [FTG16]) focus on the analysis of a given design at a time instead of systematically designing robustness into the system under development (SUD).

To address these limitations, we propose a tool-supported method for the efficient synthesis of parametric continuous-time Markov chains (pCTMCs) that correspond to robust system under designs (SUDs). Our RObust DEsign Synthesis (RODES) method generates sets of pCTMCs that: i) are resilient to pre-specified *tolerances* in the SUD parameters, i.e., to changes in the SUD's operational profile, ii) satisfy strict performance, reliability and other quality constraints, and iii) are nearly Pareto optimal with respect to a set of quality optimisation criteria.

To the best of our knowledge, our tool RODES [CČG+17b][5] provides the first end-to-end, tool-supported design method for the generation of sensitivity-aware Pareto fronts. It integrates search-based multi-objective synthesis and GPU-accelerated precise pCTMC parameter synthesis described in Section 2.1.1.

Sensitivity analysis has long been used to assess the impact that changes in the parameters of the system under development have on the system performance, reliability and other quality attributes, e.g. in [GT02, HL05, LHC+05]. However, these approaches work by repeatedly sampling the parameter space of the system and evaluating the system behaviour for the sampled values. Accordingly, their results are not guaranteed to capture the entire range of quality-attribute values for the parameter region of interest. Our method overcomes this limitation by generating safe and close over-approximations of the quality-attribute regions associated with robust designs.

The sensitivity of software operational profiles has been analysed using the perturbation theory for Markov processes [KGP03], to quantify the effect of variations in model transition probabilities. However, this approach does not synthesise the solutions, and does not work with the wide range of parameters supported by our method.

Techniques for the

2.3.1 Design space modelling and specification

We use a *parametric continuous-time Markov chain* (pCTMC) to define the design space of a SUD. To this end, we extend pCTMCs defined in Section 2.1, where only real-valued parameters $k \in K$ determining the transition rates of the Markov chain are considered, and assume that a pCTMC also includes discrete parameters $d \in D$ affecting its state space. Our definition captures the need for both discrete parameters encoding architectural structural information (e.g. by selecting between alternative implementations of a software component) and continuous parameters encoding configurable aspects of the system (e.g. network latency or throughput). A design space is thus model as a pCTMC $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ over a continuous parameter space $\mathcal{P} = \bigtimes_{k \in K} [k^{\perp}, k^{\top}]$ and a discrete parameter space $\mathcal{Q}$. As such, a candidate system design $\mathcal{C}(\mathcal{P}', q)$ is a pCTMC corresponding to a fixed discrete parameter valuation q and to continuous parameter values from a (small) region $\mathcal{P}' \subset \mathcal{P}$.

In our approach, we operate with pCTMCs expressed in a high-level modelling language extending the PRISM language [KNP11] and adopting constructs from [GCT18] for specifying the discrete and continuous parameters (see [CČG+18] for more details).

[5] Available preinstalled on an easy-to-use VirtualBox instance from our project website `https://github.com/gerasimou/RODES/wiki`.

This modelling language can be seen as a very simple sketching language, describing a space of *candidate designs*, for probabilistic systems. and can be easily extended towards a more flexible sketching language we described in Section 2.2.4.

Quality requirements. The quality requirements of a SUD with design space given by a pCTMC $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ are defined using bounded-time CSL formulas as follows:

1) A finite set of objective functions $\{f_i\}_{i\in I}$ corresponding to quality attributes of the system and defined in terms of a set of CSL path formulas $\{\varphi_i\}_{i\in I}$ and the corresponding satisfaction functions Λ_{φ_i}.
2) A finite set of Boolean constraints $\{c_j\}_{j\in J}$ corresponding to the set of CSL path formulas $\{\psi_j\}_{j\in J}$ and thresholds $\{\sim_j r_j\}_{j\in J}$ on the functions Λ_{ψ_j}.

Due to the continuous parameter space, a single candidate design induces an infinite number of objective function values, from which the designer must choose a representative value. For a candidate design $\mathcal{C}(\mathcal{P}', q)$ and objective f_i, this is typically identified as one of the minimum, maximum and mid-range value of f_i over all $p \in \mathcal{P}$. On the other hand, constraints have a unique interpretation because they must be met for any parameter value of a candidate design[6].

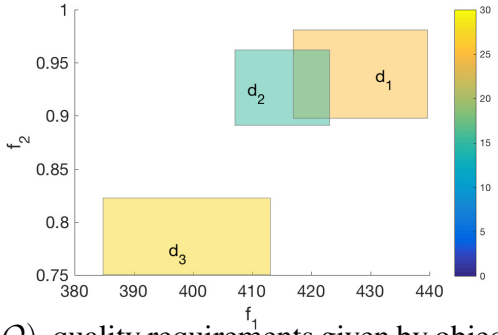
2.3.2 Sensitivity-aware synthesis

Quantifying the sensitivity of candidate designs is a crucial step in our robust synthesis method. Intuitively, the sensitivity of a design $\mathcal{C}(\mathcal{P}', q)$ captures how the objective functions $\{f_i\}_{i\in I}$ change in response to variations in the continuous parameters $k \in K$. The variation of each objective f_i is measured by the length of the interval describing the range of admissible values for f_i and $\mathcal{C}(\mathcal{P}', q)$. The degree of variation for multiple objectives is given by the product of interval lengths, i.e., the volume of the corresponding quality-attribute region. The sensitivity takes also into account the size of the underlying parameter region, in order to account for designs with different tolerance values γ_k for $k \in K$. For instance, a design with a large quality-attribute volume and high tolerance (large parameter region volume) must be considered *more robust* (less sensitive) than another design with comparable quality-attribute volume but lower tolerance. We illustrate the sensitivity in the following example (for the formal definition, see [CČG+18]).

Example 7 (Sensitivity) *Consider the candidate designs d_1, d_2, d_3 with tolerance $\gamma = 0.005$ and the objective functions f_1 and f_2.*

[6]Without loss of generality, we will further assume that all objective functions $\{f_i\}_{i\in I}$ are minimised and that all thresholds $\{\sim_j r_j\}_{j\in J}$ are upper bounds of the form of $\leq r_j$.

The three designs can be visualised in the quality-attribute space *(i.e. the objective space), as shown in the figure, providing a direct and intuitive way to assess robustness via the volume of the boxes. The figure indicates that d_2 is the most robust design (with the smallest sensitivity value).*



Consider a system with design space $\mathcal{C}(\mathcal{P},\mathcal{Q})$, quality requirements given by objective functions $\{f_i\}_{i\in I}$ and constraints $\{c_j\}_{j\in J}$, and designer-specified tolerances $\{\gamma_k\}_{k\in K}$ for the continuous parameters of the system. Also, let $\mathcal{F}$ be the set of feasible designs for the system, i.e., of candidate designs that meet the tolerances $\{\gamma_k\}_{k\in K}$ and satisfy the constraints $\{c_j\}_{j\in J}$.

Definition 7 *A* sensitivity-aware Pareto dominance relation *over a feasible design set $\mathcal{F}$ and a set of minimisation objective functions $\{f_i\}_{i\in I}$ is a relation $\prec\subset\mathcal{F}\times\mathcal{F}$ such that for any feasible designs $d,d'\in\mathcal{F}$*

$$\begin{aligned} d\prec d' \iff\ & \big(\forall i\in I.\, f_i(d)\le f_i(d')\ \wedge\ \exists i\in I.\,(1+\epsilon_i)f_i(d)<f_i(d')\big)\ \vee \\ & \big(\forall i\in I.\, f_i(d)\le f_i(d')\ \wedge\ \exists i\in I.\, f_i(d)<f_i(d')\ \wedge \\ & \qquad\qquad sens(d)\le sens(d')\big). \end{aligned} \tag{2.2}$$

where $\epsilon_i\ge 0$ are sensitivity-awareness parameters. *Note that, the sensitivity-aware Pareto dominance relation is a strict order.*

The classical Pareto dominance definition can be obtained by setting $\epsilon_i=0$ for all $i\in I$ in (2.2). When $\epsilon_i>0$ for some $i\in I$, dominance with respect to quality attribute i holds in our generalised definition in two scenarios:

1) when the quality attribute has a much lower value for the dominating design
2) when in addition to a (slightly) lower quality attribute value, the sensitivity of the dominating design is no worse than that of the dominated design.

These scenarios are better aligned with the needs of designers than those obtained by using sensitivity as an additional optimisation criterion, which induces Pareto fronts comprising many designs with low sensitivity but unsuitably poor quality attributes. Importantly, for $\epsilon_i > 0$ our generalised definition induces Pareto fronts comprising designs with non-optimal (in the classical sense) objective function values, but with low sensitivity. We call such designs *sub-optimal robust*. Thus, ϵ_i can be finely tuned to sacrifice objective function optimality (slightly) for better robustness.

The *parametric Markov chain synthesis problem* consists of finding the Pareto-optimal set PS of candidate designs (i.e. pCTMCs) with tolerances $\{\gamma_k\}_{k\in K}$ that satisfy the constraints $\{c_j\}_{j\in J}$ and are *non-dominated* with respect to the objective functions $\{f_i\}_{i\in I}$ and the sensitivity-aware dominance relation '$\prec$':

$$PS=\big\{\mathcal{C}(\mathcal{P}',q)\in\mathcal{F}\ \big|\ \nexists\mathcal{C}(\mathcal{P}'',q')\in\mathcal{F}.\ \mathcal{C}(\mathcal{P}'',q')\prec\mathcal{C}(\mathcal{P}',q)\big\}, \tag{2.3}$$

Metod overview. Computing the Pareto-optimal design set (2.3) is a very complex and time-demanding process as the design space $\mathcal{C}(\mathcal{P}, \mathcal{Q})$ is extremely large, in fact, it is uncountable due to its real-valued parameters. Also, every candidate design $\mathcal{C}(\mathcal{P}', q)$ consists of an infinite set of CTMCs that cannot all be analysed to establish its quality and sensitivity. To address these challenges, our pCTMC synthesis method combines search-based software engineering (SBSE) techniques [HMZ12, GCT18] with our techniques for effective pCTMCs analysis presented in Section 2.1. Using these techniques, our approach is able to produce a close approximation of the Pareto-optimal design set that is stored in $\overline{PS}$.

In each iteration of the synthesis loop, the method uses an SBSE metaheuristic to get a new set of candidate designs and then updates the approximate Pareto-optimal design set $\overline{PS}$. The metaheuristic is implemented as multiobjective optimisation genetic algorithm such as NSGA-II [DPAM02] or MOCell [NDL$^+$09] that are specifically tailored for the synthesis of close Pareto-optimal set approximations that are spread uniformly across the search space. This update involves analysing each candidate design $d = \mathcal{C}(\mathcal{P}', q)$ to establish its associated objective function and constraint values. The update function employs the GPU-accelerated parameter analysis techniques to compute safe enclosures of the satisfaction probability of CSL formulae over pCTMCs.

2.3.3 Case study: Google file system

We consider the design of Google File System (GFS), the replicated file system used by Google's search engine [GGL03, BHH$^+$13]. GFS partitions files into chunks of equal size, and stores copies of each chunk on multiple *chunk servers*. A master server monitors the locations of these copies and the chunk servers, replicating the chunks as needed. During normal operation, GFS stores CMAX copies of each chunk. However, as servers fail and are repaired, the number c of copies for a chunk may vary from 0 to CMAX.

We assume that GFS designers must select the hardware failure and repair rates cHardFail and cHardRepair of the chunk servers, and the maximum number of chunks NC stored on a chunk server. These parameters reflect the fact that designers can choose from a range of physical servers, can select different levels of service offered by a hardware repair workshop, and can decide a maximum workload for chunk servers. We consider an initial system state modelling a severe hardware disaster with all servers down due to hardware failures and all chunk copies lost, and we formulate a pCTMC synthesis problem for quality requirements given by two maximising objectives and one constraint.
Objective f_1 maximises the probability that the system recovers service level 1 (master up and at least one chunk copy available) in the time interval $[10, 60]$ hours. Objective f_2 maximises the expected time the system stays in (optimal) states with at least $0.5M$ chunk servers up in the first 60 hours of operation. Finally, constraint c_1 restricts the number of expected chunk replications over 60 hours of operations.

Figure 2.9 shows the Pareto fronts obtained using the "lower bound" for the objective functions. The design-space representation is given in Figure 2.10. We observe that the Pareto front for $\epsilon = 0$ and $\gamma = 0.005$ contains several large (yellow) boxes that correspond to highly sensitive designs. For $\epsilon \in \{0.05, 0.1\}$, these poor designs are "replaced"

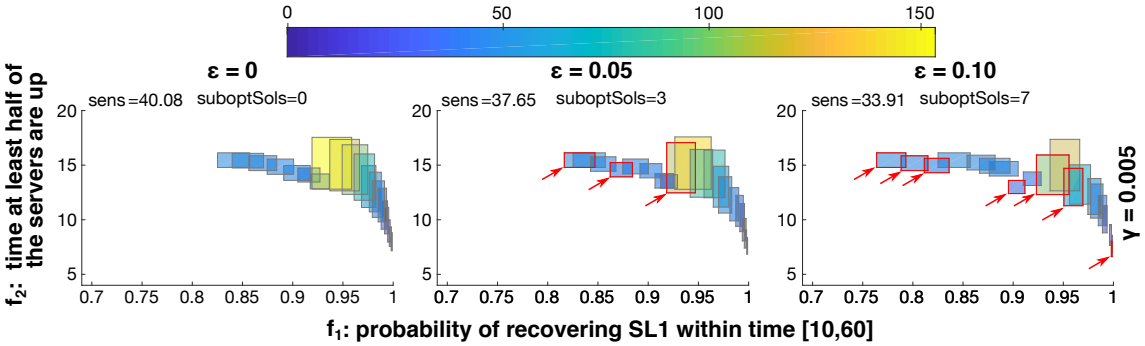


Figure 2.9: Sensitivity-aware Pareto fronts for the GFS model. Boxes represent quality-attribute regions, coloured by sensitivity (yellow: sensitive, blue: robust). Red-bordered boxes indicate sub-optimal robust designs. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box). Statistics are: sens, average sensitivity of the front; suboptSols, number of suboptimal solutions.

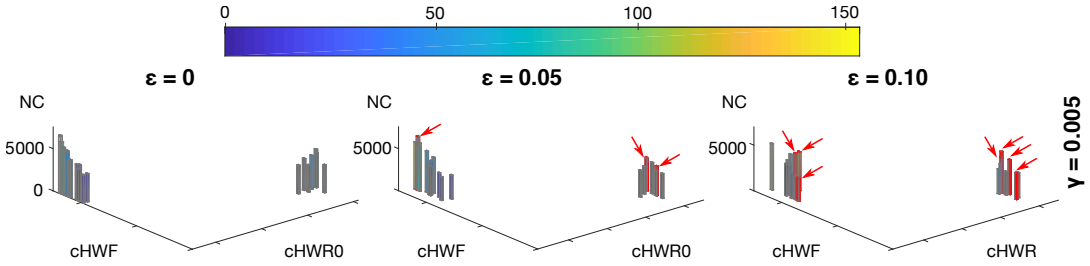


Figure 2.10: Synthesised Pareto-optimal designs for the GFS model. Rectangles in x-y plane correspond to the continuous parameter regions.

by robust designs – surrounded by red borders – with very similar quality attributes but slightly sub-optimal.

The design-space view of Figure 2.10 evidences a trade-off between cHardFail and cHardRepair, i.e., optimal designs tend to have either high failure rates and high repair rates, or low failure and repair rates. Results reveal that there is actually an ideal ratio between the two parameters as the corresponding optimal design appear to keep a relatively constant proportion between cHardFail and cHardRepair. This result was unexpected, yet very useful, since it indicates that designs not satisfying this trade-off yield excessively fast or slow recovery times, and thus are far from the optimal f_1 values.

Further, we observe that the maximum number of chunks per server, NC, has a major influence on the design robustness, with high NC values leading to highly sensitive designs. These designs should be avoided in favour of the designs with low NC values.

2.4 Future Research Directions

Our results have significantly extended capabilities of automated methods for designing probabilistic systems, but also opened new promising research directions including: i) combined strategies for the topology synthesis that would efficiently integrate the MDP abstraction, counter-example pruning and evolutionary search strategies, ii) complete synthesis methods that would effectively handle both types of parameters (i.e. parameters effecting transition probability/rate and parameters affecting the system topology), and iii) complete topology synthesis for infinite families and sketches – this problem is in general undecidable, but can be feasible for certain classes of families and sketches.

Further, we will focus on more flexible sketching languages allowing us to deploy our synthesis methods and tools into a broader class of application domains.

2.5 Contributed Papers

[BČDŠ13] **Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *CAV'13*, volume 8044 of *LNCS*, pages 107–123. Springer, 2013.**

[ČPP+16] Milan Češka, Petr Pilař, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. PRISM-PSY: Precise GPU-accelerated parameter synthesis for stochastic systems. In *TACAS'16*, volume 9636 of *LNCS*, pages 367–384. Springer, 2016.

[ČDP+17] **Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017.**

[CČG+17] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Designing robust software systems through parametric Markov chain synthesis. In *ICSA'17*, pages 131–140. IEEE Computer Society, 2017.

[CČG+17b] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. RODES: A robust-design synthesis tool for probabilistic systems. In *QEST'17*, volume 10503 of *LNCS*, pages 304–308. Springer, 2017.

[CČG+18] **Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software*, 143:140–158, 2018.**

[ČJJK19] **Milan Češka, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Shepherding hordes of Markov chains. In *TACAS'19*, volume 11428 of *LNCS*, pages 172–190. Springer, 2019.**

[ČHJK19] **Milan Češka, Christian Hensel, Sebastian Junges, and Joost-Pieter Katoen. Counterexample-driven synthesis for probabilistic program sketches. In *Formal Methods – The Next 30 Years*, volume 11800 of *LNCS*, pages 101–120. Springer, 2019.**

The highlighted papers are attached to this thesis.

Chapter 3

Analysis and Synthesis of Chemical Reaction Networks

This chapter presents our results in the area of analysis and synthesis of stochastic biochemical systems. We focus on systems described as a *chemical reaction network*(CRN). CRNs represent a convenient formalism for modelling a multitude of biological systems, including molecular signalling pathways, gene regulation, and logic gates built from DNA. For low molecule counts, and assuming a well-mixed and fixed reaction volume, the prevailing approach is to model such networks using continuous-time Markov chains (CTMCs) [Gil77a]. Stochastic model checking [KNP07] allows the analysis of the model behaviour against temporal logic properties. We envision biochemical devices that implement biosensors and medical diagnostic systems, and hence ensuring appropriate levels of reliability is important.

Our results targets at two fundamental challenges in modelling and analysing CRNs: *uncertainty of the model parameters* affecting both reaction rates and network topology, and *scalability* of existing techniques limiting the class of systems that can be effectively analysed. In this thesis, we briefly present the following three results. 1) precise synthesis of reaction rate parameters for CRNs [ČDKP14, ČDP$^+$17], 2) optimal syntax-guided synthesis of CRN topology and parameters [CČF$^+$17], and 3) semi-quantitative abstraction and scalable analysis of complex CRNs [ČK19].

3.1 Parameter Synthesis for Stochastic CRNs

Stochastic analysis of CRNs assumes that the model is fully specified, including reaction rate constants. However, the reaction rates can be unknown or given as estimates that typically include some measurement error. In spite of this uncertainty, one might want to still demonstrate robustness and reliability of a synthetic molecular device. Or, one might be interested in the identification of parameter values that reproduce experimentally observed behaviour. The *parameter synthesis problem* for stochastic CRNs assumes a specification given by a temporal formula and a CRN whose rates are given as functions of parameters. The goal is to compute the parameter valuations guaranteeing that the CRN satisfies the formula. The problem can be formalised using the pCTMCs and the

corresponding satisfaction function (recall Section 2.1). Hence, our precise synthesis algorithms [ČDKP14, ČDP+17] (described in Section 2.1.1) can be naturally employed. In the following sections, we first introduce parametric CRNs and their semantics given by pCTMCs, and then briefly present two biological case studies demonstrating the synthesis algorithm in action.

A principally different approach can be used when a linear-time specification and certain restrictions on the rate function are assumed. In that case, the satisfaction function can be approximated using statistical methods such as Gaussian Process regression, which leverage smoothness [JL11, BMS16]. Inference of parameter values in probabilistic models from time-series measurements is a well studied area of research [AMSW11, BL13, KPS+13, PYH+14], but different from the problem we consider.

3.1.1 Parametric CRNs

A *chemical reaction network (CRN)* $\mathcal{N} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where Λ is a set of *species*, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is a set of reactions. Species in Λ interact according to the reactions in $\mathcal{R}$. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the *parameter* associated with the rate of the reaction. r_τ and p_τ represent the stoichiometry of reactants and products[1]. Under the usual assumption of mass action kinetics, the *stochastic* semantics of a CRN $\mathcal{N}$ is generally given in terms of a discrete-state, continuous-time stochastic process $\mathbf{X}(\mathbf{t}) = (X_1(t), X_2(t), \ldots, X_{|\Lambda|}(t), t \geq 0)$ [EK09].

The behaviour of the parametric stochastic system $\mathbf{X}(\mathbf{t})$ can be described by the (possibly infinite) parametric continuous-time Markov chain (CTMC) $\mathcal{C} = (\mathbf{S}, s_0, \mathbf{R})$ where the parametric transition matrix $\mathbf{R}(i, j)$ gives the probability of a transition from the state s_i to the state to s_j. Formally, $\mathbf{R}(i, j) = \sum_{\tau \in \mathsf{reac}(s_i, s_j)} f_\tau(s_i)$ where $\mathsf{reac}(s_i, s_j)$ denotes all the reactions changing state s_i into s_j and f_τ is the parametric rate function of reaction τ. We focus on *mass-action kinetics* [Gil77a], according to which the rate depends on the parameter k_τ and is proportional to the concentrations of its reactants. Therefore, we assume that f_τ is a polynomial function over the rate parameters.

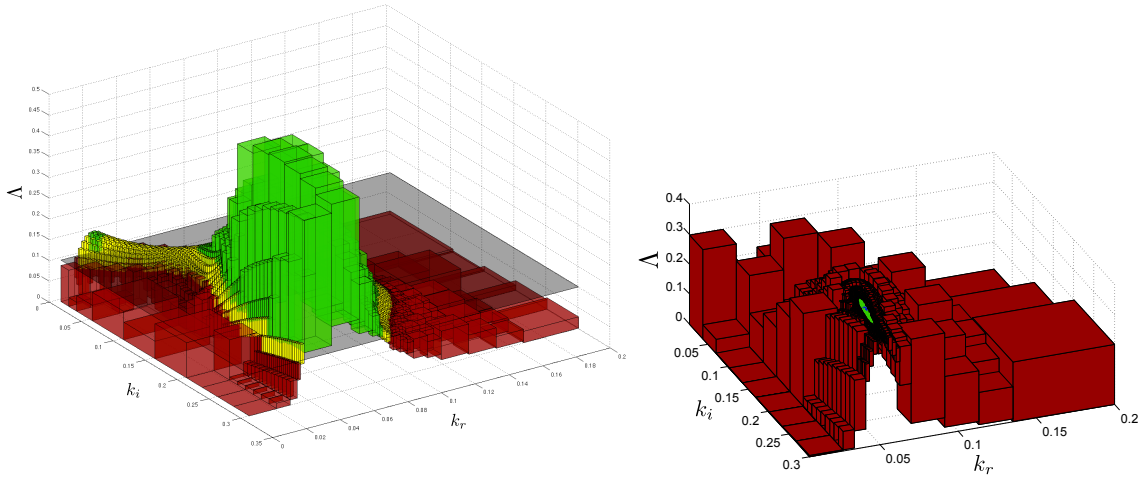
3.1.2 Case Study: Epidemic model

We consider the very famous SIR model [KM32] describing the epidemic dynamics in a well-mixed and closed population of susceptible (S), infected (I) and recovered (R) individuals. In the model, a susceptible individual is infected after a contact with an infected individual with rate k_i. Infected individuals recover with rate k_r, after which they are immune to the infection. We can describe this process with the following CRN with mass action kinetics (i.e. the rate functions are linear with respect to the parameters):

$$\texttt{Infection:} \quad S + I \xrightarrow{k_i} I + I \qquad\qquad \texttt{Recovery:} \quad I \xrightarrow{k_r} R$$

where parameters $k_i \in [0.005, 0.3]$ and $k_r \in [0.005, 0.2]$ are typically affected by a drug treatment. We consider parameters initial populations $S = 95$, $I = 5$, $R = 0$, and the

[1]Given a reaction $\tau_1 = ([1, 1, 0], [0, 0, 2], k_1)$, we often refer to it as $\tau_1 : \lambda_1 + \lambda_2 \xrightarrow{k_1} 2\lambda_3$.



(a) threshold synthesis (b) max synthesis

Figure 3.1: Solution of the synthesis problems for the SIR model and the property ϕ. For threshold synthesis, $r = 10\%$ and the volume tolerance is $\varepsilon = 10\%$. For max synthesis, the probability tolerance is $\epsilon = 1\%$. Colour code is as in Fig. 2.1.

time-bounded CSL path formula $\phi = (I > 0)\mathrm{U}^{[100,120]}(I = 0)$, specifying behaviour where the infection lasts for at least 100 time units, and dies out before 120 time units. Property and parameters are taken from [BMS16], where the authors estimate the satisfaction function for ϕ following a Bayesian approach.

First, we perform threshold synthesis to find infection and recovery rates for which ϕ is satisfied with probability at least $r = 10\%$. Figure 3.1 (a) illustrates the solution. Results evidence that a significantly higher number of refinement steps (around 1.3K) is required for parameter subspaces where the satisfaction function Λ is close to the probability threshold r. Second, we perform max synthesis experiments for the same property ϕ. Results are summarised in Figure 3.1 (b). We observe that, in order to meet the desired probability tolerance, a high number of refinement steps (almost 6K) is required due to a bell-shaped Λ with the maximising region at the top.

Our results significantly improve on the estimation of the satisfaction function Λ obtained by the Bayesian approach [BMS16] . On the other hand, our approach typically has a higher computation cost. In particular, the threshold synthesis for the SIR model took around 29 minutes using a sequential implementation in PRISM-PSY and the max synthesis took 3.6 hours. As described in Section 2.1.2, we have designed and implemented also data-parallel versions of the synthesis algorithms that are able to efficiently utilise modern GPUs. For the SIR model, the parallel implementation in PRISM-PSY achieves up to 10-fold speedup and thus significantly reduce the runtime of synthesis process [ČPP+16].

3.1.3 Case Study: Robustness of Signalling Systems Response

Signalling pathways make the main interface between cells and their environment. Their main role is to monitor biochemical conditions outside the cell and to transfer this infor-

mation into the internal logical circuits (gene regulation) of the cell. Since signal processing is carried out by several dedicated protein complexes (signalling components), it is naturally amenable to intrinsic noise in these protein populations caused by stochasticity of transcription/translation processes. Robust input-output signal mapping is crucial for cell functionality. Many models and experimental studies have been conducted attempting to explain mechanisms of robust signal processing in procaryotic cells, e.g., [SMMA07].

In order to construct robust signalling circuits in synthetically modified procaryotic cells, Steuer et al. [SWSK11] has suggested and analysed a modification of a well-studied two-component signalling pathway that is insensitive to signalling components concentration fluctuations. The study was conducted using a simplified model consisting of the two signalling components each considered in both phosphorylated and unphosphorylated forms. The authors considered two variants of the topology (the difference is in the addition of catalytic activation) and evaluated their robustness considering the average steady-state populations.

In our work, we reformulated the model in the stochastic setting and employed our method for the robustness analysis [ČŠDB14], extending the parameter analysis for CRNs, to provide a detailed analysis of the input-output signal response under fluctuations in population of both signalling components. The biochemical model of both topology variants is given in Figure 3.2. The input signal *S* is considered to be fixed and therefore it makes a constant parameter of the model. The signalling components in both phosphorylated and unphosphorylated forms make the model variables *H, Hp, R*, and *Rp*. More details about the model construction can be found in [ČŠDB14].

General signal transmission reactions			
$H + S \xrightarrow{k_1} Hp + S$			$k_1 = 0.1$
$Hp + R \xrightarrow{k_2} H + Rp$			$k_2 = 0.1$
Basic topology (model 1)		Modified topology (model 2)	
$Rp \xrightarrow{k_{31}} R$	$k_{31} = 1.0$	$Rp + H \xrightarrow{k_{32}} R + H$	$k_{32} = 15.0$
Signalling components degradation			
$H \xrightarrow{k_d} \emptyset$		$Hp \xrightarrow{k_d} \emptyset$	$k_d = 0.01$
$R \xrightarrow{k_d} \emptyset$		$Rp \xrightarrow{k_d} \emptyset$	$k_d = 0.01$
Signalling components synthesis			
$\emptyset \xrightarrow{k_p} R$		$\emptyset \xrightarrow{k_p} H$	$k_p = 0.3$

Figure 3.2: The CRN specifying the biochemical model of the two considered topologies of the two-component signalling pathway.

The key question we want to answer is "Is there a difference in the way the two models handle noise (fluctuations) for low molecular numbers of signalling components?" We formalise our question in terms of the expected reward property at time t where the reward is defined as the *mean quadratic deviation* of the distribution of *Rp*.

We preset here only the main results – for more details see [ČŠDB14]. Fig. 3.3 compers the two variants of model by *Rp* noise robustness. Robustness *Rp* noise in both models has been computed with respect to perturbations of signal *S* over five selected intervals of the input signal $S \in [2,3] \cup [6,7] \cup [10,11] \cup [14,15] \cup [19,20]$ and for three distinct levels of the intrinsic noise in signalling component dynamics represented by sigmoid coefficient $n \in \{0.1, 4.0, 10.0\}$. Perturbations were not computed over the whole interval $(S,n) \in [2,20] \times [0.1, 10.0]$ due to high computational demands. From the computed values of individual refined subspaces as well as from the aggregated robustness values for each input signal interval, we can see that for lower values of signal *S* (up-to 10), Model 2 embodies lower output response noise than Model 1 (spontaneous dephos-

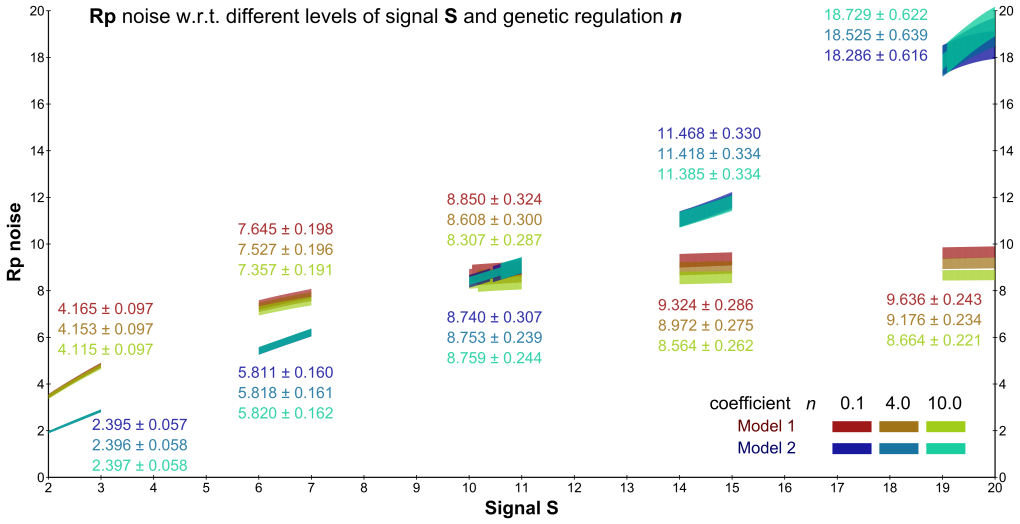


Figure 3.3: Comparison of the two variants of the model by *Rp* noise robustness.

phorylation). While the output response noise in Model 1 tends to converge to values between 8 and 10, Model 2 exhibits a permanent (almost linear) increase in the output response noise over most of the studied portion of the perturbation space. A super-linear increase of the noise is observed for strong input signals. Another interesting aspect is that, with increasing levels of gene regulation given by sigmoid coefficient n, the overall noise in *Rp* decreases over the whole interval of signal values for Model 1 and most of the interval for Model 2. However, there is an anomaly in Model 2 in the high signal region [19.0, 20.0], where with decreasing noise in *R* and *H* the noise in *Rp* increases.

Our study shown that both pathway topologies result in fluctuations of the output response, but robustness of input-output mapping varies in both models with increasing the level of the input signal. For low input signals the synthetic topology gives response with smaller variance in the output, whereas for high input signals the output variance rapidly increases. Therefore the basic topology seems to be more suitable for processing strong signals while the synthetic topology is more appropriate for low level signals. Our study has also shown that both topologies are quite robust with respect to scaling the noise in signalling components dynamics.

Robustness of stochastic biochemical models with respect to the required temporal behaviour has been also studied in [BBNS15]. In contrast to our approach, the authors provide a simulation-based method to define a notion of robust satisfiability in stochastic models. They exploit the average robustness to address the system design problem, where the goal is to optimise (few) control parameters of a stochastic model in order to maximise its robustness.

3.2 Optimal Syntax-Guided Synthesis of CRNs

To further reduce the gap between the design and verification process of CRNs, we considered the problem of automatic construction of CRNs from high-level specifications [CČF+17]. In contrast to the parameters synthesis methods presented in the previous section, we want to synthesise both the CRN topology (i.e. the set of reaction) and the reaction rates. We work again in the setting of *program sketching* [SLRBE05], where the template is a partial program with holes (incomplete information) that are automatically resolved using a constraint solver. We define a sketching language for CRNs that allows designers to not only capture the high-level topology of the network and known dependencies among particular species and reactions, but also to compactly describe parts of the CRN where only limited knowledge is available or left unspecified (partially specified) in order to examine alternative topologies. A *CRN sketch* is therefore a parametric CRN, where the parameters can be unknown species, (real-valued) rates or (integer) stoichiometric constants. Our sketching language is well-suited for biological systems, where partial knowledge and uncertainties due to noisy or imprecise measurements are very common. We associate to a sketch a *cost* function that captures the structural complexity of the CRNs and reflects the cost of physically implementing it using DNA [Car13]. Our goal is thus to find a CRN that simultaneously meets the constraints given the sketch, satisfies the formal specification and minimises the cost function.

To ensure computational feasibility of the synthesis process and still capture the stochasticity intrinsic in CRN, we employ the Linear Noise Approximation [VK92, EK09] allowing us to encode the synthesis problem as a satisfiability modulo theories problem over a set of parametric Ordinary Differential Equations (ODEs) [EFH08]. We have designed and implemented a novel algorithm for the optimal synthesis of CRNs that employs an almost complete refutation procedure for SMT over ODEs [EFH08, GAC12, GKC13] and a meta-sketching abstraction for controlling the search strategy [BTGC16].

Synthesis of CRNs from input-output functional specifications was also considered in [DMPY15], via a SMT-based generation of qualitative CRN models and consequent optimisation of the rate parameters. In [MPP+18], the authors have improved this approach and synthesised CRNs (including some novel topologies) solving important problems such as majority, maximum and division. Our approach principally offers a better scalability due to the Linear Noise Approximation and thus it is not limited to small numbers of molecules.

Recently, several different approaches have been proposed to automate the design of population protocols, chemical reaction networks and molecular devices. In [VSK18], the authors have developed a new language for programming deterministic chemical kinetics to perform computation. It includes a compiler translating the program into chemical reactions. A design method based on artificial evolution has been used to build a CRN that approximates a real function given on finite sets of input values [DHF19]. The proposed search algorithm evolves the structure of the CRN and optimises the kinetic parameters at each generation. Comparing to our approach, the algorithm cannot guarantee the completeness of the search, however, it was able to find interesting solutions for switches and oscillators. A tool for a parametric analysis of population protocols has been introduce

in [BEJ18]. For a certain class of population protocols, the tool is able to perform analysis over all of the infinitely many initial configurations and thus to provide important insights into the protocol behaviour.

3.2.1 Sketching for CRNs

CRN sketches are defined in a similar fashion to concrete CRNs, with the main difference being that species, stoichiometric constants and reaction rates are specified as unknown *variables*. The use of variables considerably increases the expressiveness of the language, allowing the modeller to specify *additional constraints* over them. Constraints facilitate the representation of key background knowledge of the underlying network, e.g. that a reaction is faster than another, or that it consumes more molecules than it produces.

Another important feature is that reactants and products of a reaction are lifted to *choices* of species (and corresponding stoichiometry). In this way, the modeller can explicitly incorporate in the reaction a set of admissible alternatives, letting the synthesiser resolve the choice.

Further, a sketch distinguishes between *optional* and *mandatory* reactions and species. These are used to express that some elements of the network *might* be present and that, on the other hand, other elements *must* be present. Our sketching language is well suited for synthesis of biological networks: it allows expressing key domain knowledge about the network, and, at the same time, it allows for network under-specification (holes, choices and variables). This is crucial for biological systems, where, due to inherent stochasticity or noisy measurements, the knowledge of the molecular interactions is often partial.

The following example illustrates the proposed sketching language and the optimal solution obtained using our synthesis algorithm.

Example 1 (Bell shape generator) *For a given species K, our goal is to synthesize a CRN such that the evolution of K, namely the expected number of molecules of K, has a bell-shaped profile during a given time interval, i.e. during an initial interval the population K increases, then reaches the maximum, and finally decreases, eventually dropping to 0. Table 3.1 (left) defines a sketch for the bell-shape generator inspired by the solution presented in [Car09].*

This sketch reflects our prior knowledge about the control mechanism of the production/degradation of K. It captures that the solution has to have a reaction generating K (τ_1) and a reaction where K is consumed (τ_2). We also know that τ_1 requires a species, represented by variable λ_1, that is consumed by τ_1, and thus τ_1 will be blocked after the initial population of the species is consumed. An additional species, λ_2, different from λ_1, may be required. However, the sketch does not specify its role exactly: reaction τ_2 consumes either none or one molecule of λ_2 and produces an unknown number of λ_2 molecules, as indicated by the hole $?$. There is also an optional reaction, τ_3, that does not have any reactants and produces either 1 molecule of λ_2 or between 1 and 2 molecules of K. The sketch further defines the mandatory and optional sets of species, the domains of the variables, and the initial populations of species.

Table 3.1 (right) shows the optimal CRN computed by our algorithm with respect to a given cost function and the bell-shape profile produced by the CRN.

$\Lambda_m = \{K\}, \Lambda_o = \{A, B\}, \mathcal{R}_m = \{\tau_1, \tau_2\},$
$\mathcal{R}_o = \{\tau_3\}$, Dec $= \{c_1, \ldots, c_4 : [0, 2],$
$k_1, k_2, k_3 : [0, 0.1], \lambda_1, \lambda_2 : \{A, B\}\},$
Con $= \{\lambda_1 \neq \lambda_2, c_1 < c_2, c_3 > c_4\},$
Ini $= \{K_0 = 1 \wedge A_0 \in [0, 100] \wedge B_0 \in [0, 100]\}$
$\tau_1 = \lambda_1 + c_1 K \xrightarrow{k_1} c_2 K$
$\tau_2 = \{0, 1\}\lambda_2 + c_3 K \xrightarrow{k_2} ?\lambda_2 + c_4 K$
$\tau_3 = \emptyset \xrightarrow{k_3} \{\lambda_2, [1, 2]K\}$

$A + K \to^{56} 2K; \quad K + B \to^{43} 2B$

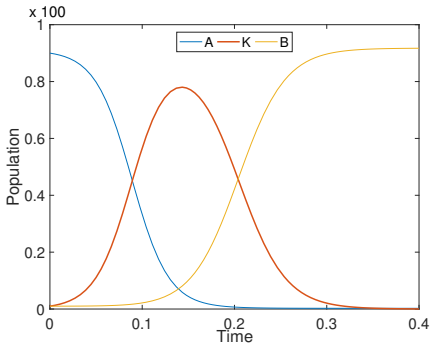


Table 3.1: **Left:** The sketch for bell-shape generator. **Right:** CRN producing the bell-shape profile (species K) synthesized by our algorithm

To specify the required behaviour (e.g., the bell-shape profile from the previous example), we use formulas describing a dynamical profile composed as a finite sequence of phases. Each phase is characterised by an arithmetic predicate, describing the system state at its start and end points (including arithmetic relations between these two), as well as by flow invariants pertaining to the trajectory observed during the phase. This allows us to reason over complex temporal specifications including, for instance, a relevant fragment of bounded metric temporal logic [OW08].

3.2.2 Optimal synthesis algorithm

Further, we denote with $L(\mathcal{S})$ the set of valid CRNs given by a sketch $\mathcal{S}$.

Problem formulation. Given a sketch S, syntactically defined cost function $G_{\mathcal{S}}$ (i.e. defined over the structure of the concrete instantiation rather than its dynamics), property φ, volume N, and precision δ, the optimal synthesis problem is to find CRS $\mathcal{C}^* \in L(\mathcal{S})$, if it exists, such that $[\![\mathcal{C}^*]\!]_N \models^\delta \varphi$ and, for each CRS $\mathcal{C} \in L(\mathcal{S})$ such that $G_{\mathcal{S}}(\mathcal{C}) < G_{\mathcal{S}}(\mathcal{C}^*)$, it holds that $[\![\mathcal{C}]\!]_N \not\models^\delta \varphi$.

Without going into technical details, $\models^\delta$ denotes a "δ-satisfiability" in the framework of *satisfiability modulo ODEs* [EFH08, ERNF15, GKC13], which provides solving procedures for this theory that are sound and complete up to a user-specified precision δ (also know as "δ-decidability" used by Gao et al. in [GAC12]). In this sense, the solvers provide reliable verdicts on either unsatisfiability of the original problem or satisfiability of a δ-relaxation [GKC13, TVKO17].

An important characteristic of the sketching language and the cost function is that for each sketch $\mathcal{S}$ the set $\{G_{\mathcal{S}}(\mathcal{C}) \mid \mathcal{C} \in L(\mathcal{S})\}$ is finite. This follows from the fact that $\mathcal{S}$ restricts the maximal number of species and reactions as well as the maximal number of reactants and products for each reaction. Therefore, we can define for each sketch $\mathcal{S}$ the minimal cost $\mu_{\mathcal{S}}$ and the maximal cost $\nu_{\mathcal{S}}$.
We further defined a meta-sketch abstraction for our sketching language that allows us to formulate an efficient optimal synthesis algorithm.

Definition 8 (Meta-sketch for CRNs) *Given a sketch $\mathcal{S}$ and a cost function $G_{\mathcal{S}}$, we define the meta-sketch $\mathcal{M}_{\mathcal{S}} = \{\mathcal{S}(i) \mid \mu_{\mathcal{S}} \leq i \leq \nu_{\mathcal{S}}\}$, where $\mathcal{S}(i)$ is a sketch whose instantiations have cost smaller than i, i.e. $L(\mathcal{S}(i)) = \{C \in L(\mathcal{S}) \mid G_{\mathcal{S}}(C) < i\}$.*

A meta-sketch $\mathcal{M}_{\mathcal{S}}$ establishes a hierarchy over the sketch $\mathcal{S}$ in the form of an ordered set of sketches $\mathcal{S}(i)$. The ordering reflects the size of the search space for each $\mathcal{S}(i)$ as well as the cost of implementing the CRNs described by $\mathcal{S}(i)$. In contrast to the abstraction defined in [BTGC16], the ordering is given by the cost function and thus it can be directly used to guide the search towards the optimum.

In [CČF+17], we shown the dynamics of $L(\mathcal{S})$ can be described symbolically by a set of parametric ODEs, plus additional constraints. These equations depend on the sketch variables and on the choice functions of each reaction, and describe the time evolution of mean and variance of the species.

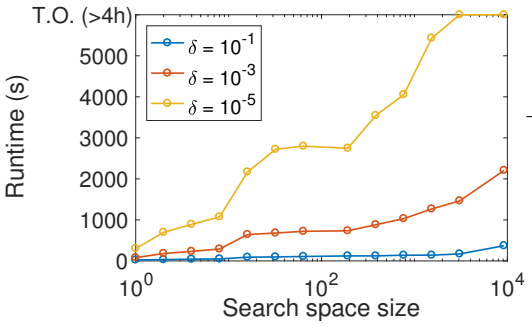
The meta-sketch abstraction, enabling effective pruning of the search space through cost constraints, and the encoding allows us to formulate an algorithm scheme for solving the optimal synthesis problem for CRNs. This scheme repeatedly invokes the SMT solver (δ-Solver) on the sketch encoding, and at each call the cost constraints are updated towards the optimal cost. We consider three approaches: 1) *top-down*: starting from the maximal cost $\nu_{\mathcal{S}}$, it solves meta-sketches with decreasing cost until no solution exists (UNSAT); 2) *bottom-up:* from the minimal cost $\mu_{\mathcal{S}}$, it increases the cost until a solution is found (SAT); 3) *binary search:* it bounds the upper estimate on the optimal solution using a SAT witness and the lower estimate with an UNSAT witness.

We further improve the algorithm by exploiting the fact that UNSAT witnesses can also be obtained at a lower precision δ_{init} ($\delta_{init} \gg \delta$), which consistently improves performance. Indeed, UNSAT outcomes are precise and thus valid for any precision. Note that the top-down strategy does not benefit from this speed-up since it only generates SAT witnesses.

3.2.3 Experimental evaluation

We evaluate the usefulness and performance of our optimal synthesis method on three case studies, representative of important problems studied in biology: (1) the *bell-shape generator*, a component occurring in signalling cascades; (2) *Super Poisson*, where we synthesize CRN implementations of stochastic processes with prescribed levels of process noise; and (3) *Phosphorelay network* [CNCS11], where we synthesize CRNs exhibiting switch-like sigmoidal profiles, which is the biochemical mechanism underlying cellular decision-making, driving in turn development and differentiation.We employ the iSAT(ODE) solver [EFH08, ERNF15], but our algorithm supports any δ-solver.

Bell-Shape Generator. We use the example described in Examples 1, resulting in 8 parametric ODEs, to demonstrate the performance of our approach – see [CČF+17] for the complete experimental evaluation. The synthesised CRN is shown in Figure 3.1. We evaluate the scalability of the solver with respect to precision δ and the size of the discrete search space, altered by changing the domains of species and coefficient variables of the



Search Strategy	# iSAT calls	Total time (s)
bottom-up	7/1	2671/1732
top-down	1/6	4863/5612
binary-search	2/4	3440/3121

Table 3.2: Performance of bell-shape generator model. **Left**: runtimes for different precisions δ and discrete search space size. **Right**: optimal synthesis for the fixed discrete search space size (1536) using different variants of the synthesis algorithm and $\delta = 10^{-3}$.

sketch. We exclude cost constraints as they reduce the size of the search space. Runtimes, reported in Table 3.2 (left), correspond to a single call to iSAT with different δ values, leading to SAT outcomes in all cases. Our experiments (not reported here) show that the size of the continuous state space, given by the domains of rate variables, does not impose such a performance degradation.

In the second experiment, we analyse how cost constraints and different variants of the synthesis algorithm affect the performance of optimal synthesis. Table 3.2 (right) shows the number of iSAT calls with UNSAT/SAT outcomes (2nd column) and total runtimes without/with the improvement that attempts to obtain UNSAT witnesses at lower precision ($\delta_{init} = 10^{-1}$). Importantly, the average runtime for a single call to iSAT is significantly improved when we use cost constraints, since these reduce the discrete search space (between 216s and 802s with cost constraints, 1267s without). Moreover, results clearly indicate that UNSAT cases are considerably faster to solve, because inconsistent cost constraints typically lead to trivial UNSAT instances. This favours the bottom-up approach over the top-down. In this example, the bottom-up approach also outperforms binary-search, but we expect the opposite situation for synthesis problems with wider spectra of costs. As expected, we observe a speed-up when using a lower precision for UNSAT witnesses, except for the top-down approach.

The experimental results show that the proposed approach is able to synthesise challenging systems with up to 37 ODEs and around 10K admissible network topologies. It thus significantly improves the performance and scalability of the existing synthesis methods and paves the way for design automation for provably-correct molecular devices.

3.3 Semi-Quantitative Abstraction of CRNs

Many important biochemical systems lead to complex dynamics that includes *state space explosion, stochasticity, stiffness, and multimodality* of the population distributions [VK92, Gou05], and that fundamentally limits the class of systems the existing verification and synthesis techniques can effectively handle. In order to cope with the computational complexity of the CNR analysis, several approximation techniques have been studied.

For CRNs including only large populations of species, fluid (mean-field) approximation techniques based on ordinary differential equations (ODEs) can be applied [BH12] and extended to approximate higher-order moments [Eng06], Linear Noise Approximation [VK92, EK09] or aggregation scheme over ODEs [CTTV17]. To handle stochasticity of CRNs various *hybrid* approximation schemes have been proposed [HMMW10, HWKT14, CKL16]. Their common idea is as follows: the dynamics of low population species is described by the discrete stochastic process and the dynamics of large population species is approximated by a continuous process. All hybrid schemes have to deal with interactions between low and large population species leading to a computationally demanding numerical analysis that typically limits their scalability.

Alternative approximation techniques for stochastic CRNs employ truncation of insignificant states [MK06, HMW09, MWDH10] or state aggregation/lumping based on exact/approximate probabilistic bisimulation [LS91, DLT08]. Several approximate aggregation schemes leveraging structural properties of CRNs have been proposes [MMR+12, ZWC09, FL09]. In our work [ABČK15][2], we proposed an adaptive aggregation that gives, in contrast to the previous methods, formal guarantees on the approximation error, but typically provide lower state space reductions.

Transient analysis of stochastic CRNs can be performed using the Stochastic Simulation Algorithm (SSA) [Gil77b]. Various partitioning schemes for species and reactions have been proposed for the purpose of speeding up the SSA in multi-scale systems [RA03, SK05, CGP05, GAK15, HGK15]. Although simulation-based analysis is generally faster than direct solution of the stochastic process underlying the given CRN, in many cases. obtaining good accuracy requires large numbers of simulations and can be very time consuming.

In our recent work [ČK19], we have proposed a different approach, so-called *semi-quantitative* approach, that shifts the focus from quantitatively precise results to a more qualitative analysis, closer to how a human would behold the system. It provides scalable and accurate techniques for the CRN analysis as well as the explanation of the system behaviour in the form of tiny models allowing for a synoptic observation. We explain the key ideas behind our approach on a simple gene expression model:

Example 2 (Gene expression) *The simple expression model includes the following reactions protein production (p), protein degradation (d) and blocking (b) the DNA, over three species: protein (P), active DNA (DNA_{on}), and blocked DNA (DNA_{off}):*

$$p\text{: } D_{on} \xrightarrow{10} D_{on} + P \qquad d\text{: } P \xrightarrow{0.1} \emptyset \qquad b\text{: } D_{on} + P \xrightarrow{0.001} D_{off}$$

Using mass-action kinetics (the higher the population of reactants, the fastrer the reaction), the CRN induces a infinite population Markov chain in Fig. 3.4.

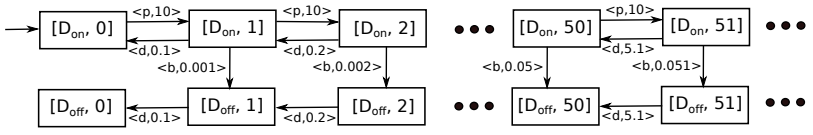


Figure 3.4: The Markov chain for Gene expression, displaying the population of P.

[2]Not part of this habilitation thesis.

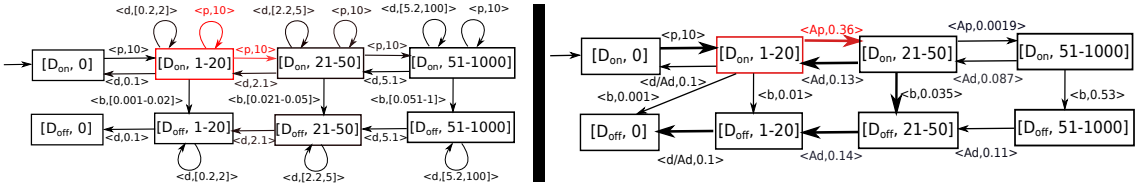


Figure 3.5: The abstract Markov chain for Gene expression with population discretization thresholds $20, 50$ and upper bound 1000. **Left:** Classic may transition function. **Right:** Semi-quantitative version with accelerated transitions.

Semiquantitative abstraction. The abstraction of the state space is simply given by a discretization of the population for each species into finitely many intervals, see Fig. 3.5. The classic may abstraction of the transition function results in non-deterministic self-loops as in Fig. 3.5 (left) in red, which make impossible to conclude anything useful (except for some safety properties) on the behaviour once we reach such a state, even whether it is ever left at all. Instead, we consider sequences of transitions: in this case, sequences of prevalently growing transitions (those increasing the population) are significantly more probable then the prevalently decreasing ones. Consequently, the self-looping transitions are *accelerated* (taken multiple times) to get a "combined" transition that brings a typical representative of this population interval into a higher interval, see Fig. 3.5 (right) also in red. Hence the new rate reflects (i) the mass-action kinetics with the typical population in the interval and (ii) the typical number of the transition repetitions before another interval is reached. These accelerated transitions (further denoted by a prefix A) are the key idea of the semi-quantitative abstraction.

Semi-quantitative analysis. The aim is to prune the abstraction so that only reasonably probable behaviour is reflected, see the thick transitions in the abstraction in Fig. 3.5 (right). To this end, we preserve in each state only the transitions with the highest rate h or almost highest rates, i.e. with $h' > h/envelope$ where $envelope > 1$ is a parameter. Parameter values in $[1, 10]$ ensure we can only look at rates of the same order of magnitude, thus the most probable events and those with e.g. only 20% chance of happening. Higher values then allow for inspection of even less probable behaviours.

Consequently, the method can naturally handle uncertainty in the reaction rates since typically only the relative magnitudes of the rates are important, actually, only their orders of magnitude. This robustness w.r.t. the input is very beneficial for biologists as the precise rates are often not known.

Technically, the analysis relies on repeated alternation of transient and steady-state analysis. First, starting from the initial state, we follow in each state only the transitions with highest rates (most probable ones), until the set of explored state reaches a fixpoint. A part of the created graph is recurrent and forms a bottom strongly connected component (BSCC) or a collection thereof. The system temporarily settles in the steady state of this BSCC. After some time has passed, also a less probable transition happens almost surely and the "BSCC" is exited. These exit points are identified by a steady-state analysis of the BSCC, taking the magnitudes of exiting and non-exiting transition rates into account.

The exit points trigger a new *iteration* of the transient and then the steady-state analysis.

Fig. 3.6 illustrates a situation with two iterations using a more refined abstraction of the simple gene expression model. Decreasing $envelope$ to 1 caused that the blocking reaction is explored in the second iteration – as an exit of the BSCC found in the first iteration. Before that exit happens, the "BSCC" represents a "temporary" steady state of the system.

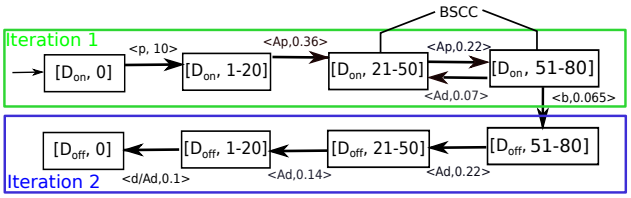


Figure 3.6: The pruned abstraction for the simple expression model using a more refined discretisation $20, 50, 80, 150$ and $envelope = 1$

3.3.1 Case Study: Gene expression

We consider a stochastic gene expression model [GPZC05] described in Table 3.3. As discussed in [HWKT14, GZLS11], the system oscillates between two phases characterised by the D_{on} state and the D_{off} state, respectively. Biologists are interested in how the distribution of the D_{on} and D_{off} states is aligned with the distribution of RNA and protein P.

Table 3.3: Gene expression. The rates are in h^{-1}.

$D_{off} \xrightarrow{0.05} D_{on}$	$D_{on} \xrightarrow{0.05} D_{off}$	$D_{on} \xrightarrow{10} D_{on} + RNA$	$RNA \xrightarrow{1} \emptyset$
$RNA \xrightarrow{4} RNA + P$	$P \xrightarrow{1} \emptyset$	$P + D_{off} \xrightarrow{0.0015} P + D_{on}$	

In order to demonstrate the refinement step and its effect on the accuracy of the model, we start with a very coarse abstraction. It distinguishes only the zero population and the non-zero populations. The pruned abstract model obtained using our approach is depicted in Fig. 3.7 (left).

The proposed analysis of the model identifies the key trends in the system dynamic. The red transitions, representing iterations 1-3 of the semi-quantitative analysis, capture the most probable paths in the system. The green component includes states with DNA on where the system oscillates. The component is reached via the blue state with D_{off} and no RNAs/P. The blue state is promptly reached from the initial state and then the system waits for the next DNA activation. The component is left via a deactivation in the iteration 4 (the blue dotted transition). The deactivation is then followed by fast red transitions leading to the blue state, where the system waits for the next activation. We obtain an oscillation between the blue state and the green component, representing the expected oscillation between the D_{on} and D_{off} states.

As expected, this abstraction does not clearly predict the bimodal distribution on the RNA/P populations – the green component includes states with both the zero and the non-zero population of the mRNA and the protein. In order to obtain a more accurate analysis of the system, we refine the population discretisation using a single level threshold for P and DNA, that is equal to 100 and 10, respectively (the rates in the CRN indicate that the population of P reaches higher values).

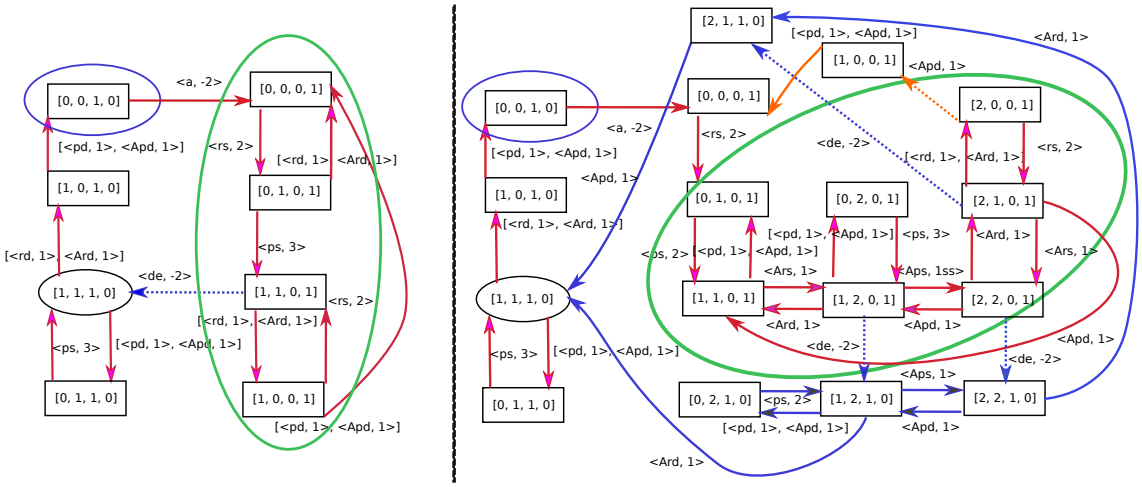


Figure 3.7: Pruned abstraction for the gene expression model using the coarse discretisation (left) and after the refinement (right). The state vector is [P, RNA, D_{off}, D_{on}]. Only order of magnitudes of the transition rates are shown.

Fig 3.7 (right) depicts the pruned abstract model with the new discretisation. We again obtain the oscillation between the green component representing DNA_{on} states and the blue DNA_{off} state. The states in the green component more accurately predicts that in the DNA_{on} states the populations of RNA and P are high and drop to zero only for short time periods. The figure also shows orange transitions within the iteration 2 that extend the green component by two states. Note that the system promptly returns from these states back to the green component. After the deactivation in the iteration 4, the system takes (within the same iteration) the fast transitions (solid blue) leading to the blue component where system waits for another activation and where the mRNA/protein populations decrease. The expected time spent in states on blue solid transitions is small and thus we can reliably predict the bimodal distribution of the mRNA/P populations and its correlation with the DNA state. These predications are in accordance with the results obtained in [HWKT14].

To conclude this case study, we observe a very aligned agreement between the results obtained using our approach with virtually no computational cost and the results in [HWKT14] obtained via advanced and time consuming numerical methods.

3.4 Future Research Directions

Our results have significantly extended capabilities of computational methods for the analysis and design of CRNs, but also opened new research avenues including: i) synthesis of robust biochemical systems that would leverage both the topology synthesis and parameter analysis for CRNs and ii) scalable synthesis of complex biochemical systems using the semiquantitative abstraction.

Further, we will focus on the tool development allowing users to apply the proposed methods on a broader class of biochemical systems. In particular, we will develop a tool

support for the semiquantitative abstraction and analysis allowing a suitable visualisation and result interpretation as well as an automated refinement and robustness analysis.

3.5 Contributed Papers

[BČDŠ13] **Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *CAV'13*, volume 8044 of *LNCS*, pages 107–123. Springer, 2013.**

[ČDKP14] Milan Češka, Frits Dannenberg, Marta Kwiatkowska, and Nicola Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *CMSB'14*, volume 8859 of *LNCS*, pages 86–98. Springer, 2014.

[ČŠDB14] Milan Češka, David Šafránek, Sven Dražan, and Luboš Brim. Robustness analysis of stochastic biochemical systems. *PloS one*, 9(4), 2014.

[ABČK15] Alessandro Abate, Luboš Brim, Milan Češka, and Marta Kwiatkowska. Adaptive aggregation of markov chains: Quantitative analysis of chemical reaction networks. In *CAV'15*, volume 9206 of *LNCS*, pages 195–213. Springer, 2015.

[ČPP+16] Milan Češka, Petr Pilař, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. PRISM-PSY: Precise GPU-accelerated parameter synthesis for stochastic systems. In *TACAS'16*, volume 9636 of *LNCS*, pages 367–384. Springer, 2016.

[ČDP+17] **Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017.**

[CČF+17] **Luca Cardelli, Milan Češka, Martin Fränzle, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. Syntax-guided optimal synthesis for chemical reaction networks. In *CAV'17*, volume 10427 of *LNCS*, pages 375–395. Springer, 2017.**

[ČK19] **Milan Češka and Jan Křetínský. Semi-quantitative abstraction and analysis of chemical reaction networks. In *CAV'19*, volume 11561 of *LNCS*, pages 475–496. Springer, 2019.**

The highlighted papers are attached to this thesis.

Chapter 4

Design of Approximate Circuits and Automata

Approximate systems that relax requirements on functional correctness play an important role in the development of resource-efficient HW and SW systems. Designing approximate systems is a very complex and time-demanding process trying to find optimal trade-offs between the approximation error and resource savings. In our recent work, we have focused on automated approximation techniques for digital circuits and automata that form essential building blocks for many systems.

In this thesis, we briefly present the following two results. 1) scalable approximation of arithmetic circuits [ČMM+17, ČMM+18, ČMM+20] and 2) automata reduction for HW-accelerated deep packet inspection and regular expression matching [ČHH+18, ČHH+19a, ČHH+19b].

4.1 Approximation of Arithmetic Circuits

Approximate circuits are circuits that trade the precision of computation for the circuit chip area or power consumption. Approximate circuits are important in many prominent applications such as image and video processing [GMRR13] or architectures for neural networks [MAFL10, MSS+16]. Automated methods allowing one to develop such circuits are thus in high demand.

There exists a vast body of literature (see e.g. [RS19, VS15b, NHT+16, MHVS17, LRY+16]) demonstrating that evolutionary-based algorithms are able to automatically design innovative implementations of approximate circuits providing high-quality trade-offs among the different design objectives. As shown in [YC16, CSGD16a], many applications favour *provable error bounds* on resulting approximate circuits, which makes automated design of such circuits a very challenging task. Note that circuit simulation does not scale beyond circuits with more than 12-bit operands even when exploiting modern computing architectures [MSS+16].

There exist several approaches employing different *formal verification methods* to check the correctness of circuits [VS11, CYB+15, SAGK+16] or to evaluate their approx-

imation error. The most promising methods for the error evaluation include binary decision diagrams (BDDs) [VMS17], boolean satisfiability (SAT) solving [VARR11], model checking [CS+16], or symbolic computer algebra employing Gröbner bases [FGD18]. The resulting approximation techniques, however, do not scale beyond approximation of multipliers with 12-bit operands and adders with 16-bit operands.

In our recent work [ČMM+17, ČMM+20], we proposed a new approximation technique that integrates SAT solving into evolutionary approximation, in particular into *Cartesian genetic programming* (CGP). We focus on the *worst case absolute error* (WCAE) metric, which is one of the most commonly used error metrics. The key distinguishing idea of our approach is simple, but it makes our approach dramatically more scalable comparing to previous approaches. Namely, we *restrict the resources* (running time) available to the SAT solver when evaluating a candidate solution. If no decision is made within the limit, a minimal score is assigned to the candidate circuit. This approach leads to a *verifiability-driven search strategy* that drives the search towards *promptly verifiable approximate circuits.* The technique is implemented within ADAC—Automatic Design of Approximate Circuits [ČMM+18] – our framework for automated design of approximate arithmetic circuits that integrates efficient circuit simulation and formal methods for approximate equivalence checking into a search-based circuit optimisation.

As we shown in [ČMM+20], our verifiability-driven approximation strategy is now able to discover complex arithmetic circuits such as 32-bit approximate multipliers, 32-bit approximate multiply-and-accumulate (MAC) circuits, and 24-bit dividers providing high quality trade-offs between the approximation error and energy savings. These results clearly demonstrate the superior performance and scalability of our approach comparing to other existing approximation techniques.

4.1.1 Verifiability-Driven approximation

The problem of finding the best trade-offs between the circuit size and the WCAE, can be naturally seen as a multi-objective optimisation problem. In our approach, we, however, treat it as a series of single-objective problems where we fix the required values of the WCAE. This approach is motivated by the fact that the WCAE is usually given by the concrete application where the approximate circuits are deployed. Moreover, as shown in several studies [VS15a], optimising the chip size for a fixed error allows one to achieve significantly better performance compared to more general multi-objective optimisation producing Pareto fronts. The optimisation problem is formalised as follows:

Problem formulation. *For a given golden circuit G and a threshold $\mathcal{T}$, our goal is to find a circuit C^* with the minimal size such that the error WCAE$(G, C^*) \leq \mathcal{T}$, where*

$$\mathrm{WCAE}(G, C) = \max_{x \in \{0,1\}^n} \frac{|\mathrm{int}(f_G(x)) - \mathrm{int}(f_C(x))|}{2^m - 1}.$$

Before presenting our approach, we emphasise that our aim is not to provide a complete algorithm that guarantees the optimality of C^*: such an algorithm clearly exists as the number of circuits with a given size is finite, and one can, in theory, enumerate

them one by one. We rather design an effective search strategy that is able to provide high-quality approximations for complex arithmetic circuits having thousands of gates.

Our novel optimisation scheme employing four key components: (1) a *generator* of candidate circuits that builds on CGP, (2) an *evaluator* that evaluates the error of the candidates by leveraging SAT-based verification methods, (3) a *verifiability-driven search* integrating the cost of the circuit evaluation into the fitness function, and (4) an *adaptive strategy* adjusting the allowed cost of evaluation of candidate solutions during the approximation process.

Generating candidate circuits using CGP. CGP is a form of genetic programming where candidate solutions are represented as a string of integers of a fixed length that is mapped to a directed acyclic graph [MT00]. This integer representation is called a *chromosome*. The chromosome can efficiently represent common computational structures including mathematical equations, computer programs, neural networks, and digital circuits. We use a standard CGP that employs the (1+λ) search method where a single generation of candidates consists of the parent and λ offspring candidates. The fitness of each of the solutions is evaluated and the best solution is preserved as the parent for the next generation. Other candidates from the generation are discarded.

In circuit approximation, the evolution loop typically starts with a *parent* representing a correctly working circuit. New candidate circuits are obtained from the parent using a *mutation operator* which performs random changes in the candidate's chromosome in order to obtain a new, possibly better candidate solution. The mutations can either modify the node interconnection or functionality. The number of the nodes of candidate circuits is reduced by making some nodes inactive, i.e. disconnected from the outputs of the circuit. However, since such nodes are not removed, they can still be mutated and eventually become active again. The whole evolution loop is repeated until a termination criterion (in our case, a time limit fixed for the evolution process) is met.

Candidate circuit evaluation. The evaluation takes into consideration two attributes of the circuit, namely, whether the approximation error represented by WCAE is smaller than the given threshold and the size of the circuit. The procedure deciding whether $\mathrm{WCAE}(G, C) \leq \mathcal{T}$ represents the most time consuming part of the design loop.

To decide whether $\mathrm{WCAE}(G, C) \leq \mathcal{T}$, we adopt the concept of an *approximation miter* introduced in [VARR11, CSGD16b]. The miter is an auxiliary circuit that consists of the inspected approximate circuit C and the golden circuit G which serves as the specification. C and G are connected to identical inputs. A subtractor and a comparator then check whether the error introduced by the approximation is greater than a given threshold $\mathcal{T}$. The output of the miter is a single bit which evaluates to logical 1 if and only if the constraint on the WCAE is violated for the given input x.

Once the miter is built, it is translated to a Boolean formula that is satisfiable if and only if $\mathrm{WCAE}(G, C) > \mathcal{T}$. This approach allows one to reduce the decision problem to a SAT problem and use existing powerful SAT solvers. We optimize the miter construction by using a novel circuit implementation of the subtractor, absolute value, and

comparator nodes as described in [ČMM+17]. In particular, it avoids long XOR chains, which are a known cause of poor performance of the state-of-the-art SAT solvers [HJ12].

Verifiability-driven search strategy. The strategy uses an additional criterion for the evaluation of the circuit C. The criterion reflects the ability of the decision procedure, in our case a SAT solver, to prove that $\mathrm{WCAE}(G, C) \leq \mathcal{T}$ with a given limit L on the resources available. It leverages the observation that a long sequence of candidate circuits B_i improving the size and having an acceptable error has to be typically explored to obtain a solution that is sufficiently close to an optimal approximation C^*. Therefore, both the SAT and the UNSAT queries to the SAT solver have to be short. If the procedure fails to prove $\mathrm{WCAE}(G, C) \leq \mathcal{T}$ within the limit L, we generate a new candidate.

The interpretation of the resource limit L on checking that $\mathrm{WCAE}(G, C) \leq \mathcal{T}$ depends on the implementation of the underlying satisfiability checking procedure. Note that a time limit is not suitable since it does not reflect how the structural complexity of candidate circuits affects the performance of the procedure. Therefore, we employ the limit on the *maximal number of backtracks* in which a single variable can be involved during the backtracking process (also called the maximal number of *conflicts* on a variable). As the backtracking represents the key and computationally demanding part of modern SAT solvers [LMS05], it allows one to effectively control the time needed for particular evaluation queries. Moreover, it takes into account the structural complexity of the underlying boolean formula capturing the complexity of the circuit.

Adaptive resource limit strategy. We further proposed a novel *adaptive strategy* that alters the resource limit within the evolutionary run and tries to set it to the most suitable value with regards to the recently achieved progress [ČMM+20]. We designed the strategy scheme based on our previous observations that the limit should be kept low in the early stages of the evolution so that the clearly redundant logic can be quickly eliminated. Later in the evolutionary process, the algorithm converges to a locally optimal solution and improvements in the fitness cease to occur. When such a stage is reached, the limit needs to be increased in order to widen the space of feasible candidate solutions at the expense of slower candidate evaluation. Moreover, once some more significantly changed solution is found, it may again be possible to shorten the time limit needed for the evaluation, and the process of extending and shrinking the time limit may repeat.

4.1.2 Experimental evaluation

In this section, we demonstrate that our approach generates approximate circuits that significantly outperform circuits obtained using state-of-the-art approximation techniques. In particular, we show that our circuits provide significantly better trade-offs between the precision and energy consumption. We focus on multipliers since their approximation represents a challenging and widely studied problem—see, e.g., the comparative study of [JHL15]. On the other hand, the existing literature does not offer a sufficient number of high-quality approximate MACs or dividers to carry out a fair comparison: our work is the first one that automatically handles such circuits.

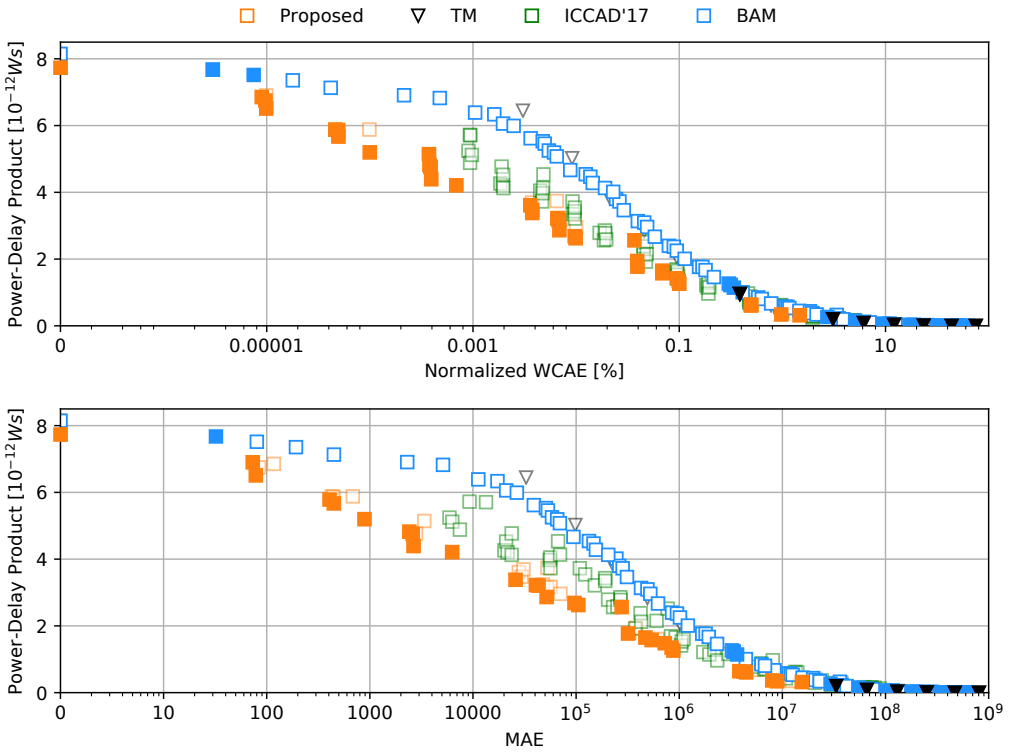


Figure 4.1: A comparison of 16-bit approximate multipliers obtained using the proposed approach and the state-of-the-art approximation techniques. The filled marks represent solutions providing the best PDP for the given precision.

In the comparison, we consider two approximate architectures for multipliers that are known to provide the best results, namely *truncated multipliers* (TMs) that ignore the values of least significant bits and *broken-array multipliers* (BAMs) [FAF13]. TMs and BAMs can be parameterised to produce approximate circuits for the given bit-width and the required error. In contrast to our search-based approach, these circuits are constructed using a deterministic procedure simplifying accurate multipliers. Note that the method is applicable for design of approximate multipliers only. To demonstrate the practical impact of the proposed adaptive strategy, we also consider circuits presented in our previous work [ČMM+17] obtained using verifiability-driven approximation with a fixed limit strategy.

Fig. 4.1 shows the parameters of resulting circuits belonging to Pareto front. For each circuit, the figure illustrates the trade-off between the precision and the power-delay-product (PDP) that adequately captures both the circuit's energy consumption and its delay. The top plot of the figure illustrates the WCAE–PDP trade-offs. We also evaluated the mean absolute error (MAE) of the solutions since MAE represents another important circuit error metric. The results are presented in the bottom plot of the figure.

The figure clearly demonstrates that our general approximation approach is able to significantly outperform both TMs and BAMs representing the dedicated state-of-the-art approximation methods for multipliers . The figure also shows that the proposed adaptive strategy improves our previously obtained results even further.

4.2 Automata Reduction for Regex Matching in HW

Deep packet inspection via regular expression (RE) matching is a crucial task of network intrusion detection systems (IDSes), such as SNORT [Sno], SURICATA [Mat], or BRO [Ver18], which secure Internet connection against attacks and suspicious network traffic. Monitoring high-speed computer networks (100 Gbps and faster) in a single-box solution demands that the RE matching, traditionally based on *finite automata* (FAs), is accelerated in hardware.

A well-suited technology for accelerating IDSes is that of *field-programmable gate arrays* (FPGAs). They provide high computing power and flexibility for network traffic processing, and they are increasingly being used in data centers [CCP+16] for this purpose. The flexibility of FPGAs allows them to match REs at speeds over 100 Gbps [MKP16b]. Such high speeds, however, put excessive demands on the resources of FPGAs. The sets of the matched REs are complex, large, and still growing, and matching on the speeds of tens and hundreds of Gbps requires massive parallelization. For instance, in the HW architecture that we propose in [ČHH+19b], processing 100 Gbps input network traffic requires 64 concurrently functioning RE matching units 200 MHz) and processing 400 Gbps requires even 256 units. Despite the fact that FPGAs provide an efficient way of implementing *nondeterministic finite automata* (NFAs), see e.g. [CS03], the required number of RE matching units (NFAs) easily exceed the size of any available FPGA chip, namely the number of available *look-up tables* (LUTs). Reducing the consumed resources, in particular the size of the NFAs, is thus of paramount importance.

Approximate reduction via probabilistic distance. Various language-preserving automata reduction approaches exist, mainly based on computing (bi)simulation relations on automata states [BG00, CM13]. The reductions they offer, however, do not satisfy the needs of high-speed hardware-accelerated NIDSes. To attack this problem, we have recently proposed *approximate reduction* of NFAs, allowing for a trade-off between the achieved reduction and the precision of the regex matching [ČHH+18, ČHH+19a].

To formalise the intuitive notion of precision, we proposed a novel *probabilistic distance* of automata. It captures the probability that a packet of the input network traffic is incorrectly accepted or rejected by the approximated NFA. The distance assumes a *probabilistic model* of the network traffic. Intuitively, the model concisely captures the significant network traffic and drives the reduction towards automata that incorrectly accept or reject only insignificant packets. We considered two variants of an optimization problem: 1) minimizing the NFA size given the maximum allowed error (distance from the original), or 2) minimizing the error given the maximum allowed NFA size. Finding such optimal approximations is, however, computationally hard (**PSPACE** -complete, the same as precise NFA minimization).

Sub-optimal solutions. To overcome the complexity, we sacrifice the optimality and, motivated by the typical structure of NFAs that emerge from a set of regexes used by NIDSes (a union of many long "tentacles" with occasional small strongly-connected components), we limit the space of possible reductions by restricting the set of operations

they can apply to the original automaton. Namely, we consider two reduction operations: 1) collapsing the future of a state into a *self-loop* (this reduction over-approximates the language) and 2) *removing states* (such a reduction is under-approximating).

The problem of identifying the optimal sets of states on which these operations should be applied is still **PSPACE** -complete. The restricted problem is, however, more amenable to an approximation by a *greedy algorithm*. The algorithm applies the reductions state-by-state in an order determined by a precomputed *error labelling* of the states. The process is stoppped once the given optimization goal in terms of the size or error is reached. The labelling is based on the probability of packets that may be accepted through a given state and hence over-approximates the error that may be caused by applying the reduction at a given state. As our experiments show, this approach can give us high-quality reductions while ensuring formal error bounds.

Finally, it turns out that even the pre-computation of the error labelling of the states is costly (again **PSPACE** -complete). Therefore, we propose several ways to cheaply over-approximate it such that the strong error bound guarantees are still preserved. Particularly, we are able to exploit the typical structure of the "union of tentacles" of the NFAs in an algorithm that is exponential in the size of the largest "tentacle" only, which is indeed much faster in practice.

Experimental evaluation. The complete experimental evaluation of the proposed reduction techniques can be found in [ČHH+19a]. We present here only the key experimental observations and conclusions. Note that after the approximate reduction, we use the tool REDUCE [M+], implementing the state-of-the-art language-preserving reduction, to further simplify the obtained NFAs

First, we observed that the error bounds obtained by the approximate error labelling (driving the reduction) provide a very good approximation of the real probabilistic distance. On the other hand, the difference between the probabilistic distance (using the traffic model) and the real traffic error (corresponding to an HTTP traffic sample) can vary significantly for different REs (different NFAs). Since all experiments use the same probabilistic automaton and the same traffic, this discrepancy is accounted to the different set of packets that are incorrectly accepted by the reduced automata. If the probability of these packets is adequately captured in the traffic model, the difference between the distance and the traffic error is small and vice versa. We emphasise that there are no guarantees on the relationship between the probabilistic distance and the traffic error. These observations demonstrate that a suitable traffic model is essential and that our approach for building the model has certain limitations (see [ČHH+19a] for more details).

Second, the results clearly demonstrate that we can often achieve a very significant reduction for a negligible loss of precision. For example, for the `backdoor` RE, representing a very challenging reduction problem (the original NFA has more than 1300 states), our approach still provides significant reductions while keeping the traffic error small: about a 5-fold reduction is obtained for the traffic error 0.03 % and a 10-fold reduction is obtained for the traffic error 6.3 %. The practical impact of such reductions are discussed in the next section.

Finally, we observed that the most time-consuming step of the reduction process is the

computation of state labellings (it takes at least 90 % of the total time). The crucial observation is that the structure of the NFAs fundamentally affects the performance of this step. The key reason behind this slowdown is the determinisation (or alternatively disambiguation) process required by the product construction underlying the state labelling computation. The size of the product directly determines the time and space complexity of solving the linear equation system required for computing the state labelling. Therefore, it is essential to employ the cheap over-approximation of the state-labelling that allows us to obtain the reduced automata for complex REs in a reasonable time: for example, for the `http-backdoor` RE, the labelling took around 20 minutes and the consequent optimisation and language preserving reduction around 6 minutes in the worst case.

Efficient lightweight labelling. In order to effectively handle significantly more complex REs corresponding to NFAs with more then 10K states, we have also proposed a lightweight state labelling [ČHH+19b]. In contrast to the previous state-labelling techniques using the probabilistic automaton, the lightweight approach directly uses a sample of the network traffic to estimate the error for the states and to determine the states to be removed. This method has two advantages: 1) it avoids the semi-automated construction of the model that requires a network expert, and 2) it can scale to NFAs that are an order of magnitude larger. Indeed, the disadvantage is that it does not provide any formal bounds on the approximation error.

The lightweight approach also takes an advantage of particularities of standard network traffic. Namely, given an NFA constructed from the REs of interest, we label its states with their *significance*—the likelihood that they will be used during processing a packet—, and then simplify the least significant parts of the automaton. The automata reduction is implemented by two operations: 1) *pruning* that removes from the automaton a set R of states considered as the most insignificant and 2) *merging* of states forming a chain and having a similar significance. The significance of a state is determined using a finite sample of the network traffic from the network node where the IDS is to be deployed. The time complexity of the most expensive step, computing the state labelling, is $\mathcal{O}(n^2k)$ where n is the number of states of the NFA and k is the size of the training traffic. Using 1M packets, the labelling took around 15 min for the NFA with 12K states.

4.2.1 Multi-stage regex matching architecture

In [ČHH+19b], we have proposed a concept of a *multi-stage* RE matching unit that uses aggressive approximate reductions, which do not preserve the language of the NFA, to utilise FPGA resources efficiently. The architecture of the RE matching unit is composed of several stages (see Fig. 4.2 for an example of a 3-stage architecture). Every stage in the architecture contains an instantiation of the RE matching engine described, i.e. a set approximate NFAs working in parallel.

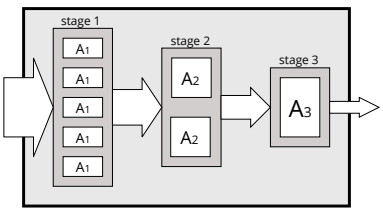


Figure 4.2: An example of the multi-stage architecture with 3 stages.

The idea is that each of the stages will use different NFAs—starting with a bigger number of smaller and imprecise NFAs and proceeding to smaller numbers of larger but more precise NFAs—to decrease, in a resource-efficient way, the number of packets entering the subsequent stage. Consider an NFA A that recognises the language L defined by the REs in a given SNORT module. The first stage of the architecture contains many copies of a small NFA A_1, which over-approximates L, i.e., apart from all packets in L, it also matches some packets not in L. All matched packets are then sent to Stage 2, which contains less copies of a larger NFA A_2, which over-approximates L, but more precisely than A_1 (which is the reason it is larger—less precise approximations of L obtained by our reduction are usually smaller than more precise ones).

The number of copies of A_2 can be smaller due to the fact that the traffic entering it is just a fraction of the input traffic since Stage 1 has removed a significant number of packets from further processing. Each subsequent stage contains an even smaller number of even more precise (and, therefore, larger) NFAs. The final stage contains either copies of A, in which case the output of the RE matching unit is exactly the packets from L (precise matching), or as precise over-approximation of L as possible given the available resources, in which case the last remaining false positives need to be removed in software.

Consider a set of approximate automata $\mathbb{A} = \{A_1, \ldots, A_k\}$ obtained using the reduction techniques from the input NFA A. Each NFA A_i comes with two parameters: 1) its size given as the number of LUTs obtained from its HW synthesis and (2) the probability that it accepts an input packet – the probability is obtained from the evaluation using the considered network traffic. Note that lower is better for both parameters. Given the set $\mathbb{A}$, we aim at obtaining a configuration of the multi-stage architecture that is as small and precise as possible. This gives rise to the following two optimisation problems: 1) minimise the amount of resources used by the RE matching unit given a maximum speed of traffic on its output and 2) minimise the speed of the traffic on the output of the last n-th stage of the RE matching unit given a fixed amount of resources. These optimisation problems lead to a mixed integer quadratically constrained program and can be easily solved by existing solvers such as Gurobi [Gur18].

Experimental evaluation. Having relevant network traffic data is essential to evaluate the performance and practical usefulness of the lightweight approximation and the proposed RE matching architecture. We used data obtained from two measuring points of a nation-wide Internet provider connected to a 100 Gbps backbone link. The testing data was sampled over the time of 105 hours and contains around 210M packets. The training data used for labelling the automata contained around 1M packets sampled at a different time. The complete experimental evaluation can be found in [ČHH+19b]. Here we present only selected results.

First, we focus on the precision of the lightweight approximation, in particular, we consider two important metrics: 1) *acceptance precision* AP expresses the ratio of correctly accepted packets to all accepted packets and hence characterises the error caused by the over-approximation and 2) *acceptance probability* $Prob$ captures what fraction of the input traffic is accepted by the reduced NFA and passed to the next stage – this is important for building efficient multi-stage architectures.

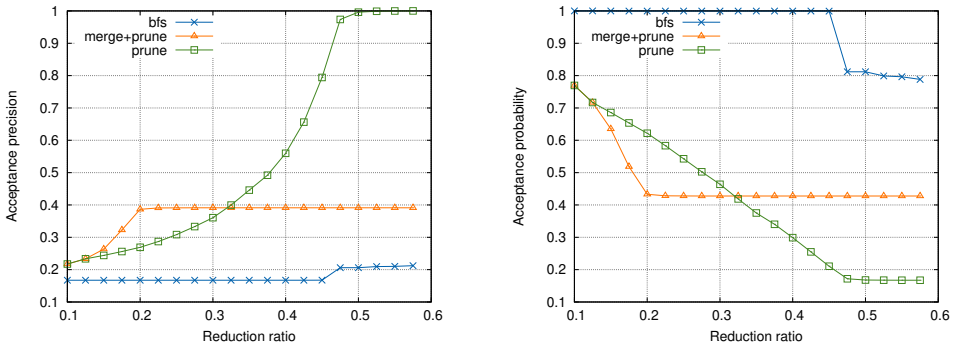


Figure 4.3: AP (left) and $Prob$ (right) for `l7-all`.

Fig. 4.3 shows results for the NFA the describing `l7-all` RE, the most challenging RE where the corresponding NFA has over 7K and 2.6M transitions. We observe that the particular reduction techniques provide a very different quality of the trade-offs. In particular, *bfs* (a simple breadth-first-search-based reduction used as the baseline), is not capable to produce any useful approximation. Further, we can observe that *merge-prune* dominates for reduction ratios lower than 0.3, but it is significantly outperformed by *prune* for higher reduction ratios. The figures show that these trends are preserved for both of our metrics. Note that, in this case, the original NFA accepts around 17 % of the traffic, and using the *prune* technique, we obtain a reduced NFA having only a half of the states with almost the same acceptance probability $Prob$.

Second, we explored whether the reduced NFAs can be compiled into a multi-stage architecture with throughput of 100 Gbps and beyond. We synthesise the proposed architectures for a card with the Xilinx Virtex UltraScale+ VU9P FPGA chip, which contains 1,182k LUTs where 737k LUTs can be used for the RE matching. The remaining LUTs are used for other necessary logic as routing the packets.

The table below presents results of precise architectures for the `backdoor` RE. The table shows how the number of required LUTs decreases with the increasing number of stages. In this case, using more than 4 stages does not bring any further reduction and contrarily introduces an overhead (not reported here). The single stage architecture (i.e., the "1 stg" column), can process traffic up to 200 Gbps only (the precise NFA consumes 3,695 LUTs). In order to process 400 Gbps, it is necessary to use the multi-stage architecture, in which case the one with four stages gives the best results.

Precise				
speed	1 stg	2 stg	3 stg	4 stg
100	236k	56k	50k	50k
200	473k	113k	99k	96k
400	946k	223k	194k	186k

The table below presents results of precise architectures for the `l7-all` RE, our most challenging example. The precise NFA consumes 27,650 LUTs, but more importantly, it is less amenable for approximate reduction because, in contrast to REs from SNORT, the RE is matched by many packets. Our best solution reduces the input traffic from 100 Gbps to 17 Gbps and uses 597k LUTs in two stages. Note that, the reduced

Precise				
speed	1 stg	2 stg	3 stg	4 stg
100	1.8M	894k	880k	880k

17 % of traffic				
speed	1 stg	2 stg	3 stg	4 stg
100	1.1M	597k	648k	701k

NFA in the final stage has almost 100 % precision and thus only a small fraction of packets are misclassified.

Our experiments clearly demonstrate the practical potential of our approach. The key observation is that the resource reductions provided by the particular multi-stage architectures directly depend on the characteristics of the underlying NFAs (both the precise NFA and the reduced variants) and the typical traffic. Apart from the size of the precise NFA, there are two crucial characteristics: (1) whether the number of packets accepted by the precise NFA is low and (2) whether the reduction can compress the NFA while not increasing the number of accepted packets too much. If both these conditions are met (as for `backdoor` and other REs from SNORT), we observe drastic resource savings allowing us to achieve throughput of the resulting IDSes going beyond 100 Gbps. Comparing to the existing state-of-the-art solutions using GPUs and FPGAs for the HW-accelerated deep packet inspection [ARS15, MKP16a, YJB+18], this throughput is unprecedented for REs of such size and complexity.

On the other hand, if the original NFA is large, accepts many packets, and highly precise reductions achieve only moderate reductions (as for `l7-all`), the resulting multi-stage architecture provides only moderate savings and ensuring 100 Gbps remains at the edge of what we can achieve.

4.3 Future Research Directions

Our results have significantly improved the scalability of automated techniques for designing approximate circuits with formal guarantees on the worst-case error. It is still an open problem whether a similar scalability can be achieved also for more complicated metrics such mean error or error rate (these metrics require counting), and for non-arithmetic circuits having a more complex specification. Efficient approximation of sequential circuits is also an open challenge with important practical applications. In the area of HW-accelerated RE matching, it is essential to reconsider the existing encoding schemes for NFAs to achieve efficient utilisation of the resources with respect to the structure of over-approximate NFAs.

Further, we will focus on applying approximate circuits and automata in practically relevant domains such as image and video processing or architectures for neural networks.

4.4 Contributed Papers

[ČMM+17] **Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In *ICCAD'17*, pages 416–423. IEEE, 2017.**

[ČMM+18] Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. ADAC: Automated design of approximate circuits. In *CAV'18*, volume 10981 of *LNCS*. Springer, 2018.

[ČHH+18] Milan Češka, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar. Approximate Reduction of Finite Automata for High-Speed Network Intrusion Detection. In *TACAS'18*, volume 10806 of *LNCS*, pages 155–175. Springer, 2018.

[ČHH+19a] **Milan Češka, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar. Approximate reduction of finite automata for high-speed network intrusion detection. *International Journal on Software Tools for Technology Transfer*, 2019.**

[ČHH+19b] Milan Češka, Vojtěch Havlena, Lukáš Holík, Jan Kořenek, Ondřej Lengál, Denis Matoušek, Jiří Matoušek, Jakub Semrič, and Tomáš Vojnar. Deep packet inspection in fpgas via approximate nondeterministic automata. In *FCCM'19*, pages 109–117. IEEE, 2019.

[ČMM+20] Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. Adaptive verifiability-driven strategy for evolutionary approximation of arithmetic circuits. *CoRR*, abs/2003.02491, 2020. *Submitted to Applied Soft Computing*.

The highlighted papers are attached to this thesis.

Bibliography

[AAER07] Dana Angluin, James Aspnes, David Eisenstat, and Eric Ruppert. The computational power of population protocols. *Distributed Computing*, 20(4):279–304, 2007.

[ABC+18] Sebastian Arming, Ezio Bartocci, Krishnendu Chatterjee, Joost-Pieter Katoen, and Ana Sokolova. Parameter-independent strategies for pMDPs via POMDPs. In *QEST'18*, volume 11024 of *LNCS*, pages 53–70. Springer, 2018.

[ABČK15] Alessandro Abate, Luboš Brim, Milan Češka, and Marta Kwiatkowska. Adaptive aggregation of markov chains: Quantitative analysis of chemical reaction networks. In *CAV'15*, volume 9206 of *LNCS*, pages 195–213. Springer, 2015.

[ÁBD+14] Erika Ábrahám, Bernd Becker, Christian Dehnert, Nils Jansen, Joost-Pieter Katoen, and Ralf Wimmer. Counterexample generation for discrete-time markov models: An introductory survey. In *SFM'14*, volume 8483 of *LNCS*, pages 65–121. Springer, 2014.

[ABD+15] Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Dependable Software Systems Engineering*, volume 40, pages 1–25. IOS Press, 2015.

[ADK+18] Alessandro Abate, Cristina David, Pascal Kesseli, Daniel Kroening, and Elizabeth Polgreen. Counterexample guided inductive synthesis modulo theories. In *CAV'18 (1)*, volume 10981 of *LNCS*, pages 270–288. Springer, 2018.

[AMSW11] Aleksandr Andreychenko, Linar Mikeev, David Spieler, and Verena Wolf. Parameter Identification for Markov Models of Biochemical Reactions. In *CAV'11*, volume 6806 of *LNCS*, pages 83–98. Springer, 2011.

[ARS15] Matteo Avalle, Fulvio Risso, and Riccardo Sisto. Scalable algorithms for nfa multi-striding and nfa-based deep packet inspection on gpus. *IEEE/ACM Transactions on Networking*, 24(3):1704–1717, 2015.

[ASB+95] Adnan Aziz, Vigyan Singhal, Felice Balarin, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *CAV'95*, volume 939 of *LNCS*, pages 155–165. Springer, 1995.

[ASFS18] Rajeev Alur, Rishabh Singh, Dana Fisman, and Armando Solar-Lezama. Search-based program synthesis. *Commun. ACM*, 61(12):84–93, 2018.

[ASSB96] Adnan Aziz, Kumud Sanwal, Vigyan Singhal, and Robert Brayton. Verifying Continuous Time Markov Chains. In *CAV'96*, volume 1102 of *LNCS*, pages 269–276. Springer, 1996.

[BBNS15] Ezio Bartocci, Luca Bortolussi, Laura Nenzi, and Guido Sanguinetti. System design of stochastic models using robustness of temporal properties. *Theor. Comput. Sci.*, 587:3–25, 2015.

[BBPM00] L. Benini, A. Bogliolo, G. Paleologo, and G. De Micheli. Policy optimization for dynamic power management. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 8(3):299–316, 2000.

[BČDŠ13] Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *CAV'13*, volume 8044 of *LNCS*, pages 107–123. Springer, 2013.

[BDH+17] Carlos E. Budde, Christian Dehnert, Ernst Moritz Hahn, Arnd Hartmanns, Sebastian Junges, and Andrea Turrini. JANI: quantitative model and tool interaction. In *TACAS'17*, volume 10206 of *LNCS*, pages 151–168, 2017.

[BDHK06] Henrik C. Bohnenkamp, Pedro R. D'Argenio, Holger Hermanns, and Joost-Pieter Katoen. MODEST: A compositional modeling formalism for hard and softly timed systems. *IEEE Trans. Software Eng.*, 32(10):812–830, 2006.

[BEJ18] Michael Blondin, Javier Esparza, and Stefan Jaax. Peregrine: A tool for the analysis of population protocols. In *CAV'18*, volume 10981 of *LNCS*, pages 604–611. Springer, 2018.

[BESW10] Dragan Bošnački, Stefan Edelkamp, Damian Sulewski, and Anton Wijs. Parallel probabilistic model checking on general purpose graphics processors. *International Journal on Software Tools for Technology Transfer*, 13(1):21–35, 2010.

[BG00] Doron Bustan and Orna Grumberg. Simulation Based Minimization. In *CADE'17*, volume 1831 of *LNCS*, pages 255–270. Springer, 2000.

[BG08] Nathan Bell and Michael Garland. Efficient sparse matrix-vector multiplication on CUDA. NVIDIA Technical Report NVR-2008-004, NVIDIA Corporation, 2008.

[BGK+11] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *TACAS'11*, volume 6605 of *LNCS*, pages 326–340. Springer, 2011.

[BH12] Luca Bortolussi and Jane Hillston. Fluid model checking. In *CONCUR'12*, pages 333–347. Springer, 2012.

[BHH+13] Christel Baier, Ernst Moritz Hahn, Boudewijn R Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model checking for performability. *Mathematical Structures in Computer Science*, 23(04), 2013.

[BJP+12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. Synthesis of reactive (1) designs. *Journal of Computer and System Sciences*, 78(3), 2012.

[BL13] Luca Bortolussi and Roberta Lanciani. Model checking markov population models by central limit approximation. In *QEST'13*, volume 8054 of *LNCS*, pages 123–138. Springer, 2013.

[BL16] Ezio Bartocci and Pietro Lió. Computational modeling, formal analysis, and tools for systems biology. *PLoS computational biology*, 12(1), 2016.

[BMS16] Luca Bortolussi, Dimitrios Milios, and Guido Sanguinetti. Smoothed model checking for uncertain continuous-time markov chains. *Information and Computation*, 247:235–253, 2016.

[Bon14] A. B. Bondy. *Foundations of Software and System Performance Engineering*. Addison Wesley, 2014.

[BTGC16] James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. Optimizing synthesis with metasketches. In *POPL'16*, pages 775–788. ACM, 2016.

[Car09] Luca Cardelli. *Artificial Biochemistry*, pages 429–462. Springer, 2009.

[Car13] Luca Cardelli. Two-domain DNA strand displacement. *Mathematical Structures in Computer Science*, 23(02):247–271, 2013.

[CBHB09] Vijaysekhar Chellaboina, S. P. Bhat, W. M. Haddad, and Dennis S. Bernstein. Modeling and analysis of mass-action kinetics. *IEEE Control Systems Magazine*, 29(4):60–78, 2009.

[CCD16] Krishnendu Chatterjee, Martin Chmelik, and Jessica Davies. A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In *AAAI*, pages 3225–3232. AAAI Press, 2016.

[CČF+17] Luca Cardelli, Milan Češka, Martin Fränzle, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. Syntax-guided optimal synthesis for chemical reaction networks. In *CAV'17*, volume 10427 of *LNCS*, pages 375–395. Springer, 2017.

[CČG+17a] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Designing robust software systems through parametric Markov chain synthesis. In *ICSA'17*, pages 131–140. IEEE Computer Society, 2017.

[CČG+17b] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. RODES: A robust-design synthesis tool for probabilistic systems. In *QEST'17*, volume 10503 of *LNCS*, pages 304–308. Springer, 2017.

[CČG+18] Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software*, 143:140–158, 2018.

[CCM+12] Iadine Chades, Josie Carwardine, Tara G. Martin, Samuel Nicol, Régis Sabbadin, and Olivier Buffet. MOMDPs: A solution for modelling adaptive management problems. In *AAAI*. AAAI Press, 2012.

[CCP+16] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, and et al. A cloud-scale acceleration architecture. In *MICRO'16*. IEEE Press, 2016.

[CDKB18] Philipp Chrszon, Clemens Dubslaff, Sascha Klüppelholz, and Christel Baier. ProFeat: feature-oriented engineering for family-based probabilistic model checking. *Formal Asp. Comput.*, 30(1):45–75, 2018.

[ČDKP14] Milan Češka, Frits Dannenberg, Marta Kwiatkowska, and Nicola Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *CMSB'14*, volume 8859 of *LNCS*, pages 86–98. Springer, 2014.

[ČDP+17] Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017.

[CGP05] Yang Cao, Daniel T Gillespie, and Linda R Petzold. The slow-scale stochastic simulation algorithm. *The Journal of chemical physics*, 122(1):014116, 2005.

[CHH+13] Taolue Chen, Ernst Moritz Hahn, Tingting Han, Marta Kwiatkowska, Hongyang Qu, and Lijun Zhang. Model repair for markov decision processes. In *TASE'13*, pages 85–92. IEEE, 2013.

[ČHH+18] Milan Češka, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar. Approximate Reduction of Finite Automata for High-Speed Network Intrusion Detection. In *TACAS'18*, volume 10806 of *LNCS*, pages 155–175. Springer, 2018.

[ČHH+19a] Milan Češka, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar. Approximate reduction of finite automata for high-speed network intrusion detection. *International Journal on Software Tools for Technology Transfer*, 2019.

[ČHH+19b] Milan Češka, Vojtěch Havlena, Lukáš Holík, Jan Kořenek, Ondřej Lengál, Denis Matoušek, Jiří Matoušek, Jakub Semrič, and Tomáš Vojnar. Deep packet inspection in fpgas via approximate nondeterministic automata. In *FCCM'19*, pages 109–117. IEEE, 2019.

[ČHJK19] Milan Češka, Christian Hensel, Sebastian Junges, and Joost-Pieter Katoen. Counterexample-driven synthesis for probabilistic program sketches. In *Formal Methods – The Next 30 Years*, volume 11800 of *LNCS*, pages 101–120. Springer, 2019.

[Cho17] Ventsislav Chonev. Reachability in augmented interval Markov chains. *CoRR*, abs/1701.02996, 2017.

[ČJJK19] Milan Češka, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Shepherding hordes of Markov chains. In *TACAS'19*, volume 11428 of *LNCS*, pages 172–190. Springer, 2019.

[ČK19] Milan Češka and Jan Křetínský. Semi-quantitative abstraction and analysis of chemical reaction networks. In *CAV'19*, volume 11561 of *LNCS*, pages 475–496. Springer, 2019.

[CKL16] Luca Cardelli, Marta Kwiatkowska, and Luca Laurenti. A stochastic hybrid approximation for chemical kinetics based on the linear noise approximation. In *CMSB'16*, volume 9859 of *LNCS*, pages 147–167. Springer, 2016.

[CM13] Lorenzo Clemente and Richard Mayr. Advanced Automata Minimization. In *POPL'13*, pages 63–74. ACM, 2013.

[ČMM+17] Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In *ICCAD'17*, pages 416–423. IEEE, 2017.

[ČMM+18] Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. ADAC: Automated design of approximate circuits. In *CAV'18*, volume 10981 of *LNCS*. Springer, 2018.

[ČMM+20] Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. Adaptive verifiability-driven strategy for evolutionary approximation of arithmetic circuits. *CoRR*, abs/2003.02491, 2020. *Submitted to Applied Soft Computing*.

[CNCS11] Attila Csikász-Nagy, Luca Cardelli, and Orkun S Soyer. Response dynamics of phosphorelays suggest their potential utility in cell signalling. *J. R. Soc. Interface*, 8(57):480–488, 2011.

[CP17] Sarah Chasins and Phitchaya Mangpo Phothilimthana. Data-driven synthesis of full probabilistic programs. In *CAV'17 (1)*, volume 10426 of *LNCS*, pages 279–304. Springer, 2017.

[ČPP+16] Milan Češka, Petr Pilař, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. PRISM-PSY: Precise GPU-accelerated parameter synthesis for stochastic systems. In *TACAS'16*, volume 9636 of *LNCS*, pages 367–384. Springer, 2016.

[CS03] Christopher R. Clark and David E. Schimmel. Efficient Reconfigurable Logic Circuits for Matching Complex Network Intrusion Detection Patterns. In *FPL'03*, volume 2778 of *LNCS*, pages 956–959. Springer, 2003.

[CS+16] Arun Chandrasekharan, Mathias Soeken, et al. Precise error determination of approximated components in sequential circuits with model checking. In *DAC'16*, pages 129:1–129:6. ACM, 2016.

[ČŠDB14] Milan Češka, David Šafránek, Sven Dražan, and Luboš Brim. Robustness analysis of stochastic biochemical systems. *PloS one*, 9(4), 2014.

[CSGD16a] Arun Chandrasekharan, Mathias Soeken, Daniel Große, and Rolf Drechsler. Approximation-aware rewriting of aigs for error tolerant applications. In *ICCAD'16*, pages 1–8. IEEE, 2016.

[CSGD16b] Arun Chandrasekharan, Mathias Soeken, Daniel Große, and Rolf Drechsler. Precise error determination of approximated components in sequential circuits with model checking. In *DAC'16*, pages 129:1–129:6. ACM, 2016.

[CTTV17] Luca Cardelli, Mirco Tribastone, Max Tschaikowski, and Andrea Vandin. Maximal aggregation of polynomial dynamical systems. *Proceedings of the National Academy of Sciences*, 114(38):10029–10034, 2017.

[CYB+15] Maciej Ciesielski, Cunxi Yu, Walter Brown, Duo Liu, and André Rossi. Verification of gate-level arithmetic circuits by function extraction. In *DAC'15*. ACM, 2015.

[DHF19] Elisabeth Degrand, Mathieu Hemery, and François Fages. On chemical reaction network design by a nested evolution algorithm. In *CMSB'19*, volume 11773 of *LNCS*, pages 78–95. Springer, 2019.

[DJJ+15] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. PROPhESY: A PRObabilistic ParamEter SYNnthesis Tool. In *CAV'15*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.

[DJKV17] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A STORM is coming: A modern probabilistic model checker. In *CAV'17*, volume 10427 of *LNCS*, pages 592–600. Springer, 2017.

[DJW+14] Christian Dehnert, Nils Jansen, Ralf Wimmer, Erika Ábrahám, and Joost-Pieter Katoen. Fast debugging of PRISM models. In *ATVA'14*, volume 8837 of *LNCS*, pages 146–162. Springer, 2014.

[DLT08] Josée Desharnais, François Laviolette, and Mathieu Tracol. Approximate analysis of probabilistic processes: logic, simulation and games. In *QEST'08*, pages 264–273. IEEE, 2008.

[DMPY15] Neil Dalchau, Niall Murphy, Rasmus Petersen, and Boyan Yordanov. Synthesizing and tuning chemical reaction networks with specified behaviours. In *DNA'15*, volume 9211 of *LNCS*, pages 16–33. Springer, 2015.

[DMY+14] S-J Dunn, Graziano Martello, Boyan Yordanov, Stephen Emmott, and AG Smith. Defining an essential transcription factor program for naive pluripotency. *Science*, 344(6188):1156–1160, 2014.

[DPAM02] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comp.*, 6(2):182–197, 2002.

[EFH08] Andreas Eggers, Martin Fränzle, and Christian Herde. SAT modulo ODE: A direct SAT approach to hybrid systems. In *ATVA'08*, volume 5311 of *LNCS*, pages 171–185. Springer, 2008.

[EK09] Stewart N. Ethier and Thomas G. Kurtz. *Markov processes: characterization and convergence*, volume 282. John Wiley & Sons, 2009.

[Eng06] Stefan Engblom. Computing the moments of high dimensional solutions of the master equation. *Applied Mathematics and Computation*, 180(2):498 – 515, 2006.

[ERNF15] Andreas Eggers, Nacim Ramdani, Nedialko S. Nedialkov, and Martin Fränzle. Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods. *Software and Systems Modeling*, 14(1):121–148, 2015.

[FAF13] Farzad Farshchi, Muhammad Saeed Abrishami, and Sied Mehdi Fakhraie. New approximate multiplier for low power digital signal processing. In *CADS'13*, pages 25–30. IEEE, 2013.

[FGD18] Saman Frohlich, Daniel Grosse, and Rolf Drechsler. Approximate hardware generation using symbolic computer algebra employing grobner basis. In *DATE'18*, pages 889–892. IEEE, 2018.

[FL09] Lars Ferm and Per Lötstedt. Adaptive solution of the master equation in low dimensions. *Applied Numerical Mathematics*, 59(1):187–204, 2009.

[FTG16] Antonio Filieri, Giordano Tamburrelli, and Carlo Ghezzi. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Trans. Softw. Eng.*, 42(1):75–99, 2016.

[GAC12] Sicun Gao, Jeremy Avigad, and Edmund M Clarke. δ-complete decision procedures for satisfiability over the reals. In *IJCAR'12*, volume 7364 of *LNCS*, pages 286–300. Springer, 2012.

[GAK15] Arnab Ganguly, Derya Altintan, and Heinz Koeppl. Jump-diffusion approximation of stochastic reaction dynamics: error bounds and algorithms. *Multiscale Modeling & Simulation*, 13(4):1390–1419, 2015.

[GCT18] Simos Gerasimou, Radu Calinescu, and Giordano Tamburrelli. Synthesis of probabilistic models for quality-of-service software engineering. *Autom. Softw. Eng.*, 25(4):785–831, 2018.

[GDF14] Sergio Giro, Pedro R. D'Argenio, and Luis María Ferrer Fioriti. Distributed probabilistic input/output automata: Expressiveness, (un)decidability and algorithms. *Theor. Comput. Sci.*, 538:84–102, 2014.

[GGG+15] Mirco Giacobbe, Călin C Guet, Ashutosh Gupta, Thomas A Henzinger, Tiago Paixao, and Tatjana Petrov. Model checking gene regulatory networks. In *TACAS'15*, volume 9035 of *LNCS*, pages 469–483. Springer, 2015.

[GGL03] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. In *SOSP'03*, pages 29–43, 2003.

[Gil77a] Daniel T. Gillespie. Exact Stochastic Simulation of Coupled Chemical Reactions. *Journal of Physical Chemistry*, 81(25):2340–2381, 1977.

[Gil77b] Daniel T Gillespie. Exact stochastic simulation of coupled chemical reactions. *The journal of physical chemistry*, 81(25):2340–2361, 1977.

[GKC13] Sicun Gao, Soonho Kong, and Edmund M Clarke. dReal: An SMT solver for nonlinear theories over the reals. In *CADE'13*, volume 7898 of *LNCS*, pages 208–214. Springer, 2013.

[GMRR13] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, Jan 2013.

[Gou05] John Goutsias. Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. *The Journal of chemical physics*, 122(18):184102, 2005.

[GPS17] Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1-2):1–119, 2017.

[GPZC05] Ido Golding, Johan Paulsson, Scott M Zawilski, and Edward C Cox. Real-time kinetics of gene activity in individual bacteria. *Cell*, 123(6):1025–1036, 2005.

[GS13] Carlo Ghezzi and Amir Molzam Sharifloo. Model-based verification of quantitative non-functional properties for software product lines. *Information & Software Technology*, 55(3):508–524, 2013.

[GT02] Swapna S. Gokhale and Kishor S. Trivedi. Reliability prediction and sensitivity analysis based on software architecture. In *ISSRE'03*, pages 64–75, 2002.

[Gur18] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2018.

[GZLS11] Saumil J Gandhi, Daniel Zenklusen, Timothée Lionnet, and Robert H Singer. Transcription of functionally related constitutive genes is not coordinated. *Nature structural & molecular biology*, 18(1):27, 2011.

[HD18] Qinheping Hu and Loris D'Antoni. Syntax-guided synthesis with quantitative syntactic objectives. In *CAV'18*, volume 10981 of *LNCS*, pages 386–403. Springer, 2018.

[Her90] Ted Herman. Probabilistic self-stabilization. *Inf. Process. Lett.*, 35(2):63–67, 1990.

[HGK15] Benjamin Hepp, Ankit Gupta, and Mustafa Khammash. Adaptive hybrid simulations for multiscale stochastic reaction networks. *The Journal of chemical physics*, 142(3):034118, 2015.

[HHZ11] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric Markov models. *Software Tools for Technology Transfer*, 13(1):3–19, 2011.

[HJ94] Hans Hansson and Bengt Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[HJ12] Cheng-Shen Han and Jie-Hong Roland Jiang. When boolean satisfiability meets gaussian elimination in a simplex way. In *CAV'12*, volume 7358 of *LNCS*, pages 410–426. Springer, 2012.

[HKM08] Tingting Han, Joost-Pieter Katoen, and Alexandru Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *RTSS'08*, pages 173–182. IEEE, 2008.

[HKN+08] John Heath, Marta Kwiatkowska, Gethin Norman, David Parker, and Oksana Tymchyshyn. Probabilistic model checking of complex biological pathways. *Theoretical Computer Science*, 391(3):239–257, 2008.

[HL05] Chin-Yu Huang and M. R. Lyu. Optimal testing resource allocation, and sensitivity analysis in software development. *Transactions on Reliability*, 54(4):592–603, 2005.

[HMMW10] Thomas A Henzinger, Linar Mikeev, Maria Mateescu, and Verena Wolf. Hybrid numerical solution of the chemical master equation. In *CMSB'10*, pages 55–65. ACM, 2010.

[HMW09] Thomas A. Henzinger, Maria Mateescu, and Verena Wolf. Sliding Window Abstraction for Infinite Markov Chains. In *CAV'09*, volume 5643 of *LNCS*, pages 337–352. Springer, 2009.

[HMZ12] Mark Harman, S. Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Comp. Surveys*, 45(1):11:1–11:61, 2012.

[HWKT14] J. Hasenauer, V. Wolf, A. Kazeroonian, and F. J. Theis. Method of conditional moments (mcm) for the chemical master equation. *Journal of Mathematical Biology*, 69(3):687–735, 2014.

[JHL15] Honglan Jiang, Jie Han, and Fabrizio Lombardi. A comparative review and evaluation of approximate adders. In *GLVLSI'15*, pages 343–348. ACM, 2015.

[JL11] Sumit Kumar Jha and Christopher James Langmead. Synthesis and infeasibility analysis for stochastic models of biochemical systems using statistical model checking and abstraction refinement. *Theor. Comput. Sci.*, 412(21):2162–2187, 2011.

[JS17] Susmit Jha and Sanjit A. Seshia. A theory of formal synthesis via inductive learning. *Acta Informatica*, 54(7), 2017.

[KGP03] S. Kamavaram and Katerina Goseva-Popstojanova. Sensitivity of software usage to changes in the operational profile. In *NFM'03*. IEEE, 2003.

[Kit04] Hiroaki Kitano. Biological robustness. *Nature Reviews Genetics*, 5:826–837, 2004.

[KLC98] Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artif. Intell.*, 101(1-2):99–134, 1998.

[KM32] William Ogilvy Kermack and Anderson G. McKendrick. Contributions to the mathematical theory of epidemics. II. the problem of endemicity. *Proceedings of the Royal society of London. Series A*, 138(834):55–83, 1932.

[KM69] Richard M Karp and Raymond E Miller. Parallel program schemata. *Journal of Computer and system Sciences*, 3(2):147–195, 1969.

[KNP07] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic Model Checking. In *SFM'07*, volume 4486 of *LNCS*, pages 220–270. Springer Berlin Heidelberg, 2007.

[KNP11] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV'11*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[KNP12] Marta Kwiatkowska, Gethin Norman, and David Parker. Probabilistic verification of herman's self-stabilisation algorithm. *Formal Aspects of Computing*, 24(4):661–670, 2012.

[KNPV09] Marta Kwiatkowska, Gethin Norman, David Parker, and Maria Grazia Vigliotti. Probabilistic mobile ambients. *Theor. Comput. Sci.*, 410(12-13):1272–1303, 2009.

[KPS+13] Ali Sinan Koksal, Yewen Pu, Saurabh Srivastava, Rastislav Bodik, Jasmin Fisher, and Nir Piterman. Synthesis of biological models from mutation experiments. In *POPL'13*, pages 469–482. ACM, 2013.

[LHC+05] Jung-Hua Lo, Chin-Yu Huang, Ing-Yi Chen, et al. Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure. *Journal of Syst. and Software*, 76(1):3–13, 2005.

[LMS05] Ines Lynce and Joao Marques-Silva. Efficient data structures for backtrack search sat solvers. In *Annals of Mathematics and Artificial Intelligence*, pages 137–152. Kluwer Academic Publishers, 2005.

[LPC+12] Matthew R Lakin, David Parker, Luca Cardelli, Marta Kwiatkowska, and Andrew Phillips. Design and analysis of dna strand displacement devices using probabilistic model checking. *J. R. Soc. Interface*, 9(72), 2012.

[LRY+16] Atieh Lotfi, Abbas Rahimi, Amir Yazdanbakhsh, Hadi Esmaeilzadeh, and Rajesh K. Gupta. Grater: An approximation workflow for exploiting data-level parallelism in FPGA acceleration. In *DATE'16*, pages 1279–1284. EDAA, 2016.

[LS91] Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1), 1991.

[M+] Richard Mayr et al. REDUCE: A Tool for Minimizing Nondeterministic Finite-Word and Büchi Automata. http://languageinclusion.org. [Online: accessed 2020-03-01].

[MAFL10] Hamid Reza Mahdiani, Ali Ahmadi, Sied Mehdi Fakhraie, and Caro Lucas. Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications. *TCAS-I*, 57(4):850–862, 2010.

[Mat] Matt Jonkman et al. SURICATA. https://suricata-ids.org. [Online: accessed 2020-03-01].

[MHVS17] Vojtěch Mrázek, Radek Hrbáček, Zdeněk Vašíček, and Lukáš Sekanina. Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In *DATE'17*, pages 258–261. EDAA, 2017.

[MK06] Brian Munsky and Mustafa Khammash. The finite state projection algorithm for the solution of the chemical master equation. *The Journal of chemical physics*, 124:044104, 2006.

[MKBR10] Anne Martens, Heiko Koziolek, Steffen Becker, and Ralf Reussner. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. In *WOSP/SIPEW*, pages 105–116, 2010.

[MKP16a] Denis Matoušek, Jan Kořenek, and Viktor Puš. High-speed regular expression matching with pipelined automata. In *FPT'16*, pages 93–100. IEEE, 2016.

[MKP16b] Denis Matousek, Jan Korenek, and Viktor Pus. High-speed Regular Expression Matching with Pipelined Automata. In *FPT'16*, pages 93–100. IEEE, 2016.

[MMR+12] Curtis Madsen, Chris J. Myers, Nicholas Roehner, Chris Winstead, and Zhen Zhang. Utilizing stochastic model checking to analyze genetic circuits. In *CIBCB'12*, pages 379–386. IEEE, 2012.

[MPP+18] Niall Murphy, Rasmus Petersen, Andrew Phillips, Boyan Yordanov, and Neil Dalchau. Synthesizing and tuning stochastic chemical reaction networks with specified behaviours. *Journal of The Royal Society Interface*, 15(145):20180283, 2018.

[MSS+16] Vojtěch Mrázek, Syed Shakib Sarwar, Lukáš Sekanina, Zdeněk Vašíček, and Kaushik Roy. Design of power-efficient approximate multipliers for approximate artificial neural networks. In *ICCAD'16*, pages 811–817. ACM, 2016.

[MT00] Julian F. Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming*. Springer, 2000.

[Mur89] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[MWDH10] Maria Mateescu, Verena Wolf, Frederic Didier, and Thomas A. Henzinger. Fast Adaptive Uniformization of the Chemical Master Equation. *IET Systems Biology*, 4(6):441–452, 2010.

[NDL+09] Antonio J. Nebro, Juan J. Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. MOCell: A cellular genetic algorithm for multiobjective optimization. *Journal of Intelligent Systems*, 24(7):726–746, 2009.

[NHT+16] Kumud Nepal, Soheil Hashemi, Hokchhay Tann, R. Iris Bahar, and Sherief Reda. Automated high-level generation of low-power approximate computing circuits. *IEEE Transactions on Emerging Topics in Computing*, 7(1):18–30, 2016.

[NORV15] Aditya V Nori, Sherjil Ozair, Sriram K Rajamani, and Deepak Vijaykeerthy. Efficient synthesis of probabilistic programs. In *PLDI'14*. ACM, 2015.

[NPZ17] Gethin Norman, David Parker, and Xueyi Zou. Verification and control of partially observable probabilistic systems. *Real-Time Systems*, 53(3):354–402, 2017.

[OW08] Joël Ouaknine and James Worrell. Some recent results in metric temporal logic. In *FORMATS'08*, volume 5215 of *LNCS*, pages 1–13. Springer, 2008.

[OZ15] Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. In *PLDI'15*, pages 619–630. ACM, 2015.

[PÁJ+15] Shashank Pathak, Erika Ábrahám, Nils Jansen, Armando Tacchella, and Joost-Pieter Katoen. A greedy approach for the efficient repair of stochastic models. In *NFM'15*, volume 9058 of *LNCS*, pages 295–309. Springer, 2015.

[Pha95] Madhan Shridhar Phadke. *Quality engineering using robust design*. Prentice Hall PTR, 1995.

[PYH+14] Nicola Paoletti, Boyan Yordanov, Youssef Hamadi, Christoph M Wintersteiger, and Hillel Kugler. Analyzing and synthesizing genomic logic functions. In *CAV'14*, volume 8559 of *LNCS*, pages 343–357. Springer, 2014.

[QDJ+16] Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Parameter synthesis for markov models: Faster than ever. In *ATVA'16*, volume 9938 of *LNCS*, pages 50–67, 2016.

[RA03] Christopher V. Rao and Adam P. Arkin. Stochastic chemical kinetics and the quasi-steady-state assumption: application to the gillespie algorithm. *The Journal of chemical physics*, 118(11):4999–5010, 2003.

[RAN+15] Genaína Nunes Rodrigues, Vander Alves, Vinicius Nunes, André Lanna, Maxime Cordy, Pierre-Yves Schobbens, Amir Molzam Sharifloo, and Axel Legay. Modeling and verification for probabilistic properties in software product lines. In *HASE'15*, pages 173–180. IEEE, 2015.

[RS19] Sherief Reda and Muhammad Shafique. *Approximate Circuits – Methodologies and CAD*. Springer, Cham, 2019.

[SAGK+16] Amr Sayed-Ahmed, Daniel Große, Ulrich Kühne, Mathias Soeken, and Rolf Drechsler. Formal verification of integer multipliers by combining Gröbner basis with logic reduction. In *DATE'16*, pages 1048–1053. IEEE, 2016.

[SK05] Howard Salis and Yiannis Kaznessis. Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. *The Journal of chemical physics*, 122(5):054103, 2005.

[SLRBE05] Armando Solar-Lezama, Rodric Rabbah, Rastislav Bodík, and Kemal Ebcioğlu. Programming by sketching for bit-streaming programs. In *PLDI'05*, pages 281–294. ACM, 2005.

[SLTB+06] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *ASPLOS'06*. ACM, 2006.

[SMMA07] Guy Shinar, Ron Milo, María Rodríguez Martínez, and Uri Alon. Input–output robustness in simple bacterial signaling systems. *Proceedings of the National Academy of Sciences*, 104(50):19931–19935, 2007.

[Sno] Snort. (`http://www.snort.org`). Accessed Jul 2019.

[SSW10] David Soloveichik, Georg Seelig, and Erik Winfree. DNA as a universal substrate for chemical kinetics. *Proceedings of the National Academy of Sciences of the United States of America*, 107(12):5393–5398, 2010.

[SWSK11] Ralf Steuer, Steffen Waldherr, Victor Sourjik, and Markus Kollmann. Robust signal processing in living cells. *PLoS computational biology*, 7(11):e1002218, 2011.

[TVKO17] Vu Xuan Tung, To Van Khanh, and Mizuhito Ogawa. rasat: an smt solver for polynomial constraints. *Formal Methods in System Design*, 51(3):462–499, 2017.

[VARR11] Rangharajan Venkatesan, Amit Agarwal, Kaushik Roy, and Anand Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *ICCAD'11*, pages 667–673. ACM, 2011.

[Ver18] Vern Paxson et al. The BRO Network Security Monitor, 2018.

[VK92] Nicolaas Godfried Van Kampen. *Stochastic processes in physics and chemistry*, volume 1. Elsevier, 1992.

[VM17] Zdeněk Vašíček and Vojtěch Mrázek. Trading between quality and non-functional properties of median filter in embedded systems. *Genetic Programming and Evolvable Machines*, 18(1):45–82, 2017.

[VMS17] Zdeněk Vašíček, Vojtěch Mrázek, and Lukáš Sekanina. Towards low power approximate DCT architecture for HEVC standard. In *DATE'17*, pages 1576–1581. EDAA, 2017.

[VS11] Zdeněk Vašíček and Lukáš Sekanina. Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware. *Genetic Programming and Evolvable Machines*, 12(3):305–327, 2011.

[VS15a] Zdeněk Vasíček and Lukáš Sekanina. Circuit approximation using single- and multi-objective cartesian GP. In *EuroGP*, volume 9025 of *LNCS*. Springer, 2015.

[VS15b] Zdeněk Vašíček and Lukáš Sekanina. Evolutionary approach to approximate digital circuits design. *IEEE Transactions on Evolutionary Computation*, 19(3):432–444, 2015.

[VSK18] Marko Vasic, David Soloveichik, and Sarfraz Khurshid. Crn++: Molecular programming language. In *International Conference on DNA Computing and Molecular Programming*, pages 1–18. Springer, 2018.

[VtBLL18] Andrea Vandin, Maurice H. ter Beek, Axel Legay, and Alberto Lluch-Lafuente. Qflan: A tool for the quantitative analysis of highly reconfigurable systems. In *FM*, volume 10951 of *LNCS*, pages 329–337. Springer, 2018.

[WB12] Anton J. Wijs and Dragan Bošnački. Improving GPU sparse matrix-vector multiplication for probabilistic model checking. In *SPIN'12*, volume 7385 of *LNCS*, pages 98–116. Springer, 2012.

[WJÁ+12] Ralf Wimmer, Nils Jansen, Erika Ábrahám, Bernd Becker, and Joost-Pieter Katoen. Minimal critical subsystems for discrete-time Markov models. In *TACAS'12*, volume 7214 of *LNCS*, pages 299–314. Springer, 2012.

[WJV+15] Ralf Wimmer, Nils Jansen, Andreas Vorpahl, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. High-level counterexamples for probabilistic automata. *Logical Methods in Computer Science*, 11(1), 2015.

[YC16] Cunxi Yu and Maciej Ciesielski. Analyzing imprecise adders using BDDs – a case study. In *ISVLSI'16*, pages 152–157. IEEE, 2016.

[YJB+18] Jiajia Yang, Lei Jiang, Xu Bai, Huailiang Peng, and Qiong Dai. A high-performance round-robin regular expression matching architecture based on fpga. In *ISCC'18*, pages 1–7. IEEE, 2018.

[ZL18] Weichao Zhou and Wenchao Li. Safety-aware apprenticeship learning. In *CAV'18*, volume 10981 of *LNCS*, pages 662–680. Springer, 2018.

[ZWC09] Jingwei Zhang, Layne T. Watson, and Yang Cao. Adaptive aggregation method for the chemical master equation. *International Journal of Computational Biology and Drug Design*, 2(2):134–148, 2009.

Part II

SELECTED PAPERS

Acknowledgment of the publishers. Due to copyright reasons, we list below the bibliographic information of all attached papers, acknowledging the original source of the publications:

[BČDŠ13] **Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In *CAV'13*, volume 8044 of *LNCS*, pages 107–123. Springer, 2013.**

[ČDP+17] **Milan Češka, Frits Dannenberg, Nicola Paoletti, Marta Kwiatkowska, and Luboš Brim. Precise parameter synthesis for stochastic biochemical systems. *Acta Informatica*, 54(6):589–623, 2017.**

[ČMM+17] **Milan Češka, Jiří Matyaš, Vojtěch Mrázek, Lukáš Sekanina, Zdeněk Vašíček, and Tomáš Vojnar. Approximating complex arithmetic circuits with formal error guarantees: 32-bit multipliers accomplished. In *ICCAD'17*, pages 416–423. IEEE, 2017.**

[CČF+17] **Luca Cardelli, Milan Češka, Martin Fränzle, Marta Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. Syntax-guided optimal synthesis for chemical reaction networks. In *CAV'17*, volume 10427 of *LNCS*, pages 375–395. Springer, 2017.**

[CČG+18] **Radu Calinescu, Milan Češka, Simos Gerasimou, Marta Kwiatkowska, and Nicola Paoletti. Efficient synthesis of robust models for stochastic systems. *Journal of Systems and Software*, 143:140–158, 2018.**

[ČHH+19a] **Milan Češka, Vojtěch Havlena, Lukáš Holík, Ondřej Lengál, and Tomáš Vojnar. Approximate reduction of finite automata for high-speed network intrusion detection. *International Journal on Software Tools for Technology Transfer*, 2019.**

[ČK19] **Milan Češka and Jan Křetínský. Semi-quantitative abstraction and analysis of chemical reaction networks. In *CAV'19*, volume 11561 of *LNCS*, pages 475–496. Springer, 2019.**

[ČJJK19] **Milan Češka, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Shepherding hordes of Markov chains. In *TACAS'19*, volume 11428 of *LNCS*, pages 172–190. Springer, 2019.**

[ČHJK19] **Milan Češka, Christian Hensel, Sebastian Junges, and Joost-Pieter Katoen. Counterexample-driven synthesis for probabilistic program sketches. In *Formal Methods – The Next 30 Years*, volume 11800 of *LNCS*, pages 101–120. Springer, 2019.**

Exploring Parameter Space of Stochastic Biochemical Systems Using Quantitative Model Checking*

Luboš Brim, Milan Češka, Sven Dražan, and David Šafránek

Systems Biology Laboratory at Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
{brim,xceska,xdrazan,xsafran1}@fi.muni.cz

Abstract. We propose an automated method for exploring kinetic parameters of stochastic biochemical systems. The main question addressed is how the validity of an *a priori* given hypothesis expressed as a temporal logic property depends on kinetic parameters. Our aim is to compute a landscape function that, for each parameter point from the inspected parameter space, returns the quantitative model checking result for the respective continuous time Markov chain. Since the parameter space is in principle dense, it is infeasible to compute the landscape function directly. Hence, we design an effective method that iteratively approximates the lower and upper bounds of the landscape function with respect to a given accuracy. To this end, we modify the standard uniformization technique and introduce an iterative parameter space decomposition. We also demonstrate our approach on two biologically motivated case studies.

1 Introduction

The importance of stochasticity in biochemical processes having low numbers of molecules has resulted in the development of stochastic models [12]. Stochastic biochemical processes can be faithfully modeled as continuous time Markov chains (CTMCs) [9]. Knowledge of stochastic rate constants (model parameters) is important for the analysis of system dynamics. Moreover, knowledge about how change in parameters influences system dynamics (parameter exploration) is of great importance in tuning the stochastic model. Prior knowledge of kinetic parameters is usually limited. The model identification routine thus typically includes parameter estimation based on experimental data. While parameter exploration and estimation is well-established for deterministic models, it has not yet been adequately addressed and sufficiently developed for stochastic models. The purpose of this work is to develop practical and effective methods for exact exploration of model parameters in stochastic biochemical models.

The main question addressed is how the validity of an *a priori* given hypothesis expressed as a temporal property depends on model parameters. Parameter estimation

* This work has been supported by the Czech Science Foundation grant No. GAP202/11/0312. M. Češka has been supported by Ministry of Education, Youth, and Sport project No. CZ.1.07/2.3.00/30.0009 – Employment of Newly Graduated Doctors of Science for Scientific Excellence. D. Šafránek has been supported by EC OP project No. CZ.1.07/2.3.00/20.0256.

N. Sharygina and H. Veith (Eds.): CAV 2013, LNCS 8044, pp. 107–123, 2013.

gives a single point in the parameter space where the values of model parameters maximize the agreement of model behaviour with experimental data. On the contrary, we often do not want to have a single objective but rather explore the property over the entire parameter space. Our main goal is to compute a *landscape function* that for each parameter point from the inspected parameter space returns the quantitative model checking result for the respective CTMC determined by the parameter point and the given property. Since the inspected parameter space is in principle dense the set of parametrized CTMCs to be explored is infinite. It is thus not possible to compute the model checking result for each CTMC individually.

As a temporal logic we use the *bounded time fragment of Continuous Stochastic Logic (CSL)* [2] further extended with rewards [19]. For most cases of biochemical stochastic systems the bounded time restriction is adequate since a typical behaviour is recognizable in finite time intervals.

In this paper we consider the *parameter exploration problem* for stochastic biochemical systems in terms of a landscape function that returns for each parameter point the probability or the expected reward of the inspected CSL formula. We propose a method, called *min-max approximation*, that computes the *lower* and *upper* approximations of the landscape function. To compute the approximation for an arbitrary nested CSL formula, we introduce the largest and smallest set of states satisfying the formula and show how to compute such sets effectively using a new method called *parametrized uniformization*. To compute the landscape function approximation with given accuracy we employ *iterative parameter space decomposition* that divides the parameter space into subspaces and allows to compute the proposed approximation independently for each subspace. This decomposition refines the approximation and enables to reach the required accuracy bound. We demonstrate our approach on two biologically motivated case studies. In the first one, we demonstrate that parametrized uniformization allows to approximate the transient probabilities of Schloegel's model [24] for the inspected parameter space. In the second case, our method is applied to parameter exploration of bi-stability in mammalian cell cycle gene regulatory control [25]. Several techniques have been employed [26,10] to analyze models of this kind, especially, it has been shown that asymptotic solutions may disagree with the exact solution imposing thus a challenge for more accurate computational techniques. Since in low molecular numbers stochasticity can produce behaviour that significantly differs from asymptotic and deterministic dynamics, the parameter exploration method reflecting this phenomenon is very important for computational systems biology.

In contrast to methods mentioned in the related work section, the accuracy of results can be fully controlled and adjusted by the user. Similarly to these methods our method is computationally intensive. However, it can be easily parallelized since the computation for each subspace is independent. Moreover, it can be also combined with fast adaptive uniformization [9] and sliding window abstraction [16].

Related Work. To the best of our knowledge there is no other work on stochastic models employing CSL model checking to systematic parameter exploration. The closest work is [22] where a CTMC is explored with respect to a property formalized as a deterministic timed automaton (DTA). It extends [1] to parameter estimation with respect to acceptance of the DTA. Approaches to parameter estimation [23,1,7] rely on

approximating the maximum likelihood. Their advantage is the possibility to analyse infinite state spaces [1] (employing dynamic state space truncation with numerically computed likelihood) or even models with no prior knowledge of parameter ranges [7] (using Monte-Carlo optimization for computing the likelihood). All these methods are not suitable for computing the landscape function since they focus on optimizing a *single objective* and not on global *exploration* of the entire requested parameter space.

Approaches based on Markov Chain Monte-Carlo sampling and Bayesian inference [13,17,18] can be extended to sample-based approximation of the landscape function, but at the price of undesired inaccuracy and high computational demands [6,4]. Compared to these methods, our method provides an exact result without neglecting any singularities caused by possible discontinuities in the landscape function.

In [15], for a given parametrized discrete time Markov chain (DTMC) the problem of synthesis for a probabilistic temporal logic is considered. The problem is reduced to constructing a regular expression representing the property validity while addressing the problem of expression explosion. Construction of the expression and also the proposed reduction techniques rely on the discrete nature of DTMC. These techniques cannot be successfully applied to CTMC since the complexity of the expression is given by maximal number of events that can occur within the inspected time horizon. In a typical biochemical system where time scales of individual reactions differ in several orders the number of reactions that can occur is enormous.

Barbuti et al. [5] treat stochastic biochemical models with parameter uncertainty in terms of interval discrete time Markov chains. They reduce quantitative reachability analysis of uncertain models to reachability analysis of a Markov Decision Process. However, no analogy to landscape function and automatized parameter decomposition is considered. Moreover, our method deals with continuous time semantics.

2 Background

Stochastic Biochemical Systems. A finite state stochastic biochemical system $\mathcal{S}$ is defined by a set of N *chemical species* in a well stirred volume with fixed size and fixed temperature participating in M *chemical reactions*. The number X_i of molecules of each species S_i has a specific bound and each reaction is of the form $u_1S_1 + \ldots + u_NS_N \longrightarrow v_1S_1 + \ldots + v_NS_N$ where $u_i, v_i \in \mathbb{N}_0$ represent *stoichiometric coefficients*.

A *state* of a system in time t is the vector $\mathbf{X}(t) = (X_1(t), X_2(t), \ldots, X_N(t))$. When a single reaction with index $r \in \{1, \ldots, M\}$ with vectors of stoichiometric coefficients U_r and V_r occurs the state changes from $\mathbf{X}$ to $\mathbf{X}' = \mathbf{X} - U_r + V_r$, which we denote as $\mathbf{X} \xrightarrow{r} \mathbf{X}'$. For such reaction to happen in a state $\mathbf{X}$ all reactants have to be in sufficient numbers and the state $\mathbf{X}'$ must reflect all species bounds. The *reachable state space* of $\mathcal{S}$, denoted as $\mathbb{S}$, is the set of all states reachable by a finite sequence of reactions from *an initial state* $\mathbf{X}_0$. For each state $\mathbf{X}_i$ we denote $\mathsf{pred}(\mathbf{X}_i) = \{(j,r) \mid \mathbf{X}_j \xrightarrow{r} \mathbf{X}_i\}$ and $\mathsf{succ}(\mathbf{X}_i) = \{(j,r) \mid \mathbf{X}_i \xrightarrow{r} \mathbf{X}_j\}$ the sets of all predecessors and successors, respectively, together with indices of corresponding reactions. The set of indices of all reactions changing the state $\mathbf{X}_i$ to the state $\mathbf{X}_j$ is denoted as $\mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j) = \{r \mid \mathbf{X}_i \xrightarrow{r} \mathbf{X}_j\}$. Henceforward the reactions will be referred directly by their indices.

According to Gillespie [12] the behaviour of a stochastic system $\mathcal{S}$ can be described by the continuous time Markov chain (CTMC) $\mathcal{C} = (\mathbb{S}, \mathbf{X}_0, \mathbf{R})$ where the transition matrix $\mathbf{R}(i,j)$ gives the probability of a transition from $\mathbf{X}_i$ to $\mathbf{X}_j$. Formally, $\mathbf{R}(i,j) = \sum_{r \in \mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j)} k_r \cdot C_{r,i}$ such that k_r is *a stochastic rate constant* of the reaction r and $C_{r,i} \stackrel{def}{=} \prod_{l=1}^{N} \binom{\mathbf{X}_{i,l}}{u_l}$ corresponds to the population dependent term of the *propensity function* where $\mathbf{X}_{i,l}$ is lth component of the state $\mathbf{X}_i$ and u_l is the stoichiometric coefficient of the reactant S_l in reaction r.

Parameter Space. Let each stochastic rate constant k_i have a value interval $[k_i^{\perp}, k_i^{\top}]$ with minimal and maximal bounds expressing *uncertainty range* of its value. *A parameter space* $\mathbf{P}$ induced by a set of stochastic rate constants k_i is defined as the Cartesian product of the individual value intervals $\mathbf{P} = \prod_{i=1}^{M} [k_i^{\perp}, k_i^{\top}]$. A single *parameter point* $p \in \mathbf{P}$ is an M-tuple holding a single value of each rate constant $p = (k_{1_p}, \ldots, k_{M_p})$. We consider only independent parameters, however, if correlated parameters can be expressed as linear functions then our method can be still applied.

A stochastic system $\mathcal{S}_p$ with its stochastic rate constants set to the point $p \in \mathbf{P}$ is represented by a CTMC $\mathcal{C}_p = (\mathbb{S}, \mathbf{X}_0, \mathbf{R}_p)$ where transition matrix $\mathbf{R}_p$ is defined as $\mathbf{R}_p(i,j) = \sum_{r \in \mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j)} k_{r_p} \cdot C_{r,i}$. A *set of parametrized* CTMCs induced by the parameter space $\mathbf{P}$ is defined as $\mathbf{C} = \{\mathcal{C}_p \mid p \in \mathbf{P}\}$. Henceforward, the states $\mathbf{X}_i \in \mathbb{S}$ will be denoted as s_i.

Uniformization. Uniformization is a standard technique that for a given CTMC $\mathcal{C} = (\mathbb{S}, s_0, \mathbf{R})$ computes the transient probability in time t. For an initial state s_0 it returns a vector $\pi^{\mathcal{C},s_0,t}$ such that $\pi^{\mathcal{C},s_0,t}(s') = Pr_{s_0}\{\omega \in Path^{\mathcal{C}}(s_0) \mid \omega@t = s'\}$ for all states $s' \in \mathbb{S}$, where Pr_{s_0} is a unique probability measure on all paths ω starting in state s_0 (denoted as $Path^{\mathcal{C}}(s_0)$) defined, e.g., in [21] and $\omega@t$ is the state on path ω occupied at time t.

The transient probability in time t is obtained as a sum of expressions giving the state distributions after i discrete reaction steps weighted by the ith Poisson probability $\gamma_{i,q\cdot t} = e^{-q\cdot t} \cdot \frac{(q\cdot t)^i}{i!}$, the probability of i such steps occurring up to t, given the delay is exponentially distributed with *rate* q. Formally, $\pi^{\mathcal{C},s_0,t} = \sum_{i=0}^{\infty} \gamma_{i,q\cdot t} \cdot \pi^{\mathcal{C},s_0,0} \cdot (\mathbf{Q}^{\mathsf{unif}(\mathcal{C})})^i \approx \sum_{i=L_\varepsilon}^{R_\varepsilon} \gamma_{i,q\cdot t} \cdot \pi^{\mathcal{C},s_0,0} \cdot (\mathbf{Q}^{\mathsf{unif}(\mathcal{C})})^i$ where $\mathbf{Q}^{\mathsf{unif}(\mathcal{C})}$ is an *uniformized infinitesimal generator matrix* defined as follows: $\mathbf{Q}^{\mathsf{unif}(\mathcal{C})}(s,s') = \frac{\mathbf{R}(s,s')}{q}$, if $s \neq s'$, and $1 - \sum_{s'' \neq s} \frac{\mathbf{R}(s,s'')}{q}$, otherwise, where $q \geq max\{E^{\mathcal{C}}(s) \mid s \in \mathbb{S}\}$ such that $E^{\mathcal{C}}(s) = \sum_{s' \in \mathbb{S}} \mathbf{R}(s,s')$ is *an exit rate* of the state s in CTMC $\mathcal{C}$. Although the sum is in general infinite, for a given precision ε the upper and lower bounds $L_\varepsilon, R_\varepsilon$ can be estimated by using techniques such as of Fox and Glynn [11] which also allow for efficient computation of Poisson probabilities $\gamma_{i,q\cdot t}$. In order to make the computation feasible the matrix-matrix multiplication is reduced to a vector-matrix multiplication by pre-multiplying, i.e., $\pi^{\mathcal{C},s_0,0} \cdot (\mathbf{Q}^{\mathsf{unif}(\mathcal{C})})^i = (\pi^{\mathcal{C},s_0,0} \cdot (\mathbf{Q}^{\mathsf{unif}(\mathcal{C})})^{i-1}) \cdot \mathbf{Q}^{\mathsf{unif}(\mathcal{C})}$.

Property Specification. We consider the *bounded time fragment of CSL with rewards*, see [19] for definition of CSL with rewards. The fragment syntax is defined as follows. A state formula Φ is given as $\Phi ::= \mathsf{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathsf{P}_{\sim p}[\phi] \mid \mathsf{R}_{\sim r}[\mathsf{C}^{\leq t}] \mid \mathsf{R}_{\sim r}[\mathsf{I}^{=t}]$

where ϕ is a path formula given as $\phi ::= \mathsf{X}\,\Phi \mid \Phi\ \mathsf{U}^I\ \Phi$, a is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $p \in [0,1]$ is a probability, $r \in \mathbb{R}_{\geq 0}$ is an expected reward and $I = [a,b]$ is a bounded time interval such that $a, b \in \mathbb{R}_{\geq 0} \wedge a \leq b$. Operators G and F can be derived in the standard way. In order to specify the reward properties, CTMCs are enhanced with reward (cost) structures. Two types of reward structure are used. *A state reward* $\rho(s)$ defines the rate with which a reward is acquired in state $s \in \mathbb{S}$. A reward of $t \cdot \rho(s)$ is acquired if a CTMC remains in state s for t time units. *A transition reward* $\iota(s_i, s_j)$ defines the reward acquired each time the transition (s_i, s_j) occurs.

Let $\mathcal{C} = (\mathbb{S}, s_0, \mathbf{R}, L)$ be a labelled CTMC such that L is a labelling function which assigns to each state $s \in \mathbb{S}$ the set $L(s)$ of atomic propositions that are valid in state s. A state s satisfies $\mathsf{P}_{\sim p}[\phi]$ (denoted as $s \vDash \mathsf{P}_{\sim p}[\phi]$) iff $Prob^{\mathcal{C}}(s,\phi) \stackrel{def}{=} Pr_s\{\omega \in Path^{\mathcal{C}}(s) \mid \omega \vDash \phi\}$ satisfies $\sim p$. A path ω satisfies $\mathsf{X}\,\Phi$ iff $\omega(1) \vDash \Phi$ where $\omega(1)$ is the second state on ω. A path ω satisfies $\Phi\ \mathsf{U}^I\ \Psi$ iff $\exists t \in I.(\omega@t \vDash \Psi \wedge \forall t' \in [0,t).(\omega@t' \vDash \Phi))$.

Intuitively, a state $s \vDash \mathsf{R}_{\sim p}[\mathsf{C}^{\leq t}]$ iff the sum of expected rewards over $Path^{\mathcal{C}}(s)$ *cumulated* until t time units (denoted as $Exp^{\mathcal{C}}(s, \mathsf{X}_{\mathsf{C}^{\leq t}})$) satisfies $\sim p$. Similarly, a state $s \vDash \mathsf{R}_{\sim p}[\mathsf{I}^{=t}]$ iff the sum of expected rewards over all paths $\omega \in Path^{\mathcal{C}}(s)$ at time t (denoted as $Exp^{\mathcal{C}}(s, \mathsf{X}_{\mathsf{I}^{=t}})$) satisfies $\sim p$. A set $Sat_{\mathcal{C}}(\Phi) = \{s \in \mathbb{S} \mid s \vDash \Phi\}$ denotes the set of states that satisfy Φ.

The formal semantics of this fragment is defined similarly as the semantics of full CSL and thus we refer the readers to original papers. In the following text all references to CSL address this fragment. Model checking of CSL can be easily reduce to the computation of transient probability, see [3,21] for more details.

3 Parameter Exploration

In this paper we propose an effective method for systematic and fully automatic parameter exploration of a given stochastic system with respect to a specified temporal property and a parameter space. Let $\mathbf{C}$ be a set of parametrized CTMCs describing the dynamics of the stochastic system $\mathcal{S}$ induced by the inspected parameter space $\mathbf{P}$ and a CSL formula Φ expressing the required behaviour. The problem of *parameter exploration* is as follows: for each state $s \in \mathbb{S}$ compute the *landscape function* $\lambda_s^{\Phi,\mathbf{P}} : \mathbf{P} \to \mathbb{R}_{\geq 0}$ that for each parameter point $p \in \mathbf{P}$ returns the numerical value of the probability or the expected reward for the formula Φ. It means that we consider "quantitative" formulae in the form $\Phi ::= \mathsf{P}_{=?}[\phi] \mid \mathsf{R}_{=?}[\mathsf{C}^{\leq t}] \mid \mathsf{R}_{=?}[\mathsf{I}^{=t}]$, i.e., the topmost operator of the formula Φ returns a quantitative result, as used, e.g., in PRISM [20]. Note that the formula Φ can contain nested probabilistic and reward operators whose evaluations define discrete sets of states further used in the computation of the resulting numerical value. Therefore, the corresponding landscape function is not in general continuous but only piecewise continuous. Also note that the landscape function is inherently bounded.

To solve the parameter exploration problem we extend *global quantitative model checking* techniques enabling to compute for all states of a CTMC the numerical value of the probability or the expected reward for formula Φ. The most crucial part of the problem is given by the fact that the parameter space $\mathbf{P}$ is continuous and thus the set $\mathbf{C}$ is infinite. Therefore, it is not possible to employ the global quantitative model checking techniques for each CTMC $\mathcal{C}_p \in \mathbf{C}$ individually.

Our approach to this problem is based on a new technique which we call *min-max approximation*. The key idea is to approximate the landscape function $\lambda_s^{\Phi,\mathbf{P}}$ using a lower bound $\overline{min}_s^{\Phi,\mathbf{P}} = min\{\lambda_s^{\Phi,\mathbf{P}}(p) \mid p \in \mathbf{P}\}$ and an upper bound $\overline{max}_s^{\Phi,\mathbf{P}} = max\{\lambda_s^{\Phi,\mathbf{P}}(p) \mid p \in \mathbf{P}\}$. Since the computation of the exact bounds is computationally infeasible, we further approximate these bounds, i.e., we compute approximations $min_s^{\Phi,\mathbf{P}}$ and $max_s^{\Phi,\mathbf{P}}$ such that $min_s^{\Phi,\mathbf{P}} \leq \overline{min}_s^{\Phi,\mathbf{P}}$ and $max_s^{\Phi,\mathbf{P}} \geq \overline{max}_s^{\Phi,\mathbf{P}}$. Although the proposed min-max approximation provides the lower and upper bounds of the landscape function, it introduces an *inaccuracy* with respect to parameter exploration, i.e., such approximation can be insufficient for the inspected parameter space $\mathbf{P}$ and the given formula Φ. Formally, the inaccuracy for a state s is given as the difference $max_s^{\Phi,\mathbf{P}} - min_s^{\Phi,\mathbf{P}}$.

A significant advantage of the min-max approximation is that it allows us to iteratively decrease the inaccuracy to a required bound. The key idea is based on *iterative parameter space decomposition* where the parameter space $\mathbf{P}$ is divided into subspaces that are processed independently. The result of such computation is an approximation of the lower bound $min_s^{\Phi,\mathbf{P}_i}$ and the upper bound $max_s^{\Phi,\mathbf{P}_i}$ for each subspace $\mathbf{P}_i$. Such decomposition provides more precise approximation of the landscape function $\lambda_s^{\Phi,\mathbf{P}}$ and enables to reach the required accuracy bound.

In order to effectively compute the min-max approximation for the given formula we design a new method called *parametrized uniformization* allowing to efficiently approximate the transient probabilities for the set $\mathbf{C}$ of parametrized CTMCs. The key idea is to modify *standard uniformization* [14] in such a way that an approximation of the minimal and maximal transient probability with respect to the set $\mathbf{C}$ can be computed. Moreover, the proposed modification preserves the asymptotic time complexity of standard uniformization. Following the model checking method for non-parametrized CTMC presented in [3,21], the result of parametrized uniformization is further used to obtain the min-max approximation of the landscape function $\lambda_s^{\Phi,\mathbf{P}}$.

We are aware that the landscape function could be computed by using standard uniformization to obtain precise values in grid points which could be afterwards interpolated linearly or polynomially. Using adaptive grid refinement such an approach could also provide an arbitrary degree of precision with computation complexity of the same asymptotic class as our method. However, the obtained result would be a general approximation not providing the strict minimal and maximal upper bounds. On the contrary, our min-max approximation guarantees upper and lower estimates without neglecting any singularities caused by possible discontinuities in the landscape function that we consider to be an important feature.

4 Min-Max Approximation

To effectively compute the proposed min-max approximation for an arbitrary nested CSL formula we introduce *the largest and smallest set of states satisfying property* Φ. Let $\mathbf{C}$ be a set of labelled parametrized CTMCs over the parameter space $\mathbf{P}$ such that $\mathbf{C} = \{\mathcal{C}_p \mid p \in \mathbf{P}\}$ where each $\mathcal{C}_p = (\mathbb{S}, s_0, \mathbf{R}_p, L)$. The maximal set of states satisfying Φ, denoted by $\overline{Sat}_{\mathbf{C}}^{\top}(\Phi)$, is defined as $\overline{Sat}_{\mathbf{C}}^{\top}(\Phi) \stackrel{def}{=} \bigcup_{\mathcal{C}_p \in \mathbf{C}} Sat_{\mathcal{C}_p}(\Phi)$. The minimal set of states satisfying Φ, denoted by $\overline{Sat}_{\mathbf{C}}^{\perp}(\Phi)$, is defined as $\overline{Sat}_{\mathbf{C}}^{\perp}(\phi) \stackrel{def}{=} \bigcap_{\mathcal{C}_p \in \mathbf{C}} Sat_{\mathcal{C}_p}(\Phi)$.

Since the set $\mathbf{C}$ is not finite, this definition is not constructive and does not allow to obtain the sets $\overline{Sat}^{\top}_{\mathbf{C}}(\Phi)$ and $\overline{Sat}^{\bot}_{\mathbf{C}}(\Phi)$. Therefore, we define satisfaction relations $\models_{\top}$ and $\models^{\bot}$ that give an alternative characterization of these sets and allow us to effectively compute their approximations.

For any state $s \in \mathbb{S}$ relations $s \models_{\top} \Phi$ and $s \models_{\bot} \Phi$ are defined inductively by:

$$
\begin{array}{ll|ll}
s \models_{\top} \mathsf{true} \wedge s \models_{\bot} \mathsf{true}, \text{ for all } s \in \mathbb{S} & & s \models_{\top} a \Leftrightarrow s \models_{\bot} a \Leftrightarrow a \in L(s) & \\
s \models_{\top} \neg\Phi & \Leftrightarrow s \not\models_{\bot} \Phi & s \models_{\bot} \neg\Phi & \Leftrightarrow s \not\models_{\top} \Phi \\
s \models_{\top} \Phi \wedge \Psi & \Leftrightarrow s \models_{\top} \Phi \wedge s \models_{\top} \Psi & s \models_{\bot} \Phi \wedge \Psi & \Leftrightarrow s \models_{\bot} \Phi \wedge s \models_{\bot} \Psi \\
s \models_{\top} \mathsf{P}_{\leq p}[\phi] & \Leftrightarrow \overline{Prob}^{\mathbf{C}}_{\bot}(s,\phi) \leq p & s \models_{\bot} \mathsf{P}_{\leq p}[\phi] & \Leftrightarrow \overline{Prob}^{\mathbf{C}}_{\top}(s,\phi) \leq p \\
s \models_{\top} \mathsf{P}_{\geq p}[\phi] & \Leftrightarrow \overline{Prob}^{\mathbf{C}}_{\top}(s,\phi) \geq p & s \models_{\bot} \mathsf{P}_{\geq p}[\phi] & \Leftrightarrow \overline{Prob}^{\mathbf{C}}_{\bot}(s,\phi) \geq p \\
s \models_{\top} \mathsf{R}_{\leq p}[\mathsf{I}^{=t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X}_{\mathsf{I}^{=t}}) \leq p & s \models_{\bot} \mathsf{R}_{\leq p}[\mathsf{I}^{=t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X}_{\mathsf{I}^{=t}}) \leq p \\
s \models_{\top} \mathsf{R}_{\geq p}[\mathsf{I}^{=t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X}_{\mathsf{I}^{=t}}) \geq p & s \models_{\bot} \mathsf{R}_{\geq p}[\mathsf{I}^{=t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X}_{\mathsf{I}^{=t}}) \geq p \\
s \models_{\top} \mathsf{R}_{\leq p}[\mathsf{C}^{\leq t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X}_{\mathsf{C}^{\leq t}}) \leq p & s \models_{\bot} \mathsf{R}_{\leq p}[\mathsf{C}^{\leq t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X}_{\mathsf{C}^{\leq t}}) \leq p \\
s \models_{\top} \mathsf{R}_{\geq p}[\mathsf{C}^{\leq t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X}_{\mathsf{C}^{\leq t}}) \geq p & s \models_{\bot} \mathsf{R}_{\geq p}[\mathsf{C}^{\leq t}] & \Leftrightarrow \overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X}_{\mathsf{C}^{\leq t}}) \geq p
\end{array}
$$

where

$$
\begin{array}{l}
\overline{Prob}^{\mathbf{C}}_{\top}(s,\phi) \stackrel{def}{=} max\{Prob^{\mathcal{C}_p}(s,\phi) \mid \mathcal{C}_p \in \mathbf{C}\} \\
\overline{Prob}^{\mathbf{C}}_{\bot}(s,\phi) \stackrel{def}{=} min\{Prob^{\mathcal{C}_p}(s,\phi) \mid \mathcal{C}_p \in \mathbf{C}\} \\
\overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X}) \stackrel{def}{=} max\{Exp^{\mathcal{C}_p}(s,\mathsf{X}) \mid \mathcal{C}_p \in \mathbf{C}\} \text{ for } \mathsf{X} \in \{\mathsf{X}_{\mathsf{I}^{=t}}, \mathsf{X}_{\mathsf{C}^{\leq t}}\} \\
\overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X}) \stackrel{def}{=} min\{Exp^{\mathcal{C}_p}(s,\mathsf{X}) \mid \mathcal{C}_p \in \mathbf{C}\} \text{ for } \mathsf{X} \in \{\mathsf{X}_{\mathsf{I}^{=t}}, \mathsf{X}_{\mathsf{C}^{\leq t}}\}
\end{array}
$$

By structural induction it can be proved that $\forall s \in \mathbb{S} : s \in \overline{Sat}^{\top}_{\mathbf{C}}(\Phi) \Rightarrow s \models_{\top} \Phi$ and $s \models_{\bot} \Phi \Rightarrow s \in \overline{Sat}^{\bot}_{\mathbf{C}}(\Phi)$. This characterization allows us to define an approximation $Sat^{\top}_{\mathbf{C}}(\Phi)$ and $Sat^{\bot}_{\mathbf{C}}(\Phi)$ in the following way. For all $s \in \mathbb{S} : s \in Sat^{\top}_{\mathbf{C}}(\Phi) \stackrel{def}{\Leftrightarrow} s \models^{*}_{\top} \Phi$ and $s \in Sat^{\bot}_{\mathbf{C}}(\Phi) \stackrel{def}{\Leftrightarrow} s \models^{*}_{\bot} \Phi$ where the definition of $\models^{*}_{\top}$ and $\models^{*}_{\bot}$ differs from the definition of $\models_{\top}$ and $\models_{\bot}$ such that the exact values $\overline{Prob}^{\mathbf{C}}_{\top}(s,\phi)$, $\overline{Prob}^{\mathbf{C}}_{\bot}(s,\phi)$, $\overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X})$ and $\overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X})$ are replaced by approximate values $Prob^{\mathbf{C}}_{\top}(s,\phi)$, $Prob^{\mathbf{C}}_{\bot}(s,\phi)$, $Exp^{\mathbf{C}}_{\top}(s,\mathsf{X})$ and $Exp^{\mathbf{C}}_{\bot}(s,\mathsf{X})$, respectively, that satisfy the following:

$$
\begin{array}{c}
Prob^{\mathbf{C}}_{\top}(s,\phi) \geq \overline{Prob}^{\mathbf{C}}_{\top}(s,\phi) \wedge\ Prob^{\mathbf{C}}_{\bot}(s,\phi) \leq \overline{Prob}^{\mathbf{C}}_{\bot}(s,\phi) \\
Exp^{\mathbf{C}}_{\top}(s,\mathsf{X}) \geq \overline{Exp}^{\mathbf{C}}_{\top}(s,\mathsf{X}) \wedge\ Exp^{\mathbf{C}}_{\bot}(s,\mathsf{X}) \leq \overline{Exp}^{\mathbf{C}}_{\bot}(s,\mathsf{X}) \text{ for } \mathsf{X} \in \{\mathsf{X}_{\mathsf{I}^{=t}}, \mathsf{X}_{\mathsf{C}^{\leq t}}\}.
\end{array}
$$

Since we get that $\forall s \in \mathbb{S} : s \in Sat^{\bot}_{\mathbf{C}}(\Phi) \Rightarrow s \models^{*}_{\bot} \Phi \Rightarrow s \models_{\bot} \Phi \Rightarrow s \in \overline{Sat}^{\bot}_{\mathbf{C}}(\Phi)$ and also $s \in \overline{Sat}^{\top}_{\mathbf{C}}(\Phi) \Rightarrow s \models_{\top} \Phi \Rightarrow s \models^{*}_{\top} \Phi \Rightarrow s \in Sat^{\top}_{\mathbf{C}}(\Phi)$, the sets $Sat^{\top}_{\mathbf{C}}(\Phi)$ and $Sat^{\bot}_{\mathbf{C}}(\Phi)$ give us the correct approximations of the sets $\overline{Sat}^{\top}_{\mathbf{C}}(\Phi)$ and $\overline{Sat}^{\bot}_{\mathbf{C}}(\Phi)$, i.e., $\overline{Sat}^{\top}_{\mathbf{C}}(\Phi) \subseteq Sat^{\top}_{\mathbf{C}}(\Phi)$ and $Sat^{\bot}_{\mathbf{C}}(\Phi) \subseteq \overline{Sat}^{\bot}_{\mathbf{C}}(\Phi)$.

In contrast to the exact values their approximations can be efficiently computed using the parametrized uniformization. Therefore, we can also effectively obtain the approximated sets $Sat^{\top}_{\mathbf{C}}(\Phi)$ and $Sat^{\bot}_{\mathbf{C}}(\Phi)$ that are further used in the computation of the

min-max approximation. Formally $min_s^{\Phi,\mathbf{P}} = Prob_{\bot}^{\mathbf{C}}(s,\phi)$ and $max_s^{\Phi,\mathbf{P}} = Prob_{\top}^{\mathbf{C}}(s,\phi)$ if the topmost operator of the formula Φ is $\mathsf{P}_{=?}[\phi]$. Similarly, $min_s^{\Phi,\mathbf{P}} = Exp_{\bot}^{\mathbf{C}}(s,\mathsf{X})$ and $max_s^{\Phi,\mathbf{P}} = Exp_{\top}^{\mathbf{C}}(s,\mathsf{X})$ for $\mathsf{X} = \mathsf{X}_{\mathsf{C}^{\leq t}}$ and $\mathsf{X} = \mathsf{X}_{\mathsf{I}^{=t}}$ if the topmost operator of the formula Φ is $\mathsf{R}_{=?}[\mathsf{C}^{\leq t}]$ and $\mathsf{R}_{=?}[\mathsf{I}^{=t}]$, respectively.

5 Parametrized Uniformization

The most important step of the proposed min-max approximation is for each state s to compute the values $Prob_{\top}^{\mathbf{C}}(s,\phi)$, $Prob_{\bot}^{\mathbf{C}}(s,\phi)$, $Exp_{\top}^{\mathbf{C}}(s,\mathsf{X})$ and $Exp_{\bot}^{\mathbf{C}}(s,\mathsf{X})$ where $\mathbf{C}$ is an infinite set of parametrized CTMCs, ϕ is an arbitrary path formula and $\mathsf{X} \in \{\mathsf{X}_{\mathsf{C}^{\leq t}}, \mathsf{X}_{\mathsf{I}^{=t}}\}$. In order to efficiently obtain these values we employ parametrized uniformization. It is a technique that for the given set $\mathbf{C}$, state $s \in \mathbb{S}$ and time $t \in \mathbb{R}_{\geq 0}$ computes vectors $\pi_{\top}^{\mathbf{C},s,t}$ and $\pi_{\bot}^{\mathbf{C},s,t}$ such that for each state $s' \in \mathbb{S}$ the following holds:

$$\pi_{\top}^{\mathbf{C},s,t}(s') \geq max\{\pi^{\mathcal{C}_p,s,t}(s') \mid \mathcal{C}_p \in \mathbf{C}\} \wedge \pi_{\bot}^{\mathbf{C},s,t}(s') \leq min\{\pi^{\mathcal{C}_p,s,t}(s') \mid \mathcal{C}_p \in \mathbf{C}\} \quad (1)$$

The key idea of parametrized uniformization is to modify standard uniformization in such a way that for each state s' and in each iteration i of the computation we locally minimize (maximize) the value $\pi^{\mathbf{C},s,t}(s')$ with respect to each $\mathcal{C}_p \in \mathbf{C}$. It means that in the ith iteration of the computation for a state s' we consider only the minimal (maximal) values of the relevant states in the iteration $i-1$, i.e., the states that affect the value of state s'. We show that the local minimum and maximum can be efficiently computed and that it gives us values satisfying Equation 1.

Let $\mathbf{Q}^{\mathsf{unif}(\mathbf{C})}$ be an uniformized infinitesimal generator matrix for a set $\mathbf{C}$ of parametrized CTMCs defined as follows:

$$\mathbf{Q}^{\mathsf{unif}(\mathbf{C})}(i,j) = \begin{cases} \sum_{r \in \mathsf{reac}(\mathbf{X}_i,\mathbf{X}_j)} k_r \cdot \frac{C_{r,i}}{q_{max}} & \text{if } i \neq j \\ 1 - \sum_{l \neq i} \sum_{r \in \mathsf{reac}(\mathbf{X}_i,\mathbf{X}_l)} k_r \cdot \frac{C_{r,i}}{q_{max}} & \text{otherwise.} \end{cases} \quad (2)$$

where $q_{max} \geq E_{max} = max\{E^{\mathcal{C}_p}(s) \mid \mathcal{C}_p \in \mathbf{C}, s \in \mathbb{S}\}$ and k_r is a variable from $[k_r^{\bot}, k_r^{\top}]$.

For sake of simplicity, we present only the method allowing to efficiently compute the vector $\pi_{\top}^{\mathbf{C},s,t}$, since the computation of $\pi_{\bot}^{\mathbf{C},s,t}$ is symmetric. We start with the trivial observation that vectors $\pi^{\mathcal{C}_p,s,0}$ (initial probability distributions, e.g., $\pi^{\mathcal{C}_p,s,0}(s') = 1$, if $s = s'$, and 0, otherwise) are equal for all $\mathcal{C}_p \in \mathbf{C}$. Therefore $\pi^{\mathbf{C},s,0} = \pi_{\top}^{\mathbf{C},s,0} = \pi^{\mathcal{C}_p,s,0}$ for all $\mathcal{C}_p \in \mathbf{C}$. In order to present parametrized uniformization, we introduce an operator $\odot_{\top}$ such that for each $s' \in \mathbb{S}$ the following holds:

$$\left(\pi_{\top}^{\mathbf{C},s,0} \odot_{\top} \left(\mathbf{Q}^{\mathsf{unif}(\mathbf{C})}\right)^i\right)(s') \geq max\left\{\left(\pi^{\mathcal{C}_p,s,0} \cdot \left(\mathbf{Q}^{\mathsf{unif}(\mathcal{C}_p)}\right)^i\right)(s') \mid \mathcal{C}_p \in \mathbf{C}\right\}.$$

Moreover, we further require that vectors from the previous iteration can be used, in particular, $\pi_{\top}^{\mathbf{C},s,0} \odot_{\top} (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^i = \left(\pi_{\top}^{\mathbf{C},s,0} \odot_{\top} (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^{i-1}\right) \odot_{\top} \mathbf{Q}^{\mathsf{unif}(\mathbf{C})}$. The operator $\odot_{\top}$ returns a vector $\pi'_{\top} \in \mathbb{R}_{\geq 0}^{|\mathbb{S}|}$ containing for each state $s_i \in \mathbb{S}$ the maximal possible probability after a single discrete step of a DTMC obtained by uniformization of any CTMC

$\mathcal{C}_p$, i.e., $\left(\pi \odot_\top \mathbf{Q}^{\mathsf{unif}(\mathbf{C})}\right)(s) \stackrel{def}{=} max\left\{\left(\pi \cdot \mathbf{Q}^{\mathsf{unif}(\mathcal{C}_p)}\right)(s) \mid p \in \mathbf{P}\right\} = \pi'_\top(s)$ where π is a general vector. Since $\sum \pi'_\top(i) \geq 1$, the vector $\pi'_\top$ is no longer a state distribution.

To show how the operator $\odot_\top$ is computed let $\sigma(s_i)$ be an algebraic expression defined as the part of the vector-matrix multiplication for state s_i. For each $s_i \in \mathbb{S}$ we get $\sigma(s_i) = \left(\pi \cdot \mathbf{Q}^{\mathsf{unif}(\mathbf{C})}\right)(s_i) = \sum_{j=0}^{|\mathbb{S}|-1} \pi(j) \cdot \mathbf{Q}^{\mathsf{unif}(\mathbf{C})}(j,i)$. Rewriting $\sigma(s_i)$ by Equation 2 and using the sets $\mathsf{pred}(s_i)$ and $\mathsf{succ}(s_i)$ we obtain the following:

$$\sigma(s_i) = \sum_{(j,r)\in \mathsf{pred}(s_i)} \pi(j) \cdot k_r \cdot \frac{C_{r,j}}{q_{max}} + \pi(i)\left(1 - \sum_{(j,r)\in \mathsf{succ}(s_i)} k_r \cdot \frac{C_{r,i}}{q_{max}}\right) \tag{3}$$

The first summand in Equation 3, indexed over predecessors of s_i, corresponds to the probability mass inflowing into state s_i through all reactions. The second summand corresponds to the portion of probability mass remaining in s_i from the previous iteration.

The operator $\odot_\top$ locally maximizes expression $\sigma(s)$ for all $s \in \mathbb{S}$ with respect to $\mathbf{P}$, i.e., $\left(\pi \odot_\top \mathbf{Q}^{\mathsf{unif}(\mathbf{C})}\right)(s) = max\{\sigma_p(s) \mid p \in \mathbf{P}\}$ where $\sigma_p(s)$ is the evaluation of $\sigma(s)$ in the parameter point $p = (k_{1_p}, \ldots k_{M_p})$. First, we show that to compute expression $\pi_\top^{\mathbf{C},s,0} \odot_\top (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^i$ it is sufficient to consider only maximal values from the previous iteration, i.e., vector $\pi_\top^{\mathbf{C},s,0} \odot_\top (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^{i-1}$. Note that $\forall s_i \in \mathbb{S}.\ \pi(i) \geq 0$ and $\forall (j,r) \in \mathsf{pred}(s_i) \cup \mathsf{succ}(s_i).\ k_r \geq 0 \ \wedge\ C_{r,j} \geq 0$. Moreover, since $q_{max} \geq E_{max} \geq 0$, we get that $(1 - \sum_{(j,r)\in \mathsf{succ}(s_i)} k_r \cdot \frac{C_{r,i}}{q_{max}}) \geq (1 - \frac{E_{max}}{q_{max}}) \geq 0$. Now, we can see from Equation 3 that in order to maximize $\sigma(s_i)$ maximal values of $\pi(i)$ for each $0 \leq i < |\mathbb{S}|$ have to be taken.

Second, we show how to determine $p = \{k_1, \ldots, k_M\} \in \mathbf{P}$ such that $\sigma(s_i)$ is evaluated as a maximum. Equation 3 can be rewritten in the following way:

$$\sigma(s_i) = \sum_{(j,r)\in \mathsf{in}} k_r \cdot \frac{\pi(j) \cdot C_{r,j}}{q_{max}} + \sum_{(j,r)\in \mathsf{inout}} k_r \cdot \frac{\pi(j) \cdot C_{r,j} - \pi(i) \cdot C_{r,i}}{q_{max}} - \sum_{(j,r)\in \mathsf{out}} k_r \cdot \frac{\pi(i) \cdot C_{r,i}}{q_{max}}$$

where $\mathsf{in} = \mathsf{pred}(s_i) \setminus \mathsf{succ}(s_i), \mathsf{inout} = \mathsf{pred}(s_i) \cap \mathsf{succ}(s_i)$ and $\mathsf{out} = \mathsf{succ}(s_i) \setminus \mathsf{pred}(s_i)$. The three sums range over disjoint sets of reactions. The first sum represents all incoming reactions that do not have an outgoing counterpart, for these $k_r = k_r^\top$, since they only increase $\sigma(s_i)$. The second sum represents reactions flowing into s_i as well as flowing out of s_i. In this case, expression $\pi(j) \cdot C_{r,j} - \pi(i) \cdot C_{r,i}$ has to be evaluated. If it is positive, $k_r = k_r^\top$, otherwise, $k_r = k_r^\bot$. The last sum represents only reactions flowing out of s_i and hence $k_r = k_r^\bot$. The operator $\odot_\top$ is now computed as $\left(\pi \odot_\top \mathbf{Q}^{\mathsf{unif}(\mathbf{C})}\right)(s_i) = \sigma(s_i)$ where each k_r inside $\sigma(s_i)$ is chosen according to the aforementioned rules.

The computation of $\pi_\bot^{\mathbf{C},s,t}$ is symmetric to the case of $\pi_\top^{\mathbf{C},s,t}$. It means that we define the operator $\odot_\bot$ which locally minimize expression $\sigma(s)$ for all $s \in \mathbb{S}$ with respect to $\mathbf{P}$. In order to minimize $\sigma(s_i)$ it is sufficient to inverse the aforementioned rules.

Vectors $\pi_\top^{\mathbf{C},s,t}$ and $\pi_\bot^{\mathbf{C},s,t}$ are now computed similarly as in the case of standard uniformization, i.e., $\pi_\star^{\mathbf{C},s,t} = \sum_{i=L_\varepsilon}^{R_\varepsilon} \gamma_{i,q\cdot t} \cdot \pi^{\mathbf{C},s,0} \odot_\star (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^i$ where $\star \in \{\bot, \top\}$. To obtain the required values $Prob_\top^{\mathbf{C}}(s,\phi)$, $Prob_\bot^{\mathbf{C}}(s,\phi)$, $Exp_\top^{\mathbf{C}}(s,\mathsf{X})$ and $Exp_\bot^{\mathbf{C}}(s,\mathsf{X})$, we employ the standard model checking technique [3,21] where transient probability $\pi^{\mathcal{C},s,t}$ for a non-parametrized CTMC $\mathcal{C}$ is replaced by vectors $\pi_\top^{\mathbf{C},s,t}$ and $\pi_\bot^{\mathbf{C},s,t}$.

Compared to standard uniformization, only a constant amount of additional work has to be performed in order to determine parameter values. Therefore, asymptotic complexity of parametrized uniformization remains the same as standard uniformization.

6 Parameter Space Decomposition

Before we describe parameter space decomposition – a method allowing to reduce the inaccuracy of the proposed min-max approximation – we briefly discuss the key characteristics of parametrized uniformization. The most important fact is that parametrized uniformization for the set $\mathbf{C}$ in general does not correspond to standard uniformization for any CTMC $\mathcal{C}_p \in \mathbf{C}$. The reason is that we consider a behaviour of a parametrized CTMC that has no equivalent counterpart in any particular $\mathcal{C}_p$. First, the parameter k_r in Equation 3 is determined individually for each state. Therefore, in a single iteration $k_r = k_r^{\top}$ for one state and $k_r = k_r^{\bot}$ for another state. Second, the parameter is determined individually for each iteration and thus for a state s_i the parameter k_r can be chosen differently in individual iterations.

Inaccuracy of the proposed min-max approximation related to the computation of parametrized uniformization, called *unification error*, is given as $(max_s^{\Phi,\mathbf{P}} - \overline{max}_s^{\Phi,\mathbf{P}}) + (\overline{min}_s^{\Phi,\mathbf{P}} - min_s^{\Phi,\mathbf{P}})$. Apart from the unification error our approach introduces an inaccuracy related to approximation of the landscape function $\lambda_s^{\Phi,\mathbf{P}_i}$, called *approximation error*, given as $\overline{max}_s^{\Phi,\mathbf{P}} - \overline{min}_s^{\Phi,\mathbf{P}}$. Finally, the *overall error* of the min-max approximation, denoted as $\mathsf{Err}_s^{\Phi,\mathbf{P}}$, is defined as a sum of both errors, i.e., $\mathsf{Err}_s^{\Phi,\mathbf{P}} = max_s^{\Phi,\mathbf{P}} - min_s^{\Phi,\mathbf{P}}$. Fig. 1 illustrates both types of errors. The approximation error is depicted as blue rectangles and the unification error is depicted as the red rectangles.

We are not able to effectively distinguish the proportion of the approximation error and the unification error nor to reduce the unification error as such. Therefore, we design a method based on the parameter space decomposition that allows us to effectively reduce the overall error of the min-max approximation to a user specified *absolute error bound*, denoted as ERR.

In order to ensure that the min-max approximation meets the given absolute error bound ERR, we iteratively decompose the parameter space $\mathbf{P}$ into finitely many subspaces such that $\mathbf{P} = \mathbf{P}_1 \cup \ldots \cup \mathbf{P}_n$ and each partial result satisfies the overall error bound, i.e., $\forall s \in \mathbb{S} : max_s^{\Phi,\mathbf{P}_i} - min_s^{\Phi,\mathbf{P}_i} \leq \text{ERR}$. Therefore, the overall error for each state $s \in \mathbb{S}$ equals to $\mathsf{Err}_s^{\Phi,\mathbf{P}} = \sum_{i=1}^{n} \frac{|\mathbf{P}_i|}{|\mathbf{P}|} \left(max_s^{\Phi,\mathbf{P}_i} - min_s^{\Phi,\mathbf{P}_i} \right) \leq \sum_{i=1}^{n} \frac{|\mathbf{P}_i|}{|\mathbf{P}|} \text{ERR} = \text{ERR}$. Fig. 1 illustrates such a decomposition and demonstrates convergence of $\mathsf{Err}_s^{\Phi,\mathbf{P}_i}$ to 0 provided that the function $\lambda_s^{\Phi,\mathbf{P}_i}$ is continuous.

For sake of simplicity, we present parametric decomposition on the computation of $\pi_{\top}^{\mathbf{C},s,t}$ since it can be easily extended to the computation of $Prob_{\top}^{\mathbf{C}}(s,\phi)$, $Prob_{\bot}^{\mathbf{C}}(s,\phi)$, $Exp_{\top}^{\mathcal{C}}(s,\mathsf{X})$ and $Exp_{\bot}^{\mathcal{C}}(s,\mathsf{X})$. If during the computation in an iteration i for a state $s' \in \mathbb{S}$ holds that $\left(\pi_{\top}^{\mathbf{C},s,0} \odot_{\top} (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^i\right)(s') - \left(\pi_{\bot}^{\mathbf{C},s,0} \odot_{\bot} (\mathbf{Q}^{\mathsf{unif}(\mathbf{C})})^i\right)(s') > \text{ERR}$ we cancel the current computation and decompose the parameter space $\mathbf{P}$ to n subspaces such that $\mathbf{P} = \mathbf{P}_1 \cup \ldots \cup \mathbf{P}_n$. Each subspace $\mathbf{P}_j$ defines a new set of CTMCs $\mathbf{C}_j = \{\mathcal{C}_j \mid j \in \mathbf{P}_j\}$ that is independently processed in a new computation branch. Note that we could

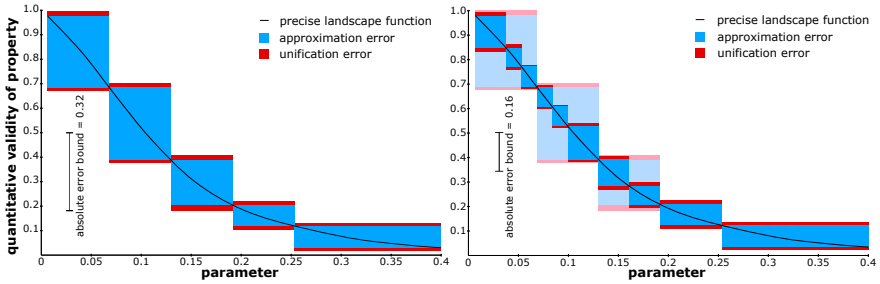


Fig. 1. Illustration of the min-max approximation computation of the landscape function $\lambda_s^{\Phi,\mathbf{P}}$ for an initial state s, property Φ and parameter space $\mathbf{P} = [0, 0.4]$. Left graph shows the decomposition of $\mathbf{P}$ into 5 subspaces for absolute error bound ERR = 0.32. Right graph shows a more refined decomposition for ERR = 0.16 resulting in 10 subspaces. This decomposition reduces both types of errors in each refined subspaces. The exact shape of $\lambda_s^{\Phi,\mathbf{P}}$ is visualized as the black curve.

reuse the previous computation and continue from the iteration $i-1$. However, the most significant part of the error is usually cumulated during the the previous iterations and thus the decomposition would have only a negligible impact on error reduction.

A minimal decomposition with respect to the parameter space $\mathbf{P}$ defines a minimal number of subspaces m such that $\mathbf{P} = \mathbf{P}_1 \cup \ldots \cup \mathbf{P}_m$ and for each subspace $\mathbf{P}_j$ where $1 \leq j \leq m$ holds that $\mathrm{Err}_s^{\Phi,\mathbf{P_j}} \leq \mathrm{ERR}$. Note that the existence of such decomposition is guaranteed only if the landscape function $\lambda_s^{\Phi,\mathbf{P}}$ is continuous. If the landscape function is continuous there can exist more than one minimal decomposition. However, it can not be straightforwardly found. To overcome this problem we have considered and implemented several heuristics allowing to iteratively compute a decomposition satisfying the following: (1) it ensures the required error bound whenever $\lambda_s^{\Phi,\mathbf{P}}$ is continuous, (2) it guarantees the refinement termination in the situation where $\lambda_s^{\Phi,\mathbf{P}}$ is not continuous and the discontinuity causes that ERR can not be achieved. To ensure the termination an additional parameter has to be introduced as a lower bound on the subspace size. Hence this parameter provides a supplementary termination criterion.

7 Case Studies

We implemented our method on top of the tool PRISM 4.0 [20]. We run all experiments on a Linux workstation with an AMD Phenom™ II X4 940 Processor @ 3GHz, 8 GB DDR2 @ 1066 MHz RAM. We used PRISM version 4.0.3. running with sparse engine, since this engine is typically faster than its symbolic counterparts due to efficient matrix vector multiplication.

Schloegl's Model. We use Schlogel's model [24] to demonstrate the practicability of our method for parameter exploration with respect to basic transient analysis. It is the simplest biochemical reaction model for which stochasticity is crucial due to bi-stability

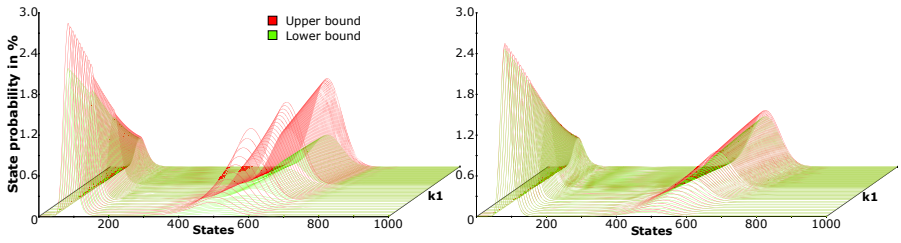


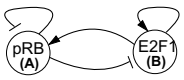
Fig. 2. Species X distribution at 20 time units for $k_1 \in [0.029, 0.031]$ (in s^{-1}). The two presented cases differ in absolute error bound: (left) ERR = 0.01, (right) ERR = 0.001.

– existence of two different steady states to which the species population can (nondeterministically) converge. The model is defined by the following reactions [8]: $2X \xrightarrow{k_1} 3X$, $3X \xrightarrow{k_2} 2X$, $\emptyset \xrightarrow{k_3} X$, $X \xrightarrow{k_4} \emptyset$; $k_1 = 0.03s^{-1}, k_2 = 10^{-4}s^{-1}, k_3 = 200s^{-1}, k_4 = 3.5s^{-1}$. Deterministic formulation of the model by means of ordinary differential equations (ODE) predicts for $k_1 \in [0.0285, 0.035]$ two steady states to which the population converges in the horizon of 20 time units. We will focus on the range $k_1 \in [0.029, 0.031]$. Under the deterministic setting, from any initial state the dynamics evolves to a single steady state. In the noisy setting [26], the population of molecules distributes around both steady states (in short time perspective, here 20 time units). In long time perspective, the population oscillates around both steady states.

We focus on the short time-scale – to analyze the population of X at time 20 starting from the initial state where the number of X is 250. According to the respective ODE model, the population always converges to an asymptotic steady state $X_{st} \leq 1000$. Considering this as an assumption it allows us to bound the state space. The corresponding CTMC has 1001 states and 2000 transitions. The goal of the analysis is to explore how the observed distribution is affected when perturbing k_1 in the range $[0.029, 0.031]$. By executing our method for the absolute error bound ERR $= 0.01$ we got the result visualized in Fig. 2 (left). It can be directly seen that for each parameter point there is a non-zero probability that some individuals reside near the higher steady state while some reside near the lower steady state at time 20. The jumps that are mostly observable in distributions around the higher steady state are caused by the approximation error. Computation with a one order lower error gives a smooth result, see Fig. 2 (right).

The computation required $7.36 \cdot 10^5$ iterations of the parametrized uniformization. The parameter decomposition resulted in 76 subspaces for ERR $= 0.01$ and 639 subspaces for ERR $= 0.001$. The overall computation took 2 and 16.5 hours, respectively.

Gene Regulation of Mammalian Cell Cycle. We have applied the min-max approximation to the gene regulation model published in [25], the regulatory network is shown in Fig. 3a. The model explains regulation of a transition between early phases of the mammalian cell cycle. In particular, it targets the transition from the control G_1-phase to S-phase (the synthesis phase). G_1-phase makes an important checkpoint controlled by a *bistable regulatory circuit* based on an interplay of the retinoblastoma protein *pRB*, denoted by A (the so-called tumour suppressor, HumanCyc:HS06650) and the



(a)

Gene a interactions		Gene b interactions	
$a \to a+A$	1	$b \to b+B$	0.05
$aB \to aB+A$	1	$bB \to bB+B$	1
$A+a \leftrightarrow aA$	100;10	$A+b \leftrightarrow bA$	100;10
$B+a \leftrightarrow aB$	100;10	$B+b \leftrightarrow bB$	100;10
Protein degradation			
$A \to$	γ_A	$B \to$	γ_B

(b)

Property	# iter.	# subsp.	time[h]
(1a)	$1.2{\cdot}10^6$	153	9
(2a)	$2.0{\cdot}10^6$	69	5.5
(3a)	$2.0{\cdot}10^6$	66	4.5
(1b)	$4.0{\cdot}10^6$	159	10.5
(2b)	$4.0{\cdot}10^6$	132	8
(3b)	$4.0{\cdot}10^6$	80	5

(c)

Fig. 3. (a) Two-gene regulatory circuit controlling G_1/S transition in mammalian cell cycle. (b) Stochastic mass action model of the G_1/S regulatory circuit – a,b represent genes, aA, aB, bA, bB represent transcription factor-gene promoter complexes (c) Computation results.

retinoblastoma-binding transcription factor E_2F_1, denoted by B (a central regulator of a large set of human genes, HumanCyc:HS02261). In high concentration levels, the E_2F_1 protein activates the G_1/S transition mechanism. On the other hand, a low concentration of E_2F_1 prevents committing to S-phase.

Positive autoregulation of B causes bi-stability of its concentration depending on the parameters. Especially, of specific interest is the degradation rate of A, γ_A. In [25] it is shown that for increasing γ_A the low stable mode of B switches to the high stable mode. When mitogenic stimulation increases under conditions of active growth, rapid phosphorylation of A starts and makes the degradation of unphosphorylated A stronger (the degradation rate γ_A increases). This causes B to lock in the high stable mode implying the cell cycle commits to S-phase. Since mitogenic stimulation influences the degradation rate of A, our goal is to study the population distribution around the low and high steady state and to explore the effect of γ_A by means of the landscape function.

We have translated the original ODE model into the framework of stochastic mass action kinetics [12]. The resulting reactions are shown in Fig. 3b. Since the detailed knowledge of elementary chemical reactions occurring in the process of transcription and translation is incomplete, we use the simplified form as suggested in [10]. In the minimalistic setting, the reformulation requires addition of rate parameters describing the transcription factor–gene promoter interaction while neglecting cooperativeness of transcription factors activity. Our parametrization is based on time-scale orders known for the individual processes [27] (parameters considered in s^{-1}). Moreover, we assume the numbers of A and B are bounded by 10 molecules. Upper bounds for A and B are set with respect to behaviour of an ensemble of stochastic simulations. We consider minimal population number distinguishing the two stable modes. All other species are bounded by the initial number of DNA molecules (genes a and b) which is conserved and set to 1. The corresponding CTMC has 1078 states and 5919 transitions.

We consider three hypotheses: (1) stabilization in the low mode where $B < 3$, (2) stabilization in the high mode where $B > 5$, (3) stabilization in the high mode where $B > 7$ ((3) is more focused than (2)). All the hypotheses are expressed within time horizon 1000 seconds reflecting the time scale of gene regulation response. We employ two alternative CSL formulations to express each of the three hypothesis. According to [25], we consider the parameter space $\gamma_A \in [0.005, 0.5]$.

First, we express the property of being inside the given bound during the time interval $I = [500, 1000]$ using globally operator: (1a) $\mathsf{P}_{\sim ?}[\mathsf{G}^I\,(B < 3)]$, (2a) $\mathsf{P}_{\sim ?}[\mathsf{G}^I\,(B > 5)]$ and $(3a)$ $\mathsf{P}_{\sim ?}[\mathsf{G}^I\,(B > 7)]$. The interval starts from 500 seconds in order to bridge the

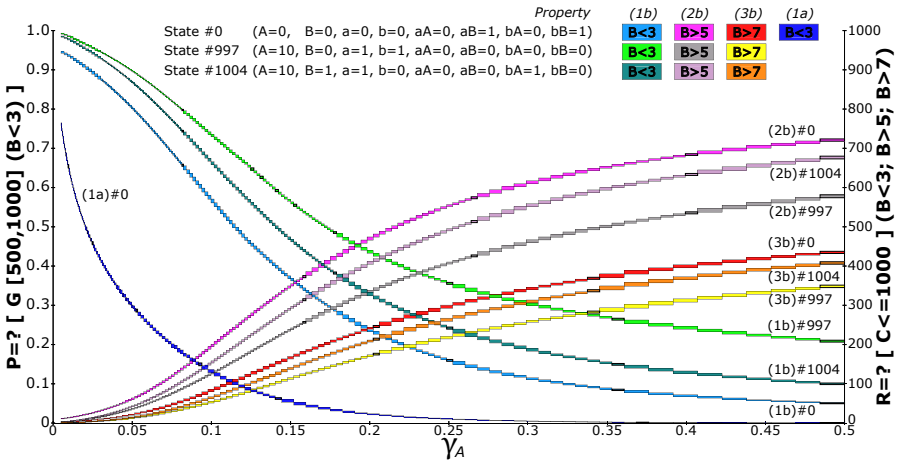


Fig. 4. Landscape functions of properties *(1a,1b,2b,3b)* for $\gamma_A \in [0.005, 0.5]$ (in s^{-1}) and initial states #0, #997 and #1004. The left Y-axis scale corresponds to *(1a)*, the right to *(1b,2b,3b)*.

initial fluctuation region and let the system stabilize. Since the stochastic noise causes molecules to repeatedly escape the requested bound, the resulting probability is significantly lower than expected. Namely, in cases (2a) and (3a) the resulting probability is close to 0 for the whole parameter space. Moreover, the selection of an initial state has only a negligible impact on the result. Therefore, in Fig. 4 only the resulting probability for case (1a) and a single selected initial state is visualized.

Second, we use a cumulative reward property to capture the fraction of the time the system has the required number of molecules within the time interval $[0, 1000]$: $(1b)$ $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B < 3)$, $(2b)$ $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B > 5)$, $(3b)$ $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B > 7)$ where $t = 1000$ and $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B \sim X)$ denotes that state reward ρ is defined such that $\forall s \in \mathbb{S}.\rho(s) = 1$ iff $B \sim X$ in s. The result is visualized for three selected initial states in Fig. 4.

Fig. 4 also illustrates inaccuracy of our approach with respect to the absolute error bound $\textsc{Err} = 0.01$ by means of small rectangles depicting approximations of the resulting probabilities and expected rewards. The analyses predict that the distribution of the low steady mode interferes with the distribution of the high steady mode. It confirms bi-stability predicted in [25] but in contrast to ODE analysis our method shows how the population of cells distributes around the two stable states. Results of computations including the number of iterations performed during parametrized uniformization, numbers of resulting subspaces and execution times in hours, are presented in Fig. 3c.

Finally, to see how degradation rates of A and B cooperate in affecting property $(3b)$, we explore two-dimensional parameter space $(\gamma_A, \gamma_B) \in [0.005, 0.1] \times [0.05, 0.1]$. The computation also required $4.0 \cdot 10^6$ iterations of the parametrized uniformization, the parameter decomposition resulted in 143 subspaces for $\textsc{Err} = 0.1$ and the overall execution took 14 hours. Fig. 5 illustrates the computed upper bound of the landscape function for initial state #0 and the absolute error. The result predicts antagonistic relation between the degradation rates which is in agreement with the ODE model [25].

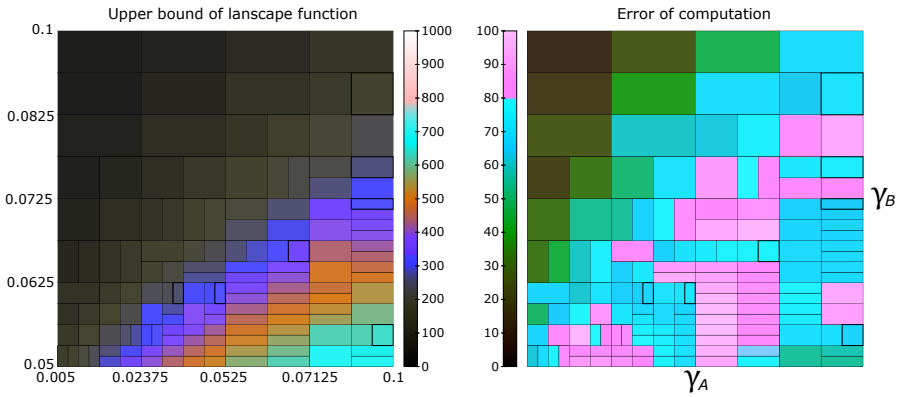


Fig. 5. Landscape function for property $(3b)$, initial state #0 $(A = 0, B = 0, a = 0, b = 0, aA = 0, aB = 1, bA = 0, bB = 1)$ and two-dimensional parameter space $(\gamma_A, \gamma_B) \in [0.005, 0.1] \times [0.05, 0.1]$ (represented in s^{-1} by X and Y axes, respectively). On the left, the upper bound of the landscape function is illustrated. On the right, the absolute error given as difference between computed upper and lower bounds is depicted. In both cases the color scale is used.

8 Conclusions

We have introduced the parameter exploration problem for stochastic biochemical systems as the computation of a landscape function for a given temporal logic formula. The key idea of our approach is to approximate the lower and upper bounds of the landscape function. To obtain such approximation for an arbitrary nested CSL formula, we compute the largest and smallest set of states satisfying the formula using parametrized uniformization. This allows to approximate the minimal and maximal transient probability with respect to the parameter space. In order to reach a required error bound of the proposed approximation, we iteratively decompose the parameter space and compute the approximation for each subspace individually. We have demonstrated our approach to the parameter exploration problem on two biologically motivated case studies.

The experiments show that our method can be extremely time demanding and thus in our future work we will focus on its acceleration. We plan to apply techniques allowing to accelerate the underlying transient analysis [9,16] and more efficient heuristics for the parameter space decomposition. Moreover, our method can be easily parallelized and thus a significant acceleration can be obtained.

References

1. Andreychenko, A., Mikeev, L., Spieler, D., Wolf, V.: Parameter Identification for Markov Models of Biochemical Reactions. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 83–98. Springer, Heidelberg (2011)
2. 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)

3. 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)
4. Ballarini, P., Forlin, M., Mazza, T., Prandi, D.: Efficient Parallel Statistical Model Checking of Biochemical Networks. In: PDMC 2009. EPTCS, vol. 14, pp. 47–61 (2009)
5. Barbuti, R., Levi, F., Milazzo, P., Scatena, G.: Probabilistic Model Checking of Biological Systems with Uncertain Kinetic Rates. Theor. Comput. Sci. 419, 2–16 (2012)
6. Bernardini, F., Biggs, C., Derrick, J., Gheorghe, M., Niranjan, M., Sanguinetti, G.: Parameter Estimation and Model Checking in a Model of Prokaryotic Autoregulation. Tech. rep., University of Sheffield (2007)
7. Daigle, B., Roh, M., Petzold, L., Niemi, J.: Accelerated Maximum Likelihood Parameter Estimation for Stochastic Biochemical Systems. BMC Bioinformatics 13(1), 68–71 (2012)
8. Degasperi, A., Gilmore, S.: Sensitivity Analysis of Stochastic Models of Bistable Biochemical Reactions. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 1–20. Springer, Heidelberg (2008)
9. Didier, F., Henzinger, T.A., Mateescu, M., Wolf, V.: Fast Adaptive Uniformization of the Chemical Master Equation. In: HIBI 2009, pp. 118–127. IEEE Computer Society (2009)
10. El Samad, H., Khammash, M., Petzold, L., Gillespie, D.: Stochastic Modelling of Gene Regulatory Networks. Int. J. of Robust and Nonlinear Control 15(15), 691–711 (2005)
11. Fox, B.L., Glynn, P.W.: Computing Poisson Probabilities. CACM 31(4), 440–445 (1988)
12. Gillespie, D.T.: Exact Stochastic Simulation of Coupled Chemical Reactions. Journal of Physical Chemistry 81(25), 2340–2381 (1977)
13. Golightly, A., Wilkinson, D.J.: Bayesian Parameter Inference for Stochastic Biochemical Network Models Using Particle Markov Chain Monte Carlo. Interface Focus 1(6), 807–820 (2011)
14. Grassmann, W.: Transient Solutions in Markovian Queueing Systems. Computers & Operations Research 4(1), 47–53 (1977)
15. Hahn, E.M., Han, T., Zhang, L.: Synthesis for PCTL in Parametric Markov Decision Processes. In: NASA Formal Methods, pp. 146–161 (2011)
16. Henzinger, T.A., Mateescu, M., Wolf, V.: Sliding Window Abstraction for Infinite Markov Chains. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 337–352. Springer, Heidelberg (2009)
17. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A Bayesian Approach to Model Checking Biological Systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
18. Koh, C.H., Palaniappan, S., Thiagarajan, P., Wong, L.: Improved Statistical Model Checking Methods for Pathway Analysis. BMC Bioinformatics 13(suppl. 17), S15 (2012)
19. Kwiatkowska, M., Norman, G., Pacheco, A.: Model Checking Expected Time and Expected Reward Formulae with Random Time Bounds. Compu. Math. Appl. 51(2), 305–316 (2006)
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)
21. 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)
22. Mikeev, L., Neuhäußer, M., Spieler, D., Wolf, V.: On-the-fly Verification and Optimization of DTA-properties for Large Markov Chains. Form. Method. Syst. Des., 1–25 (2012)
23. Reinker, S., Altman, R., Timmer, J.: Parameter Estimation in Stochastic Biochemical Reactions. IEEE Proc. Syst. Biol. 153(4), 168–178 (2006)

24. Schlögl, F.: Chemical Reaction Models for Non-Equilibrium Phase Transitions. Zeitschrift fur Physik 253, 147–161 (1972)
25. Swat, M., Kel, A., Herzel, H.: Bifurcation Analysis of the Regulatory Modules of the Mammalian G1/S transition. Bioinformatics 20(10), 1506–1511 (2004)
26. Vellela, M., Qian, H.: Stochastic Dynamics and Non-Equilibrium Thermodynamics of a Bistable Chemical System: the Schlögl Model Revisited. Journal of The Royal Society Interface 6(39), 925–940 (2009)
27. Yang, E., van Nimwegen, E., Zavolan, M., Rajewsky, N., Schroeder, M.K., Magnasco, M., Darnell, J.E.: Decay Rates of Human mRNAs: Correlation With Functional Characteristics and Sequence Attributes. Genome Research 13(8), 1863–1872 (2003)

Acta Informatica (2017) 54:589–623
DOI 10.1007/s00236-016-0265-2

ORIGINAL ARTICLE

Precise parameter synthesis for stochastic biochemical systems

Milan Češka[2,3] · Frits Dannenberg[3] · Nicola Paoletti[3] · Marta Kwiatkowska[3] · Luboš Brim[1]

Received: 16 April 2015 / Accepted: 29 February 2016 / Published online: 28 March 2016

Abstract We consider the problem of synthesising rate parameters for stochastic biochemical networks so that a given time-bounded CSL property is guaranteed to hold, or, in the case of quantitative properties, the probability of satisfying the property is maximised or minimised. Our method is based on extending CSL model checking and standard uniformisation to parametric models, in order to compute safe bounds on the satisfaction probability of the property. We develop synthesis algorithms that yield answers that are precise to within an arbitrarily small tolerance value. The algorithms combine the computation of probability bounds with the refinement and sampling of the parameter space. Our methods are precise and efficient, and improve on existing approximate techniques that employ discretisation and refinement. We evaluate the usefulness of the methods by synthesising rates for three biologically motivated case studies: infection control for a SIR epidemic model; reliability

This work has been partially supported by the ERC Advanced Grant VERIWARE, Microsoft Research PhD Scholarship (F. Dannenberg), the Czech Grant Agency grant No. GA15-11089S (L. Brim), and the IT4Innovations Excellence in Science project No. LQ1602 (M. Češka).

Nicola Paoletti
nclpltt@gmail.com

Milan Češka
m.ceska@cs.ox.ac.uk

Frits Dannenberg
f.dannenberg@cs.ox.ac.uk

Marta Kwiatkowska
m.kwiatkowska@cs.ox.ac.uk

Luboš Brim
brim@fi.muni.cz

1 Faculty of Informatics, Masaryk University, Botanická 68a, 602 00 Brno, Czech Republic

2 Faculty of Information Technology, Brno University of Technology, Božetěchova 1/2, 612 66 Brno, Czech Republic

3 Department of Computer Science, University of Oxford, Wolfson Building, Parks Road, Oxford OX1 3QD, UK

analysis of molecular computation by a DNA walker; and bistability in the gene regulation of the mammalian cell cycle.

1 Introduction

Biochemical reaction networks are a convenient formalism for modelling a multitude of biological systems, including molecular signalling pathways, logic gates built from DNA and DNA walker circuits. For low molecule counts, and assuming a well-mixed and fixed reaction volume, the prevailing approach is to model such networks using continuous-time Markov chains (CTMCs) [20]. Stochastic model checking [31], implemented in programs such as `PRISM` [32], allows the analysis of the model behaviour against temporal logic properties expressed in continuous stochastic logic (CSL) [3]. For instance, the reliability and performance of DNA walker circuits are evaluated using properties such as "what is the probability that the walker reaches the correct final anchorage within 10 min?". We envision biochemical devices that implement biosensors and medical diagnostic systems, and hence ensuring appropriate levels of reliability is important.

Stochastic model checking assumes that the model is fully specified, including reaction rate constants. However, the reaction rates can be unknown or given as estimates that typically include some measurement error. In spite of this uncertainty, one might want to still demonstrate robustness and reliability of a synthetic molecular device. Or, one might be interested in the identification of parameter values that reproduce experimentally observed behaviour. The *parameter synthesis problem*, studied for CTMCs in [13,23], assumes a formula and a model whose rates are given as functions of parameters, and aims to compute the parameter valuations that guarantee the satisfaction of the formula. Previously the parameter synthesis problem was solved for CTMCs approximately, and only for probabilistic time-bounded reachability [23]. In this paper, we address the parameter synthesis problem for stochastic biochemical reaction networks for the full time-bounded fragment of the (branching-time) logic CSL [3]. We formulate two variants: *threshold synthesis*, which inputs a CSL formula and a probability threshold and identifies the parameter valuations which meet the threshold, and *max synthesis*, where the maximum probability of satisfying the property and the maximizing set of parameter valuations are returned.

We develop efficient synthesis algorithms that yield answers with arbitrary precision. The algorithms exploit a recently published technique that computes safe approximations to the lower and upper bounds for the probability to satisfy a CSL property over a fixed parameter space [11]. Our algorithms automatically derive the satisfying parameter regions through iterative decomposition of the parameter space up to a given tolerance value. We also demonstrate a significant speed-up of the max synthesis algorithm through the use of a sampling-based heuristic. The method is demonstrated using three case studies: the SIR epidemic model [27], where we synthesize infection and recovery rates that maximize the probability of disease extinction; the DNA walker circuit [17], where we derive stepping rates that ensure a predefined level of reliability; and a gene regulation model of the mammalian cell cycle [11], where we investigate degradation rates that lead to bi-stability.

This work is an extended version of [13], where we first introduced parameter synthesis problems and algorithms for CTMCs. In this version, we provide a rigorous treatment of the method to compute safe probability bounds and extend the approach to reward operators. We also include an additional case study on the gene regulation of the mammalian cell cycle.

1.1 Structure of the paper

In Sect. 2 preliminary definitions are given. In Sect. 3, we introduce the threshold problem and the max synthesis problem. In Sect. 4, we describe the methods to bound the probability of a formula for a fixed parameter region. These methods are then used in the synthesis algorithms that are described in Sect. 5. In Sect. 6, case studies and results of synthesis experiments are discussed. Related work is discussed in Sect. 7. Concluding remarks are given in Sect. 8.

2 Background

This section introduces the main concepts relevant for model checking of (parametric) continuous-time Markov chains and stochastic modelling of biochemical reactions.

2.1 Parametric CTMCs

Before we introduce parametric CTMCs, we recall the standard definition of CTMCs and describe the uniformisation procedure that is employed for their model checking based on [31].

Definition 1 [*Continuous-time Markov chain (CTMC)*] A CTMC is a tuple $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ where:

- S is a finite set of *states*;
- $\pi_0 : S \to [0, 1]$ is the *initial state distribution* where $\sum_{s \in S} \pi_0(s) = 1$;
- $\mathbf{R} : S \times S \to \mathbb{R}_{\geq 0}$ is the *rate matrix*; and
- $L : S \to 2^{AP}$ is a *labelling function* mapping each state $s \in S$ to the set $L(s) \subseteq AP$ of atomic propositions that hold true in s.

A transition between states $s, s' \in S$ can occur only if $\mathbf{R}(s, s') > 0$ and, in that case, the probability of triggering the transition within time t is $1 - e^{-t\mathbf{R}(s,s')}$. The time spent in state s, before a transition is triggered, is exponentially distributed with *exit rate* $E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$, and when the transition occurs the probability of moving to state s' is given by $\frac{\mathbf{R}(s,s')}{E(s)}$.

A CTMC $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ can be extended with a reward structure (ρ, ι). $\rho : S \to \mathbb{R}_{\geq 0}$ is called *state reward* and defines the rate with which a reward is acquired in state $s \in S$, e.g. a reward of $t\rho(s)$ is acquired if $\mathcal{C}$ remains in state s for t time units. The function ι: $S \times S \to \mathbb{R}_{\geq 0}$ defines the *transition reward*, such that $\iota(s_i, s_j)$ describes the reward acquired each time the transition (s_i, s_j) occurs.

We now describe the computation of transient probabilities for CTMCs, based on standard uniformisation (also called Jensen's method or randomisation). Let $\mathbf{E}$ be a $S \times S$ diagonal matrix such that $\mathbf{E}(s_i, s_i) = E(s_i)$, and define the *generator matrix* by setting $\mathbf{Q} = \mathbf{R} - \mathbf{E}$. Then, the vector π_t: $S \to \mathbb{R}_{\geq 0}$ of transient probabilities at time t is given by $\pi_t = \pi_0 e^{\mathbf{Q}t}$, such that $\pi_t(s)$ is the probability of being in state s at time instant t. Below we describe the uniformisation method.

Definition 2 (*Uniformised matrix*) Let $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ be a CTMC and $\mathbf{Q}$ the associated generator matrix. Then, the *uniformised matrix* $\mathbf{P}$ of $\mathcal{C}$ is defined by $\mathbf{P} = \mathbf{I} + \frac{1}{q}\mathbf{Q}$, where $q \geq \max_s\{E(s) - \mathbf{R}(s, s)\}$ is called the *uniformisation rate*.

Definition 3 (*Path of a CTMC*) Let $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ be a CTMC. A path ω of $\mathcal{C}$ is a sequence $\omega = s_0 t_0 s_1 t_1 \ldots$, where for all i, $s_i \in S$ and $t_i \in \mathbb{R}_{\geq 0}$ is the time spent in state s_i.

ω is infinite when $\mathbf{R}(s_i, s_{i+1}) > 0$ for all i, and finite of length n when $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i < n-1$ and $E(s_{n-1}) = 0$.

The set of paths starting in state s is denoted as Path(s) and a unique probability measure, Pr, exists on Path(s) [31]. Function $\omega(i) = s_i$ maps a position i of ω to its i-th state. The state at time t in ω is denoted as $\omega@t$, and is equal to $\omega(i)$ for the smallest i such that $\sum_{n=0}^{i} t_n \geq t$.

The transient state-probabilities by time t are obtained by standard uniformisation as a sum of state distributions after i discrete-stochastic steps, weighted by the probability of observing i jumps in a Poisson process.

Definition 4 (*Transient probabilities with standard uniformisation*) Let $\mathcal{C} = (S, \pi_0, \mathbf{R}, L)$ be a CTMC. Let q and $\mathbf{P}$ be the associated uniformisation rate and uniformised matrix, respectively. The vector of transient probabilities at time t, π_t, is given by standard uniformisation as follows [21,24,38]:

$$\pi_t = \sum_{i=0}^{\infty} \gamma_{i,qt}\, \tau_i \tag{1}$$

where $\tau_i = \pi_0 \mathbf{P}^i$ is the vector of probabilities in the discretized process at the i-th step; and $\gamma_{i,qt} = e^{-qt}\frac{(qt)^i}{i!}$ denotes the i-th Poisson probability for a process with parameter qt. An approximate value is given by finite summation

$$\hat{\pi}_t = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt}\, \tau_i \tag{2}$$

when k_ϵ satisfies the convergence bound $\sum_{0}^{k_\epsilon} \gamma_{i,qt} \geq 1 - \epsilon$ for some $\epsilon > 0$. The Poisson terms and the summation bound k_ϵ are computed efficiently using an algorithm due to Fox and Glynn [19].

Parametric continuous-time Markov chains (pCTMCs) [23] extend the notion of CTMCs by allowing transition rates to depend on model parameters. We assume a set K of model parameters. The domain of each parameter $k \in K$ is given by a closed real interval describing the range of possible values, i.e, $[k^\perp, k^\top] \subseteq \mathbb{R}$. The parameter space $\mathcal{P}$ induced by K is defined as the Cartesian product of the individual intervals, $\mathcal{P} = \times_{k \in K} [k^\perp, k^\top]$, so that $\mathcal{P}$ is a hyper-rectangular space. Subsets of the parameter space $\mathcal{P}$ are referred to as *parameter regions* or *subspaces*.

Definition 5 [*Parametric CTMC (pCTMC)*] Let K be a set of parameters. A pCTMC over K is a tuple $(S, \pi_0, \mathbf{R}, L)$, where:

- S, π_0 and L are as in Definition 1; and
- $\mathbf{R}$: $S \times S \to \mathbb{R}[K]$ is the *parametric rate matrix*, where $\mathbb{R}[K]$ denotes the set of polynomials over the reals $\mathbb{R}$ with variables $k \in K$.

Given a pCTMC and a parameter space $\mathcal{P}$, we denote with $\mathcal{C}_\mathcal{P}$ the set $\{\mathcal{C}_p \mid p \in \mathcal{P}\}$ where $\mathcal{C}_p = (S, \pi, \mathbf{R}_p, L)$ is the instantiated CTMC obtained by replacing the parameters in $\mathbf{R}$ with their valuation in p. The definition restricts the rates to be polynomials, which are sufficient to describe a wide class of biological systems.

2.2 CSL for parametric CTMCs

To specify properties over pCTMCs, we employ the time-bounded fragment of *continuous stochastic logic (CSL)* [3].

Definition 6 (*Time-bounded CSL*) The syntax of time-bounded CSL consists of state formulas (Φ) and path formulas (ϕ) given as

$$\Phi ::= \mathsf{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\sim r}[\phi]$$
$$\phi ::= \mathsf{X}\,\Phi \mid \Phi\ \mathrm{U}^I\,\Phi$$

where $a \in AP$ is an atomic proposition, $\sim \in \{<, \leq, \geq, >\}$, $r \in [0, 1]$ is a probability threshold and I is an interval of $\mathbb{R}_{\geq 0}$.

$P_{\sim r}[\phi]$ holds if the probability of the path formula ϕ being satisfied from a given state meets $\sim r$. Path formulas are defined by combining state formulas through temporal operators: $\mathsf{X}\,\Phi$ is true if Φ holds in the next state, $\Phi_1\ \mathrm{U}^I\,\Phi_2$ is true if Φ_2 holds at some time point $t \in I$, and Φ_1 holds for all time points $t' < t$. The future operator, F, and globally operator, G, are derived from U as follows:

$$P_{\sim r}\left[\mathrm{F}^I\,\Phi\right] \equiv P_{\sim r}\left[\mathsf{true}\ \mathrm{U}^I\ \Phi\right]$$
$$P_{\sim r}\left[\mathrm{G}^I\,\Phi\right] \equiv P_{\tilde{\sim} 1-r}\left[\mathrm{F}^I\ \neg\Phi\right]$$

where $\tilde{<} \equiv >$, $\tilde{\leq} \equiv \geq$, $\tilde{\geq} \equiv \leq$ and $\tilde{>} \equiv <$. Informally, $\mathrm{F}^I\,\Phi$ is true if Φ holds at some time instant in the interval I, while $\mathrm{G}^I\,\Phi$ is true if Φ holds for all $t \in I$. The logic can be extended with the following time-bounded reward operators [31]:

$$\mathrm{R}_{\sim r}[\mathrm{C}^{\leq t}] \mid \mathrm{R}_{\sim r}[\mathrm{I}^{=t}] \tag{3}$$

where $t, r \in \mathbb{R}_{\geq 0}$. $\mathrm{R}_{\sim r}[\mathrm{C}^{\leq t}]$ holds if the expected reward cumulated up to time t meets the bound $\sim r$, while $\mathrm{R}_{\sim r}[\mathrm{I}^{=t}]$ holds if the expected reward at time t meets $\sim r$. We now provide the formal semantics of time-bounded CSL with rewards for parametric CTMCs. To this end, we introduce two satisfaction relations, $\models_\perp$ and $\models_\top$, to describe if a CSL property holds for all and some instantiations, respectively, of a pCTMC.

Definition 7 (*Semantics of time-bounded CSL for pCTMCs*) Let $\mathcal{C}_\mathcal{P} = (S, \pi_0, \mathbf{R}, L)$ be a pCTMC over a parameter space $\mathcal{P}$ with reward structure (ρ, ι). For each state $s \in S$ the satisfaction relations $s \models_\perp \Phi$ and $s \models_\top \Phi$ are defined inductively by:

$s \models_\top \mathsf{true}$	for all $s \in S$	$s \models_\perp \mathsf{true}$	for all $s \in S$
$s \models_\top a$	$\Leftrightarrow a \in L(s)$	$s \models_\perp a$	$\Leftrightarrow a \in L(s)$
$s \models_\top \neg\Phi$	$\Leftrightarrow s \not\models_\perp \Phi$	$s \models_\perp \neg\Phi$	$\Leftrightarrow s \not\models_\top \Phi$
$s \models_\top \Phi \wedge \Psi$	$\Leftrightarrow s \models_\top \Phi \wedge s \models_\top \Psi$	$s \models_\perp \Phi \wedge \Psi$	$\Leftrightarrow s \models_\perp \Phi \wedge s \models_\perp \Psi$

$$s \models_\top P_{\sim r}[\phi] \Leftrightarrow \exists p \in \mathcal{P}.\ Pr(\omega \in \mathrm{Path}(s) \mid \omega \models \phi) \sim r \text{ in } \mathcal{C}_p$$
$$s \models_\perp P_{\sim r}[\phi] \Leftrightarrow \forall p \in \mathcal{P}.\ Pr(\omega \in \mathrm{Path}(s) \mid \omega \models \phi) \sim r \text{ in } \mathcal{C}_p$$
$$s \models_\top \mathrm{R}_{\sim r}[\mathrm{C}^{\leq t}] \Leftrightarrow \exists p \in \mathcal{P}.\ Exp(s, X_{\mathrm{C}^{\leq t}}) \sim r \text{ in } \mathcal{C}_p$$
$$s \models_\perp \mathrm{R}_{\sim r}[\mathrm{C}^{\leq t}] \Leftrightarrow \forall p \in \mathcal{P}.\ Exp(s, X_{\mathrm{C}^{\leq t}}) \sim r \text{ in } \mathcal{C}_p$$
$$s \models_\top \mathrm{R}_{\sim r}[\mathrm{I}^{=t}] \Leftrightarrow \exists p \in \mathcal{P}.\ Exp(s, X_{\mathrm{I}^{=t}}) \sim r \text{ in } \mathcal{C}_p$$
$$s \models_\perp \mathrm{R}_{\sim r}[\mathrm{I}^{=t}] \Leftrightarrow \forall p \in \mathcal{P}.\ Exp(s, X_{\mathrm{I}^{=t}}) \sim r \text{ in } \mathcal{C}_p$$

where the path formula ϕ is expanded as

$$\omega \vDash \mathrm{X}\Phi \quad \Leftrightarrow \omega(1) \text{ exists and } \omega(1) \vDash \Phi$$
$$\omega \vDash \Phi_1 \mathrm{U}^I\ \Phi_2 \quad \Leftrightarrow \exists t \in I. \text{ such that } [\omega@t \vDash \Phi_2 \wedge (\forall r \in [0,t).\ \omega(r) \vDash \Phi_1)]$$

and $Exp(s, X)$ for $X \in \{X_{\mathrm{C}^{\leq t}}, X_{\mathrm{I}^{=t}}\}$ denotes the expectation of the random variable X with respect to the probability measure Pr over paths starting in s, defined for any $\omega = s_0 t_0 s_1 t_1 \ldots \in \mathrm{Path}(s)$ by

$$X_{\mathrm{C}^{\leq t}} = \sum_{i=0}^{j_t-1} (t_i \cdot \rho(s_i) + \iota(s_i, s_{i+1})) + \left(t - \sum_{i=0}^{j_t-1} t_i\right) \cdot \rho(s_{j_t})$$
$$X_{\mathrm{I}^{=t}} = \rho(\omega@t)$$

where $j_t = \min\{j \mid \sum_{i=0}^{j} t_i \leq t\}$.

Note that, for formula $P_{\sim r}[\phi]$, $\vDash_\perp$ and $\vDash_\top$ are defined by quantifying over $p \in \mathcal{P}$ and evaluating the satisfaction probability of ϕ on the instantiation $\mathcal{C}_p$. This probability can thus be obtained using regular CSL satisfaction relation $\vDash$. Therefore, relations $\vDash_\perp$ and $\vDash_\top$ reduce to $\vDash$ when the parameter space contains only a single valuation, i.e. $\mathcal{P} = \{p\}$.

We further define the minimal satisfaction set $\mathrm{Sat}_\perp(\Phi)$ and the maximal satisfaction set $\mathrm{Sat}_\top(\Phi)$ as follows:

$$\mathrm{Sat}_\perp(\Phi) = \{s \in S \mid s \vDash_\perp \Phi\} \text{ and } \mathrm{Sat}_\top(\Phi) = \{s \in S \mid s \vDash_\top \Phi\}. \tag{4}$$

We now describe the *satisfaction function* to capture how the satisfaction probability of a given property relates the parameters and the initial state. For simplicity we define the function and further describe parameter synthesis only for the P operator: the method also allows a definition based on reward operators, which we describe in Sect. 4.3.

Definition 8 (*Satisfaction function*) Let ϕ be a CSL path formula, $\mathcal{C}_\mathcal{P}$ be a pCTMC over a space $\mathcal{P}$ and $s \in S$. We denote with $\Lambda_\phi : \mathcal{P} \to S \to [0, 1]$ the satisfaction function such that $\Lambda_\phi(p)(s) = Pr(\omega \in \mathrm{Path}(s) \mid \omega \vDash \phi)$ in $\mathcal{C}_p$.

Since ϕ allows nested probabilistic operators, the satisfaction function is, in general, not continuous.

2.3 Stochastic models of biochemical reaction networks

Biochemical reaction networks provide a convenient formalism for describing various biological processes as a system of well-mixed reactive species in a volume of fixed size. A CTMC semantics can be derived where states describe the number of molecules of each species, and transitions correspond to reactions that consume and produce molecules. The rate matrix is defined as

$$\mathbf{R}(s_i, s_j) = \sum_{r \in \mathsf{reac}(s_i, s_j)} f_r(s_i) \tag{5}$$

where $\mathsf{reac}(s_i, s_j)$ denotes all the reactions changing state s_i into s_j and f_r is the rate function of reaction r. Recalling that the rates of a pCTMC are polynomials over the parameters, f_r can be used to describe, among others, *mass-action kinetics* [20], according to which the rate of a reaction is proportional to the concentrations of its reactants. For instance, a bimolecular

chemical reaction of the form r: $A + B \rightarrow \cdots$ has rate $f_r(s_i) = k_r \frac{A(s_i)}{V} \frac{B(s_i)}{V}$, where $A(s_i)$, $B(s_i)$ are the numbers of molecules for species A, B in state s_i, k_r is the rate constant of reaction r and V is the size of the reaction volume.

3 Problem definition

We consider the problem of synthesizing parameters for pCTMC models of biochemical reaction networks, so that a given specification, expressed in time-bounded CSL, is satisfied. We allow models that are parametric in the rate constants and in the initial state. In contrast to previous approaches that support only specific kinds of properties (e.g. reachability as in [23]), we support the full time-bounded fragment of CSL with rewards, thus enabling generic and more expressive synthesis requirements.

We introduce two parameter synthesis problems: the *threshold synthesis* problem that, given a threshold $\sim r$ and a CSL path formula ϕ, aims to find the parameter region where the probability of ϕ meets $\sim r$; and the *max synthesis* problem that asks for the parameter region where the probability of the input formula attains its maximum, together with an interval bounding that maximum. In the latter case, all the synthesised parameters yield probabilities within this interval, but not all of them are maximising. On the other hand, solutions to the threshold synthesis problem admit parameter points left undecided. Our approach supports precise solutions through an input tolerance that limits the volume of the undecided region for the threshold synthesis problem. For max synthesis, the tolerance determines the precision of the probability interval and in turn, of the returned region. To the best of our knowledge, no other parameter synthesis methods for CTMCs exist that provide guaranteed error bounds. In the remainder of the paper, we omit the min synthesis problem that is defined and solved in a symmetric way to the max case. In addition, we assume there is a single initial state s_0, i.e. $\forall s \in S\ \pi_0[s] = 1$ if $s = s_0$, and 0 otherwise.

Problem 1 (*Threshold synthesis*) Let $\mathcal{C}_{\mathcal{P}}$ be a pCTMC over a parameter space $\mathcal{P}$, s_0 an initial state, ϕ a CSL path formula, $\sim r$ a threshold where $r \in [0, 1]$, $\sim \in \{\leq, <, >, \geq\}$ and $\varepsilon > 0$ be a volume tolerance. The *threshold synthesis* problem is finding a partition $\{\mathcal{T}, \mathcal{U}, \mathcal{F}\}$ of $\mathcal{P}$, such that:

1. $\forall p \in \mathcal{T}.\ \Lambda_\phi(p)(s_0) \sim r$; and
2. $\forall p \in \mathcal{F}.\ \Lambda_\phi(p)(s_0) \nsim r$; and
3. $\text{vol}(\mathcal{U})/\text{vol}(\mathcal{P}) \leq \varepsilon$

where $\text{vol}(A) = \int_A 1 d\mu$ is the volume of A.

Observe that a Boolean combination of state formulas results in a partition of the parameter space in a natural fashion, by following a three-valued logic interpretation. For example, consider the state formula $\Phi = P_{\sim r_1}[\phi_1] \wedge P_{\sim r_2}[\phi_2]$. Let $\{\mathcal{T}_1, \mathcal{U}_1, \mathcal{F}_1\}$ and $\{\mathcal{T}_2, \mathcal{U}_2, \mathcal{F}_2\}$ be a partition of $\mathcal{P}$ that satisfies the threshold synthesis problem for ϕ_1 and ϕ_2, respectively, and $\varepsilon > 0$ be a tolerance value. The partition $\{\mathcal{T}, \mathcal{U}, \mathcal{F}\}$ of $\mathcal{P}$ for Φ is given as follows:

$$\mathcal{F} = \mathcal{F}_1 \cup \mathcal{F}_2, \quad \mathcal{T} = \mathcal{T}_1 \cap \mathcal{T}_2, \quad \mathcal{U} = \mathcal{P} \backslash (\mathcal{F} \cup \mathcal{T}) \tag{6}$$

The new partition satisfies $\text{vol}(\mathcal{U})/\text{vol}(\mathcal{P}) < 2\varepsilon$.

Problem 2 (*Max synthesis*) Let $\mathcal{C}_{\mathcal{P}}$ be a pCTMC over a parameter space $\mathcal{P}$, s_0 an initial state, ϕ a CSL path formula, and $\epsilon > 0$ a probability tolerance. The *max synthesis* problem is finding a partition $\{\mathcal{T}, \mathcal{F}\}$ of $\mathcal{P}$ and probability bounds $\Lambda_\phi^\perp$, $\Lambda_\phi^\top$ such that:

1. $\Lambda_\phi^\perp - \Lambda_\phi^\top \leq \epsilon$;
2. $\forall p \in \mathcal{T}.\ \Lambda_\phi^\perp \leq \Lambda_\phi(p)(s_0) \leq \Lambda_\phi^\top$; and
3. $\exists p \in \mathcal{T}.\ \forall p' \in \mathcal{F}.\ \Lambda_\phi(p)(s_0) > \Lambda_\phi(p')(s_0)$.

The above formulation implies two important properties of the set $\mathcal{T}$: (i) $\mathcal{T}$ contains all the maximising parameters and (ii) all the parameters in $\mathcal{T}$ are ϵ-optimal, i.e. $\forall p \in \mathcal{T}.\ |\Lambda_\phi(p)(s_0) - \Lambda^*| \leq \epsilon$, where Λ^* is the optimal value of the satisfaction function. Note that some parameters in $\mathcal{F}$ can be also ϵ-optimal, but the third condition ensures they are not maximising.

Example 1 Figure 1 illustrates a simple birth-death process with an uncertain parameter k_1 representing the birth rate. It depicts the corresponding pCTMC and the satisfaction function Λ for a reachability property. Figure 2 illustrates the results of threshold synthesis (left) and max synthesis (right) for this model.

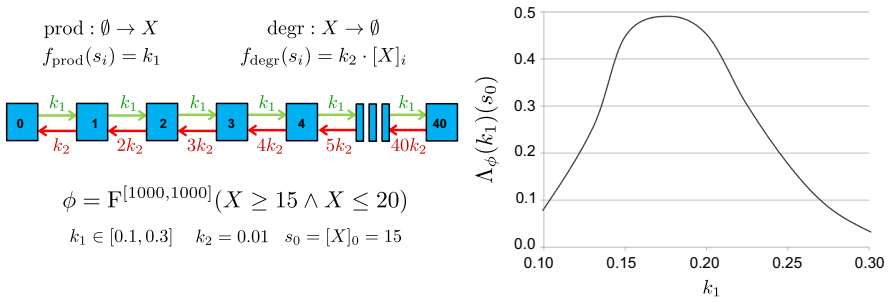


Fig. 1 *Left* the example model contains one species X (bounded by 40) and two reactions: production of X ($\emptyset \to X$ with parametric rate k_1) and degradation of X ($X \to \emptyset$ with rate $k_2 \cdot [X]$ and $k_2 = 0.01$). $[X]_i$ denotes the number of X molecules in state s_i. The initial state s_0 is given by the population $X = 15$. The corresponding pCTMC has 41 states. Property ϕ indicates that the population of X is between 15 and 20 at time 1000. The parameter space $\mathcal{P}$ is given by the interval of the stochastic rate constant $k_1 \in [0.1, 0.3]$. *Right* the satisfaction function Λ_ϕ

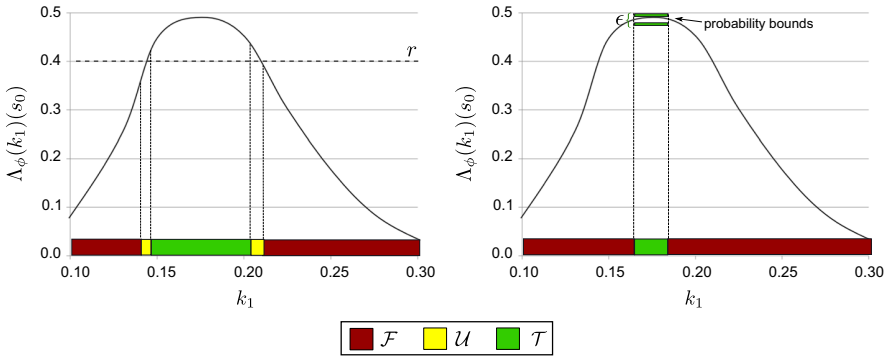


Fig. 2 Synthesis for the birth-death process of Fig. 1. *Left* threshold synthesis for $P_{\geq 0.4}[\phi]$ and with volume tolerance $\varepsilon = 5\ \%$. *Right* max-synthesis with probability tolerance $\epsilon = 2\ \%$

4 Computing lower and upper probability bounds

This section presents a generalization of the parameter exploration procedure originally introduced in [11]. The procedure takes a pCTMC $\mathcal{C}_{\mathcal{P}}$ and CSL path formula ϕ, and provides safe under- and over-approximations for the minimal and maximal probability that $\mathcal{C}_{\mathcal{P}}$ satisfies ϕ, that is, lower and upper bounds satisfying, for all $s \in S$,

$$\begin{aligned} \Lambda_{\phi,\min}(s) &\leq \inf_{p\in\mathcal{P}} \Lambda_{\phi}(p)(s) \quad \text{and} \\ \Lambda_{\phi,\max}(s) &\geq \sup_{p\in\mathcal{P}} \Lambda_{\phi}(p)(s). \end{aligned} \tag{7}$$

The accuracy of these approximations is improved by partitioning the parameter space $\mathcal{P}$ into subspaces and re-computing the corresponding bounds, which forms the basis of the synthesis algorithms that we discuss in the next section. We first show how to compute bounds $\Lambda_{\phi,\min}(s)$, $\Lambda_{\phi,\max}(s)$ for unnested path formulas. Then, we extend the method to nested path formulas, by providing under- and over-approximations of the satisfaction sets $\mathrm{Sat}_{\perp}$ and $\mathrm{Sat}_{\top}$ (see Eq. 4), and to reward operators. Finally, we analyse the accuracy and consistency of the method, and show that in case of nested properties, the satisfaction function is characterized as a piecewise polynomial function.

4.1 Computing bounds for unnested path formulas

Regular time-bounded CSL model checking for an unnested path formula ϕ reduces to the computation of transient probabilities [4]. A similar reduction is also applicable to the computation of lower and upper bounds $\Lambda_{\phi,\min}$ and $\Lambda_{\phi,\max}$. In the following, we extend standard uniformisation to obtain safe bounds for a class of parametric rate functions.

Definition 9 (*Parametric transient probabilities*) Let $\mathcal{C}_{\mathcal{P}} = (S, \pi_0, \mathbf{R}, L)$ be a pCTMC over a parameter space $\mathcal{P}$. The vector of transient probabilities at time t and for parameter valuation $p \in \mathcal{P}$ is approximated as follows

$$\hat{\pi}_{t,p} = \pi_0 \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \mathbf{P}_p^i = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \tau_{i,p} \tag{8}$$

where π_0, $\gamma_{i,qt}$ and k_ϵ are as in Definition 4, $\tau_{i,p} = \pi_0 \mathbf{P}_p^i$ is the probability evolution in the discretized process, and $\mathbf{P}_p$ is the uniformised matrix obtained from $\mathbf{R}_p$.

We now show how to obtain safe approximations, $\hat{\pi}_t^{\min}$ and $\hat{\pi}_t^{\max}$, of $\hat{\pi}_{t,p}$, such that for all $s \in S$:

$$\begin{aligned} \hat{\pi}_t^{\min}(s) &\leq \min_{p\in\mathcal{P}} \hat{\pi}_{t,p}(s) \quad \text{and} \\ \hat{\pi}_t^{\max}(s) &\geq \max_{p\in\mathcal{P}} \hat{\pi}_{t,p}(s). \end{aligned} \tag{9}$$

The function $\hat{\pi}_t(s)$, which maps each parameter p to $\hat{\pi}_{t,p}(s)$, is a polynomial of degree $k_\epsilon d$, where d is the maximum degree of the elements of the parametric rate matrix $\mathbf{R}$. Thus, bounding the polynomial expression of $\hat{\pi}_t(s)$ is infeasible due to the large number of uniformisation steps, k_ϵ, and previous approaches have provided only an approximate solution by sampling the value of $\hat{\pi}_t$ over a grid in $\mathcal{P}$ [23].

We overcome this problem through a stepwise and statewise approximation. Specifically, for each uniformisation step i, we derive bounds $\tau_i^{\min}$ and $\tau_i^{\max}$, such that for all $s \in S$:

$$\tau_i^{\min}(s) \leq \min_{p \in \mathcal{P}} \tau_{i,p}(s) \quad \text{and} \quad \tau_i^{\max}(s) \geq \max_{p \in \mathcal{P}} \tau_{i,p}(s). \tag{10}$$

This allows robust approximations to the transient probabilities given by

$$\hat{\pi}_t^{\min} = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt}\, \tau_i^{\min} \quad \text{and} \tag{11}$$

$$\hat{\pi}_t^{\max} = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt}\, \tau_i^{\max} \tag{12}$$

which satisfy Eq. 9. For fixed $p \in \mathcal{P}$ and step i, the vector $\tau_{i,p}$ is given by

$$\tau_{i,p}(s) = \begin{cases} \tau_{i-1,p}(s) + \frac{1}{q} \cdot \text{flux}\left(\tau_{i-1,p}, s\right)(p) & \text{if } i > 0 \\ \pi_0(s) & \text{if } i = 0 \end{cases} \tag{13}$$

where q is the uniformisation constant and $\text{flux}(\tau, s)(p)$ is the net probability inflow of s in one step, starting from distribution τ. This is defined as:

$$\text{flux}(\tau, s)(p) = \sum_{s' \in S} \mathbf{R}_p(s', s) \cdot \tau(s') - \sum_{s' \in S} \mathbf{R}_p(s, s') \cdot \tau(s). \tag{14}$$

In the stepwise approximation, $\tau_i^{\min}$ and $\tau_i^{\max}$ are computed from $\tau_{i-1}^{\min}$ and $\tau_{i-1}^{\max}$, respectively, in such a way that:

$$\tau_i^{\min}(s) \leq \tau_{i-1}^{\min}(s) + \frac{1}{q} \cdot \min_{p \in \mathcal{P}} \text{flux}\left(\tau_{i-1}^{\min}, s\right)(p) \quad \text{and} \tag{15}$$

$$\tau_i^{\max}(s) \geq \tau_{i-1}^{\max}(s) + \frac{1}{q} \cdot \max_{p \in \mathcal{P}} \text{flux}\left(\tau_{i-1}^{\max}, s\right)(p). \tag{16}$$

The above inequalities imply Eq. 10, since they establish coarser under- and over-approximations where the parameter valuation p is optimised locally, i.e. at each step and at each state.

It can be shown that the computation of $\tau_i^{\min}(s)$ and $\tau_i^{\max}(s)$ reduces to bounding the range of a polynomial of degree d over the parameters, where d is the maximum degree in $\mathbf{R}$. Henceforth, we restrict the class of allowed rate functions in order to compute $\tau_i^{\min}(s)$ and $\tau_i^{\max}(s)$. Specifically, we consider models where the entries of $\mathbf{R}$ are *multi-affine polynomials*, i.e. multivariate polynomials where each variable has degree at most 1. We remark that this class of models includes biochemical reaction networks with mass-action kinetics. Due to the following proposition, we can optimise the flux terms in a precise and efficient way, thus providing an effective method to compute $\tau_i^{\min}$ and $\tau_i^{\max}$.

Proposition 1 *Let $\mathcal{C}_\mathcal{R}$ be a pCTMC over a rectangular space $\mathcal{R}$, with state space S and parametric rate matrix $\mathbf{R}$. If the entries of $\mathbf{R}$ are multi-affine functions, then for any vector $\tau : S \to [0, 1]$ and state $s \in S$,*

$$\min_{p \in \mathcal{R}} \mathit{flux}(\tau, s)(p) = \min_{p \in V_\mathcal{R}} \mathit{flux}(\tau, s)(p) \quad \mathit{and} \quad \max_{p \in \mathcal{R}} \mathit{flux}(\tau, s)(p) = \max_{p \in V_\mathcal{R}} \mathit{flux}(\tau, s)(p) \tag{17}$$

where $V_\mathcal{R}$ is the set of vertices of $\mathcal{R}$ and flux *is as in Eq. 14.*

Proof The expression $\text{flux}(\tau, s)$ is a linear combination of the entries of $\mathbf{R}$, and thus is, in turn, a multi-affine function. By [7,40], the extrema of a multi-affine function defined over a rectangular domain $\mathcal{R}$ are found in the vertices of $\mathcal{R}$. □

Therefore, the bounds are computed as

$$\tau_i^{\min}(s) = \tau_{i-1}^{\min}(s) + \frac{1}{q} \cdot \min_{p \in V_{\mathcal{R}}} \text{flux}\left(\tau_{i-1}^{\min}, s\right)(p) \quad \text{and} \tag{18}$$

$$\tau_i^{\max}(s) = \tau_{i-1}^{\max}(s) + \frac{1}{q} \cdot \max_{p \in V_{\mathcal{R}}} \text{flux}\left(\tau_{i-1}^{\max}, s\right)(p) \tag{19}$$

which requires evaluating the flux terms only at the corner points of $\mathcal{R}$.

The above derivation describes forward computation of the probability bounds, i.e. the computation starts with an initial distribution at time 0 and the probability mass is propagated forward in time. Then, the bounds on the satisfaction function $\Lambda_{\phi,\min}(s)$ and $\Lambda_{\phi,\max}(s)$ are computed from $\hat{\pi}_t^{\min}$ and $\hat{\pi}_t^{\max}$, respectively, by setting $\pi_0(s) = 1$. When model checking a CSL formula, the computation of transient probabilities actually proceeds backwards [31]. For a target set $A \subseteq S$, parametric backward analysis computes a series of vectors $\sigma_i^{\min}$ and $\sigma_i^{\max}$ such that for all $s \in S$:

$$\sigma_i^{\min}(s) \leq \min_{p \in \mathcal{P}} \sigma_{i,p}(s) \quad \text{and} \quad \sigma_i^{\max}(s) \geq \max_{p \in \mathcal{P}} \sigma_{i,p}(s) \tag{20}$$

where $\sigma_{i,p}(s)$ is the probability that, starting from the state s, a state in A is reached after i steps in the discretised process corresponding to $\mathcal{C}_p$. The computation of $\sigma_i^{\min}$ and $\sigma_i^{\max}$ exploits Proposition 1 and is analogous to that of the forward method. In this way, the vectors $\Lambda_{\phi,\min}$, $\Lambda_{\phi,\max}$ are obtained as:

$$\Lambda_{\phi,\min}(s) = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \sigma_i^{\min}(s) \tag{21}$$

$$\Lambda_{\phi,\max}(s) = \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \sigma_i^{\max}(s) + e_{\text{f-g}} \tag{22}$$

for all $s \in S$, where the $e_{\text{f-g}}$ error is due to the truncation of the infinite summation in the discretised process, and can be controlled using the Fox and Glynn algorithm [19]. The set of target states A and time-horizon considered in the uniformisation procedure depend on the CSL formula. Note that the uniformised matrix $\mathbf{P}_p$ is modified according to the formula in a similar way to standard non-parametric CSL model checking [31].

4.2 Computing bounds for nested path formulas

To obtain $\Lambda_{\phi,\min}$ and $\Lambda_{\phi,\max}$ for an arbitrary path formula ϕ that contains nested state formulas, we have to correctly approximate the sets $\text{Sat}_\perp(\Phi)$ and $\text{Sat}_\top(\Phi)$ for each subformula $\Phi = P_{\sim r}[\phi]$. The approximated sets, denoted as $\overline{\text{Sat}_\perp}(\Phi)$ and $\overline{\text{Sat}_\top}(\Phi)$ are defined as

$$\overline{\text{Sat}_\perp}(\Phi) = \left\{s \in S \mid s \,\overline{\models_\perp}\, \Phi\right\} \quad \text{and} \quad \overline{\text{Sat}_\top}(\Phi) = \left\{s \in S \mid s \,\overline{\models_\top}\, \Phi\right\} \tag{23}$$

where $\overline{\models_\perp}$ and $\overline{\models_\top}$ approximate the satisfaction relations $\models_\perp$ and $\models_\top$ (see Definition 7), respectively. For a pCTMC $\mathcal{C}_\mathcal{P}$, their semantics is defined as follows:

$$s \,\overline{\models_\top}\, P_{\sim r}[\phi] \Leftrightarrow \exists p \in \mathcal{P}.\ Pr\left(\omega \in \text{Path}(s) \mid \omega \,\overline{\models_\top}\, \phi\right) \sim r \text{ in } \mathcal{C}_\mathcal{P}$$

$$s \,\overline{\models_\perp}\, P_{\sim r}[\phi] \Leftrightarrow \forall p \in \mathcal{P}.\ Pr\left(\omega \in \text{Path}(s) \mid \omega \,\overline{\models_\perp}\, \phi\right) \sim r \text{ in } \mathcal{C}_\mathcal{P}$$

where the path formula ϕ is expanded for $+ \in \{\top, \bot\}$ as

$$\begin{aligned} \omega \overline{\models_+} \mathrm{X}\Phi \quad &\Leftrightarrow \omega(1) \text{ exists and } \omega(1) \overline{\models_+} \Phi \\ \omega \overline{\models_+} \Phi_1 \mathrm{U}^I \Phi_2 \quad &\Leftrightarrow \exists t \in I. \text{ such that } \left[\omega@t \overline{\models_+} \Phi_2 \wedge \left(\forall r \in [0,t).\, \omega(r) \overline{\models_+} \Phi_1\right)\right]. \end{aligned}$$

The semantics of the other state formulas is the same as in Definition 7.

The CSL model checking for pCTMCs proceeds through a bottom-up procedure that computes the sets $\overline{\mathrm{Sat}_\bot}(\Phi)$ and $\overline{\mathrm{Sat}_\top}(\Phi)$ by iteratively replacing the innermost $P_{\sim r}[\phi]$ operators with the corresponding sets of satisfying states. When ϕ is non-nested, these sets are obtained from the safe bounds of the corresponding satisfaction function Λ_ϕ (computed as per Sect. 4.1) as follows:

$$s \overline{\models_\bot} P_{\sim r}[\phi] \Leftrightarrow \begin{cases} \Lambda_{\phi,\min}(s) \sim r & \text{if } \sim \in \{\geq, >\} \\ \Lambda_{\phi,\max}(s) \sim r & \text{if } \sim \in \{\leq, <\} \end{cases} \tag{24}$$

$$s \overline{\models_\top} P_{\sim r}[\phi] \Leftrightarrow \begin{cases} \Lambda_{\phi,\max}(s) \sim r & \text{if } \sim \in \{\geq, >\} \\ \Lambda_{\phi,\min}(s) \sim r & \text{if } \sim \in \{\leq, <\}. \end{cases} \tag{25}$$

The approximations $\overline{\models_\bot}$ and $\overline{\models_\top}$ propagate the bounds on the satisfaction function inductively on the structure of the CSL formula, such that:

$$\overline{\mathrm{Sat}_\bot}(\Phi) \subseteq \mathrm{Sat}_\bot(\Phi) \quad \text{and} \quad \overline{\mathrm{Sat}_\top}(\Phi) \supseteq \mathrm{Sat}_\top(\Phi). \tag{26}$$

Correctness follows from the expansion of the satisfaction relations, which we demonstrate for the $P_{\geq r}$ operator only. The left-hand side of Eq. 26 follows for $\Phi = P_{\geq r}[\phi]$ through:

$$\begin{aligned} s \overline{\models_\bot} P_{\geq r}[\phi] &\Rightarrow \Lambda_{\phi,\min}(s) \geq r \Rightarrow \forall p \in \mathcal{P}.\, \Lambda_\phi(p)(s) \geq r \\ &\Rightarrow \forall p \in \mathcal{P}.\, Pr\left(\omega \in \mathrm{Path}(s) \text{ in } \mathcal{C}_p \mid \omega \models \phi\right) \geq r \Rightarrow s \models_\bot P_{\geq r}[\phi]. \end{aligned}$$

The right-hand side follows from:

$$s \models_\top P_{\geq r}[\phi] \Rightarrow \exists p \in \mathcal{P}.\, \Lambda_\phi(p)(s) \geq r \Rightarrow \Lambda_{\phi,\max}(s) \geq r \Rightarrow s \overline{\models_\top} P_{\geq r}[\phi].$$

An example of synthesis for nested formulas is illustrated in Sect. 6.1.1.

4.2.1 Complexity

For $\mathcal{C}_\mathcal{R} = (S, \pi_0, \mathbf{R}, L)$, time-bounded path formula ϕ and fixed n-dimensional rectangular space $\mathcal{R}$, the time complexity of the procedure for computing the probability bounds is $\mathcal{O}(t_{CSL} \cdot t_{pCSL})$. The factor $t_{CSL} = |\phi| \cdot M \cdot q \cdot t_{\max}$ is the worst-case time complexity of time-bounded CSL model checking (see [4]), where $|\phi|$ is the number of time-bounded path sub-formulas in ϕ, M is the number of non-zero elements in the rate matrix, $t_{\max}$ is the highest time bound occurring in ϕ and q is the uniformisation rate. The factor t_{pCSL} is due to the parametric analysis. Following Proposition 1 and Eq. 18, for general multi-affine rate functions the bounds $\tau_i^{\min}$ and $\tau_i^{\max}$ are obtained by performing $2 \cdot 2^n$ evaluations of the vector $\tau_{i,p}$ (there are 2^n corner points in $\mathcal{R}$), at each uniformisation step i. Thus, $t_{pCSL} = 2^{n+1}$. On the other hand, for linear rate functions $t_{pCSL} = \mathcal{O}(n)$, as shown in [11].

4.3 Computing bounds for reward operators

The standard model checking algorithm for reward operators is based on the uniformisation procedure [31]. To obtain the sets $\overline{\mathrm{Sat}_\top}(\Phi)$ and $\overline{\mathrm{Sat}_\bot}(\Phi)$ for the reward operators, we have

to compute for $X \in \{X_{C^{\leq t}}, X_{I^{=t}}\}$ bounds $Exp_{\min}(s, X)$ and $Exp_{\max}(s, X)$ on the expected rewards such that:

$$Exp_{\min}(s, X) \leq \inf_{p \in \mathcal{P}} Exp(s, X) \text{ in } \mathcal{C}_p \tag{27}$$

$$Exp_{\max}(s, X) \geq \sup_{p \in \mathcal{P}} Exp(s, X) \text{ in } \mathcal{C}_p. \tag{28}$$

The quantities can be obtained using the forward computation where the initial distribution is defined as $\pi_0(s) = 1$. For a reward structure $(\rho, \boldsymbol{\iota})$, the instantaneous reward is computed as:

$$Exp(s, X_{I^{=t}}) = \sum_{s' \in S} \rho(s')\pi_t(s'). \tag{29}$$

To find the cumulative reward, the state-transition rewards $\boldsymbol{\iota}$ are additionally taken into account as follows [30]:

$$Exp(s, X_{C^{\leq t}}) = \sum_{s' \in S} \int_0^t \left(\rho(s')\pi_u(s') + \sum_{s'' \in S} \mathbf{R}(s', s'')\boldsymbol{\iota}(s', s'')\pi_u(s') \right) du \tag{30}$$

$$= \sum_{s' \in S} \int_0^t \left(\rho(s') + \sum_{s'' \in S} \mathbf{R}(s', s'')\boldsymbol{\iota}(s', s'') \right) \pi_u(s') du \tag{31}$$

$$= \sum_{s' \in S} \left(\rho(s') + \sum_{s'' \in S} \mathbf{R}(s', s'')\boldsymbol{\iota}(s', s'') \right) \int_0^t \pi_u(s') du. \tag{32}$$

where $\int_0^t \pi_u(s')du$ is the expected amount of time the Markov process spends in state s' up until time t. Following the parametric uniformisation of Sect. 4.1, safe bounds for the reward operators are found as:

$$Exp_{\min}(s, X_{I^{=t}}) = \sum_{s' \in S} \rho(s')\hat{\pi}_t^{\min}(s') \tag{33}$$

$$Exp_{\max}(s, X_{I^{=t}}) = \sum_{s' \in S} \rho(s') \left(\hat{\pi}_t^{\max}(s') + e_{\text{f-g}} \right) \tag{34}$$

$$Exp_{\min}(s, X_{C^{\leq t}}) = \sum_{s' \in S} \left(\text{rew}(\rho, \boldsymbol{\iota}, s') \frac{1}{q} \sum_{i=0}^{\infty} \overline{\gamma}_{i,qt} \cdot \tau_i^{\min}(s') \right) \tag{35}$$

$$Exp_{\max}(s, X_{C^{\leq t}}) = \sum_{s' \in S} \left(\text{rew}(\rho, \boldsymbol{\iota}, s') \frac{1}{q} \sum_{i=0}^{\infty} \overline{\gamma}_{i,qt} \cdot \tau_i^{\max}(s') \right) \tag{36}$$

where the mixed Poisson probabilities and the combined rewards are

$$\overline{\gamma}_{i,qt} = 1 - \sum_{j=i}^{\infty} \gamma_{j,qt} \tag{37}$$

$$\text{rew}(\rho, \iota, s) = \rho(s) + \sum_{s' \in S} \mathbf{R}(s, s')\boldsymbol{\iota}(s, s'). \tag{38}$$

The bounds for the cumulative rewards (Eqs. 35 and 36) are understood as follows: $\overline{\gamma}_{i,qt}$ is the probability to see at least i jumps in the discretised process, which is multiplied by the under- or over-approximation of the probability to be in state s. So $\sum_{i=0}^{\infty} \overline{\gamma}_{i,qt} \cdot \tau_i^{\min}(s)$ is an under-approximation of the number of epochs the discretised process spends in state s.

Observe that $\frac{1}{q}$ is the expected time until a jump occurs and $\mathrm{rew}(\rho, \iota, s)$ is the expected reward obtained per time unit spent in s. As discussed in [31], the infinite sums can be approximated using methods based on Fox and Glynn [19].

Note that, also for rewards, backward computation allows obtaining safe bounds for all states $s \in S$, using the vectors $\sigma_i^{\min}$ and $\sigma_i^{\max}$.

4.4 Analysis of satisfaction function and approximation error

When computing bounds $\tau_i^{\min}$ and $\tau_i^{\max}$ on the transient probabilities, an approximation error occurs because the values are obtained by optimizing $\tau_{i,p}$ locally, i.e. at each step and at each state, and this error accumulates at each uniformisation step. We examine this error for the multi-affine case where Proposition 1 applies. For a fixed state s, let the maximizing argument of the transient probability be (cf. Eq. 8):

$$p^* = \arg\max_{p \in \mathcal{R}} \pi_t(s) \tag{39}$$

Then, the optimal probabilities at step i, τ_i^*, are defined by

$$\tau_i^* = \pi_0 \mathbf{P}_{p^*}^i. \tag{40}$$

For state s, the global error after i uniformisation steps corresponds to the difference between the maximum probability in s and its over-approximation:

$$g_i(s) = \left| \tau_i^*(s) - \tau_i^{\max}(s) \right|. \tag{41}$$

This error depends linearly on the size of the parameter space and exponentially on the number of uniformisation steps, which we summarize as follows.

Proposition 2 *Let $\mathcal{C}_{\mathcal{R}} = (S, \pi_0, \mathbf{R}, L)$ be a pCTMC with multi-affine rates on an n-dimensional rectangular space $\mathcal{R}$, ϕ be an unnested time-bounded CSL path formula and $g_i(s)$ be the global approximation error for the maximum probability of being in state s after i uniformisation steps. Then, there exist $M_1, M_2 < \infty$ such that, for any $s \in S$ and step $i > 0$, an upper bound to the error, $\overline{g}_i \geq \max_{s \in S} g_i(s)$, is given as*

$$\overline{g}_i = \begin{cases} 0 & \text{if } i = 0 \\ \overline{g}_{i-1} \cdot \left(1 + \frac{M_2}{q}\right) + \frac{M_1}{q} w_{\mathcal{R}} & \text{if } i > 0, \end{cases} \tag{42}$$

where $w_{\mathcal{R}} = \max_{j=1,\ldots,n}(x_j^{\top} - x_j^{\perp})$ is the width of $\mathcal{R}$.

Proof See Proposition 2 in "Appendix". □

Let $\hat{\Lambda}_{\phi}(.)(s_0)$ be the approximation of the satisfaction function $\Lambda_{\phi}(.)(s_0)$ for initial state s_0 obtained using standard transient analysis and uniformisation [31]. We now provide an important characterization of $\hat{\Lambda}_{\phi}(.)(s_0)$, which holds for pCTMCs with general polynomial rates.

Theorem 1 *For a pCTMC $\mathcal{C}_{\mathcal{P}}$ on a bounded parameter space $\mathcal{P}$, an initial state s_0 and a finitely-nested and time-bounded CSL path formula ϕ, the approximate satisfaction function $\hat{\Lambda}_{\phi}(.)(s_0)$ is piecewise polynomial in $\mathcal{P}$ over a finite number of subdomains.*

Proof See Theorem 1 in "Appendix". □

5 Refinement-based parameter synthesis

We present algorithms to solve Problems 1 and 2, utilising the approximation of probability bounds introduced in Sect. 4. The algorithms iteratively refine the parameter space $\mathcal{P}$ and compute the probability bounds on the satisfaction function for each subspace until a required accuracy is obtained.

5.1 Threshold synthesis

Algorithm 1 describes the method to solve the threshold synthesis problem with input formula ϕ and threshold $\geq r$. The idea, also illustrated in Fig. 3a, is to iteratively refine the undecided parameter subspace $\mathcal{U}$ (line 3) until the termination condition is met (line 14). At each step, we obtain a partition D of $\mathcal{U}$. For each subspace $\mathcal{R} \in D$, the algorithm computes bounds $\Lambda^{\mathcal{R}}_{\min}$ and $\Lambda^{\mathcal{R}}_{\max}$ on the minimal and maximal probability that $\mathcal{C}_{\mathcal{R}}$ with the initial state s_0 satisfies ϕ (line 5). We then evaluate if $\Lambda^{\mathcal{R}}_{\min}$ is above the threshold r, in which case the satisfaction of the threshold is guaranteed for the whole region $\mathcal{R}$, which is then added to $\mathcal{T}$. Otherwise, the algorithm tests whether $\mathcal{R}$ can be added to the set $\mathcal{F}$ by checking if $\Lambda^{\mathcal{R}}_{\max}$ is below the threshold r. If $\mathcal{R}$ is neither in $\mathcal{T}$ nor in $\mathcal{F}$, it forms an undecided subspace that is added to the

Algorithm 1 Threshold Synthesis

Require: pCTMC $\mathcal{C}_{\mathcal{P}}$ over parameter space $\mathcal{P}$, initial state s_0, CSL path formula ϕ, threshold $\geq r$ and volume tolerance $\varepsilon > 0$
Ensure: $\mathcal{T}, \mathcal{U}$ and $\mathcal{F}$ as in Problem 1
1: $\mathcal{T} \leftarrow \emptyset, \mathcal{F} \leftarrow \emptyset, \mathcal{U} \leftarrow \mathcal{P}$
2: **repeat**
3: $D \leftarrow \mathsf{decompose}(\mathcal{U}), \mathcal{U} \leftarrow \emptyset$
4: **for all** $\mathcal{R} \in D$ **do**
5: $(\Lambda^{\mathcal{R}}_{\min}, \Lambda^{\mathcal{R}}_{\max}) \leftarrow \mathsf{computeBounds}(\mathcal{C}_{\mathcal{R}}, s_0, \phi)$
6: **if** $\Lambda^{\mathcal{R}}_{\min} \geq r$ **then**
7: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{R}$
8: **else if** $\Lambda^{\mathcal{R}}_{\max} < r$ **then**
9: $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{R}$
10: **else**
11: $\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{R}$
12: **until** $\mathrm{vol}(\mathcal{U})/\mathrm{vol}(\mathcal{P}) > \varepsilon$ ▷ where $\mathrm{vol}(A) = \int_A 1 d\mu$

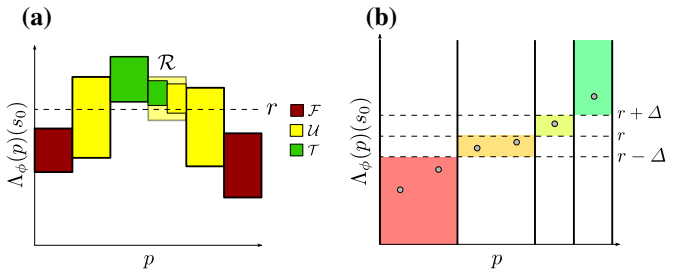


Fig. 3 **a** Refinement in threshold synthesis with $\geq r$. Parameter values are on the x-axis, probabilities on the y-axis. *Each box* describes a parameter region (width), and its probability bounds (height). The refinement of $\mathcal{R}$ yields regions in $\mathcal{T}$ and in $\mathcal{U}$. **b** Initial sampling-guided refinement of $\mathcal{P}$. Sampled probabilities and a tolerance Δ are used to identify regions that are *likely* to be in $\mathcal{T}$ (*green area*, samples $\geq r + \Delta$), in $\mathcal{F}$ (*red*, $\leq r - \Delta$), or close to the threshold r (*orange* and *lime green*, $\in (r - \Delta, r + \Delta)$)

set $\mathcal{U}$. The algorithm terminates when the volume of the undecided subspace is negligible with respect to the volume of the entire parameter space, i.e. $\text{vol}(\mathcal{U})/\text{vol}(\mathcal{P}) \leq \varepsilon$, where ε is the input tolerance. Otherwise, the algorithm continues to the next iteration, where $\mathcal{U}$ is further refined.

5.1.1 Correctness and termination

Correctness of the algorithm follows from the construction of the regions $\mathcal{T}$ (lines 6, 7) and $\mathcal{F}$ (lines 8, 9). The termination condition guarantees the required bound of the relative volume of $\mathcal{U}$ (line 12). Termination of the algorithm for (possibly nested) CSL properties is stated below.

Proposition 3 *For a pCTMC $\mathcal{C}_{\mathcal{P}}$ over a parameter space $\mathcal{P}$, initial state s_0, CSL path formula ϕ and volume tolerance ε, Algorithm 1 terminates.*

Proof See Proposition 3 in "Appendix". □

5.1.2 Initial decomposition

Optionally, a heuristic based on an initial decomposition precedes the refinement procedure. The initial decomposition can speed up the refinement, since it decomposes the parameter space $\mathcal{P}$ in advance. It is guided by a priori uniform sampling of probability values. In particular, we sample points $p_1, p_2, \ldots, p_n \in \mathcal{P}$ and compute $\hat{\Lambda}_\phi(p_i)(s_0)$ for $i = 1, \ldots, n$ using standard CSL model checking. Then, we partition $\mathcal{P}$ into subspaces that set apart samples where $\hat{\Lambda}_\phi(p_i)(s_0) \geq r$ from those where $\hat{\Lambda}_\phi(p_i)(s_0) < r$. As depicted in Fig. 3b, we also use a tolerance $\Delta > 0$ to identify regions close to the threshold that are more likely to be further decomposed. In this case the initial decomposition returns four regions. Our experiments demonstrates that, in some cases, depending on the shape of the satisfaction function and the threshold r, the initial decomposition accelerates the synthesis.

5.2 Max synthesis

Algorithm 2 is used to solve the max synthesis problem, which returns the set $\mathcal{T}$ containing the parameter valuations that maximize the probability of the path formula ϕ and the set $\mathcal{F}$ not yielding the maximum value. Starting from $\mathcal{T} = \mathcal{P}$, the algorithm iteratively refines $\mathcal{T}$ until the probability tolerance condition at Problem 2 is met (line 14).

Let D be a partition of $\mathcal{T}$. For each subspace $\mathcal{R} \in D$, the algorithm computes bounds $\Lambda^{\mathcal{R}}_{\min}$ and $\Lambda^{\mathcal{R}}_{\max}$ on the minimal and maximal probability that $\mathcal{C}_{\mathcal{R}}$ with the initial state s_0 satisfies ϕ (line 5). The algorithm then rules out subspaces that are guaranteed to be included in $\mathcal{F}$, by deriving an under-approximation (M) to the maximum satisfaction probability (line 7). If $\Lambda^{\mathcal{R}}_{\max}$ is below the under-approximation, the subspace $\mathcal{R}$ can be safely added to the set $\mathcal{F}$ (line 9). Otherwise, it is kept in $\mathcal{T}$.

We consider two approaches for deriving the bound M, namely a naive approach and a sampling-based approach. In the naive method, we set M to the maximum over the least bounds in the partition of $\mathcal{T}$, that is, $M = \max\{\Lambda^{\mathcal{R}'}_{\min} \mid \mathcal{R}' \in D\}$. Let $\overline{\mathcal{R}}$ be the region with highest lower bound. The sampling-based method, illustrated in Algorithm 3, improves on this by sampling a set of parameters $\{p_1, p_2, \ldots\} \subseteq \overline{\mathcal{R}}$ (line 2) and taking the highest value of $\hat{\Lambda}_\phi(p)(s_0)$, that is, $M = \max\left\{\hat{\Lambda}_\phi(p_i)(s_0) \mid p_i \in \{p_1, p_2, \ldots\}\right\}$ (line 3). Each $\hat{\Lambda}_\phi(p)(s_0)$ is computed through regular CSL model checking, and is equally expensive as computing the

Algorithm 2 Max Synthesis

Require: pCTMC $\mathcal{C}_{\mathcal{P}}$ over parameter space $\mathcal{P}$, initial state s_0, CSL path formula ϕ and probability tolerance $\epsilon > 0$
Ensure: $\Lambda_\phi^\perp$, $\Lambda_\phi^\top$, $\mathcal{T}$ and $\mathcal{F}$ as in Problem 2
1: $\mathcal{F} \leftarrow \emptyset, \mathcal{T} \leftarrow \mathcal{P}$
2: **repeat**
3: $D \leftarrow \mathsf{decompose}(\mathcal{T}), \mathcal{T} \leftarrow \emptyset, \Lambda_\phi^\perp \leftarrow +\infty, \Lambda_\phi^\top \leftarrow -\infty$
4: **for all** $\mathcal{R} \in D$ **do**
5: $(\Lambda_{\min}^{\mathcal{R}}, \Lambda_{\max}^{\mathcal{R}}) \leftarrow \mathsf{computeBounds}(\mathcal{C}_{\mathcal{R}}, s_0, \phi)$
6: $M \leftarrow \mathsf{getMaximalLowerBound}(D)$
7: **for all** $\mathcal{R} \in D$ **do**
8: **if** $\Lambda_{\max}^{\mathcal{R}} < M$ **then**
9: $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{R}$
10: **else**
11: $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{R}$
12: $\Lambda_\phi^\perp \leftarrow \min\{\Lambda_\phi^\perp, \Lambda_{\min}^{\mathcal{R}}\}$
13: $\Lambda_\phi^\top \leftarrow \max\{\Lambda_\phi^\top, \Lambda_{\max}^{\mathcal{R}}\}$
14: **until** $\Lambda_\phi^\top - \Lambda_\phi^\perp > \epsilon$

Algorithm 3 Sampling-guided computation of a maximal lower bound

Require: Parameter decomposition D and number of samples n
Ensure: $\overline{M}$, an improved lower bound for max probability in D
1: $\overline{\mathcal{R}} = \arg\max_{\mathcal{R}' \in D} \Lambda_{\min}^{\mathcal{R}'}$
2: $(p_1, \ldots, p_n) \leftarrow \mathsf{Uniform}(\overline{\mathcal{R}}, n)$
3: $\overline{M} \leftarrow \max_{p_i} \hat{\Lambda}(p_i)(s_0)$

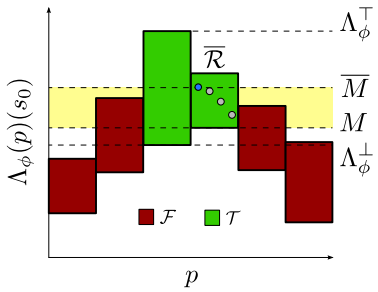


Fig. 4 Refinement in max synthesis. The two outermost regions (in *red*) cannot contain the maximum, as their upper bound is below the maximum lower bound (M) found at region $\overline{\mathcal{R}}$. The maximum lower bound is improved by sampling several points $p \in \overline{\mathcal{R}}$ and taking the highest value ($\overline{M}$) of the satisfaction function $\hat{\Lambda}_\phi(p)(s_0)$. The *yellow area* highlights the improvement

bounds on a fixed pCTMC. The sampling method results in an improved under-approximation to the maximum of the satisfaction function. As a result, the bound rules out more regions, and fewer refinements are required in the next iteration (see Fig. 4).

5.2.1 Correctness and termination

The correctness of the algorithm derives from the construction of the sets $\mathcal{T}$ (lines 12, 13) and $\mathcal{F}$ (lines 8, 9), and from the termination condition (line 14).

We remark that, for nested properties, the satisfaction function is in general discontinuous, which allows $\Lambda_{\max}^{\mathcal{R}} - \Lambda_{\min}^{\mathcal{R}} > \epsilon$ when $\mathcal{T}$ contains a jump discontinuity. This prevents

the algorithm from terminating. For this reason a volume-based stopping criterion, such as $\text{vol}(\mathcal{T}) \leq \epsilon$ for $\epsilon > 0$, which replaces the condition on line 14, should be used when analysing nested properties. Indeed, the volume of any region containing such a discontinuity can be made arbitrarily small in a finite number of refinement steps, as discussed in the proof of Proposition 3. With unnested properties, the following proposition ensures termination.

Proposition 4 *For a pCTMC $\mathcal{C}_{\mathcal{P}}$ over a parameter space $\mathcal{P}$, initial state s_0, non-nested CSL path formula ϕ and tolerance ϵ, Algorithm 2 terminates.*

Proof See Proposition 4 in "Appendix". □

5.3 Complexity

The time complexity of the procedure computing the probability bounds for a fixed region has been discussed in Sect. 4.2. The overall runtime of both algorithms further depends on the number of subspaces that are required to obtain the desired precision. This number scales exponentially in the number of parameters and linearly in the volume of the parameter space. However, in practice, the number of required subspaces also depends on the shape of the satisfaction function and the type of synthesis.

6 Results

We implemented the synthesis algorithms on top of the tool PRISM 4.0 [32]. Currently, a prototype command-line version is available at https://github.com/Palmik/prism-pse/. Models and properties are specified using the native specification languages of PRISM. Note that the online version of the tool only supports linear rate functions and non-nested formulas.

We demonstrate the applicability and efficiency of the developed algorithms on three case studies. We run all experiments on a Linux workstation with an AMD Phenom™ II X4 940 Processor @ 3 GHz, 8 GB DDR2 @ 1066 MHz RAM.

6.1 Epidemic model

The SIR model [27] describes the epidemic dynamics in a well-mixed and closed population of susceptible (S), infected (I) and recovered (R) individuals. In the model, a susceptible individual is infected after a contact with an infected individual with rate k_i. Infected individuals recover with rate k_r, after which they are immune to the infection. We can describe this process with the following biochemical reaction model with mass action kinetics (i.e. the rate functions are linear with respect to the parameters):

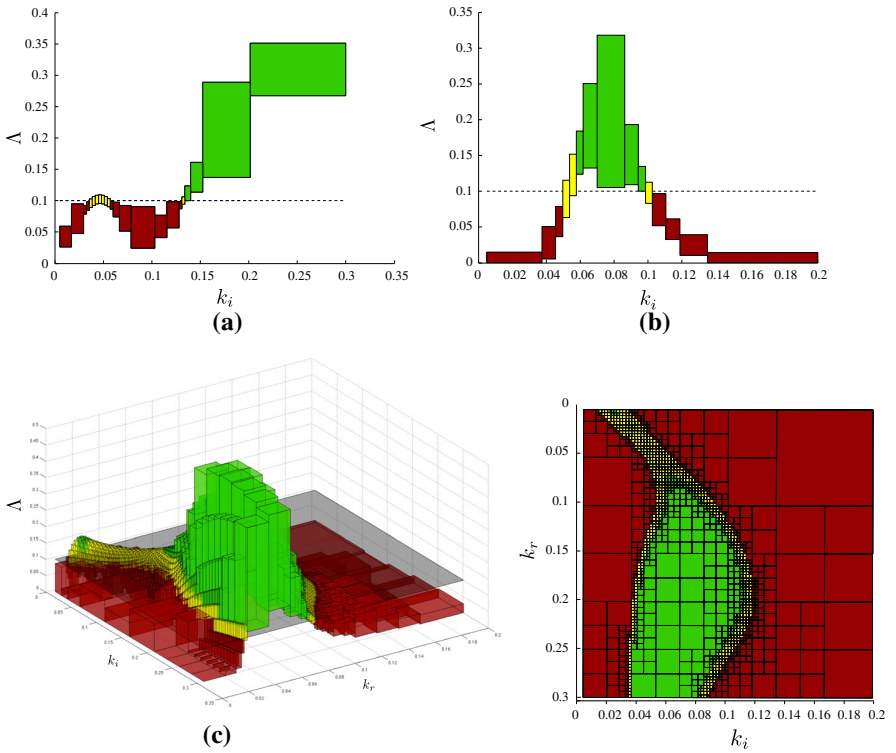
$$S + I \xrightarrow{k_i} I + I \qquad \text{(Infection)}$$

$$I \xrightarrow{k_r} R \qquad \text{(Recovery)}$$

We represent the model as a pCTMC with parameters $k_i \in [0.005, 0.3]$ and $k_r \in [0.005, 0.2]$, and initial populations $S = 95$, $I = 5$, $R = 0$.

We consider the time-bounded CSL path formula $\phi = (I > 0)\text{U}^{[100,120]}(I = 0)$, specifying behaviour where the infection lasts for at least 100 time units, and dies out before 120 time units. Property and parameters are taken from [9], where the authors estimate the satisfaction function for ϕ following a Bayesian approach.[1]

[1] In [9], a linear-time specification equivalent to ϕ is given.



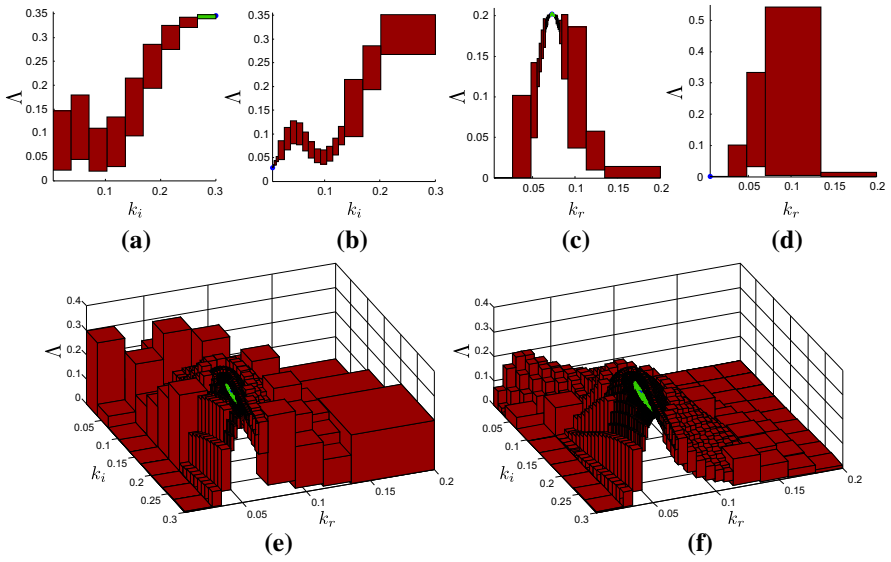
	$k_i \times k_r$	Runtime	Subspaces
a)	$[0.005, 0.3] \times 0.05$	42.2 s	23
b)	$0.12 \times [0.005, 0.2]$	26.7 s	15
c,d)	$[0.005, 0.3] \times [0.005, 0.2]$	29.3 min	1320

Fig. 5 Solution to threshold synthesis problems for the SIR model and the property $P_{\geq 0.1}[(I > 0)\mathrm{U}^{[100,120]}(I = 0)]$. *Plots* **c** and **d** depict the same result with two different angles. Runtime and number of subspaces in the final partition of $\mathcal{P}$ are listed. Volume tolerance is $\varepsilon = 10$ %. *Colour* code is as in Fig. 3a

First, we perform threshold synthesis experiments to find infection and recovery rates for which ϕ is satisfied with probability at least $r = 10$ %. Figure 5 illustrates the solutions for one-dimensional parameter spaces (plots a, b) obtained by fixing $k_i = 0.12$ and $k_r = 0.05$, respectively; and for the two-dimensional parameter space (plots c, d). Results evidence that a significantly higher number of refinement steps is required for parameter subspaces where the satisfaction function Λ is close to the probability threshold r.

Second, we perform max and min synthesis experiments for property ϕ over one-dimensional and two-dimensional parameter spaces. Results are summarized in Fig. 6. For the experiments in Fig. 6b, c we observe that, in order to meet the desired probability tolerance, a high number of refinement steps is required due to two local extrema close to the minimizing region and a bell-shaped Λ with the maximizing region at the top, respectively.

Our precise results for the problem in Fig. 6a improve on the estimation in [9], where in a similar experiment the maximal satisfaction probability is imprecisely registered at $k_i = 0.25$.



	$k_i \times k_r$	Runtime	Subspaces	$\Lambda_\phi^\perp$	$\Lambda_\phi^\top$	$\mathcal{T}$
a)	$[0.005, 0.3] \times 0.05$	16.5 s	9	33.94%	34.85%	$[0.267, 0.3] \times 0.05$
b)	$[0.005, 0.3] \times 0.05$	49.5 s	21	2.83%	2.91%	$[0.005, 0.0054] \times 0.05$
c)	$0.12 \times [0.005, 0.2]$	99.7 s	57	19.94%	20.42%	$0.12 \times [0.071, 0.076]$
d)	$0.12 \times [0.005, 0.2]$	10.4 s	5	†	0.005%	$0.12 \times [0.005, 0.026]$
e)	$[0.005, 0.3] \times [0.005, 0.2]$	3.6 h	5817	35.01%	35.72%	$[0.217, 0.272] \times [0.053, 0.059]$
f)	$[0.005, 0.3] \times [0.005, 0.2]$	6.2 h	10249	34.77%	35.72%	$[0.209, 0.29] \times [0.051, 0.061]$

Fig. 6 Solution to max (**a**, **c**, **e**, **f**) and min (**b**, **d**) synthesis for the SIR model and the formula $\phi = (I > 0)\mathrm{U}^{[100,120]}(I = 0)$. Sampling-based refinement is used for all experiments but **e**. Colour code is as in Fig. 4. In the table, we report runtime, number of subspaces, approximation ($\Lambda_\phi^\perp$ and $\Lambda_\phi^\top$) of min or max probability, and bounding box of $\mathcal{T}$. Probability tolerance is $\epsilon = 1$ % for max synthesis and $\epsilon = 0.1$ % for min synthesis. † The found value, 3.05×10^{-10}, is of the same order of magnitude as the precision used during uniformisation. **a** Max, k_r fixed. **b** Min, k_r fixed. **c** Max, k_i fixed. **d** Min, k_i fixed. **e** Max, sampling-based. **f** Max, no-sampling

We also compare the solutions to the max synthesis problem over the two-dimensional parameter space obtained by applying Algorithm 2 with sampling (Fig. 6e) and without (Fig. 6f). In the former case, a more precise $\mathcal{T}$ region is obtained (with volume 2.04 times smaller than in the approach without sampling), hence giving a more accurate approximation of the max probability. Sampling also allows us to rule out earlier those parameter regions that are outside the final solution, thus avoiding unnecessary decompositions and improving the runtime (1.72 times faster than in the approach without sampling). This is visible by the coarser approximations of probabilities in the $\mathcal{F}$ region.

6.1.1 Parameter synthesis for nested CSL formulas

We use the SIR model to illustrate parameter synthesis for a nested CSL formula. We consider initial populations of $S = 3$, $I = 1$ and $R = 0$ to obtain a small model with only 14 reachable states, which allows us visualise the main steps of the synthesis algorithm. We modify the original path formula ϕ to specify behaviour where the infection lasts at least 100 time units and, before 200 time units, the system reaches a state where the infection becomes extinct

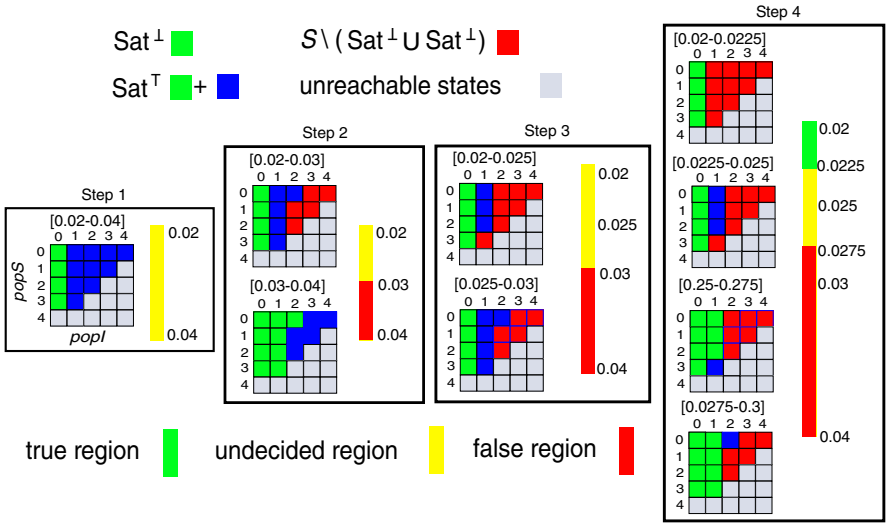


Fig. 7 Visualisation of four steps of the threshold synthesis algorithm for a small variant of the SIR model, nested CSL formula $P_{>0.1}[(I > 0)\mathrm{U}^{[100,120]}(\Psi)]$ ($\Psi = P_{>0.9}[F^{[0,100]}(I = 0)]$) and parameter space $k_i \times k_r = 0.12 \times [0.02, 0.04]$. The two-dimensional grids represent the state space. Since the total population is preserved by the model dynamics, each state is unambiguously given by the population of S ($popS$) and population of I ($popI$). For a given parameter region, the satisfaction sets $\mathrm{Sat}_\perp(\Psi)$ and $\mathrm{Sat}_\top(\Psi)$ for Ψ are depicted using *different colouring* of the grid cells. For each step, the *vertical coloured bar* illustrates the current partitioning of the parameter space into regions $\mathcal{T}, \mathcal{U}$ and $\mathcal{F}$

before time 100 with probability higher than 90 %. Such a property can be formalised as the nested CSL path formula $\phi' = (I > 0)\mathrm{U}^{[100,120]}(\Psi)$, where $\Psi = P_{>0.9}[F^{[0,100]}(I = 0)]$.

We show threshold synthesis for property $P_{>0.1}[\phi']$ and parameter space $k_i \times k_r = 0.12 \times [0.02, 0.04]$. Recall that the probability bounds $\Lambda_{\phi',\min}$ and $\Lambda_{\phi',\max}$ for a nested formula ϕ' are computed as:

$$\Lambda_{\phi',\min}(s) = \Lambda_{\phi^\perp,\min}(s) \text{ where } \phi^\perp = (I > 0)\mathrm{U}^{[100,120]}(\mathrm{Sat}_\perp(\Psi))$$
$$\Lambda_{\phi',\max}(s) = \Lambda_{\phi^\top,\min}(s) \text{ where } \phi^\top = (I > 0)\mathrm{U}^{[100,120]}(\mathrm{Sat}_\top(\Psi))$$

Figure 7 depicts four steps of the refinement algorithm. In the first step, $\mathrm{Sat}_\perp(\Psi)$ contains only states where the population of I is 0 and $\mathrm{Sat}_\top(\Psi)$ contains all reachable states. The threshold synthesis algorithm for $P_{>0.1}[\phi']$ partitions the parameter space into a single undecided region. In the second step, the parameter space refinement yields a refinement of the satisfaction sets. In particular, for $k_r = [0.02, 0.03]$, we observe that some states no longer appear in $\mathrm{Sat}_\top(\Psi)$ (shown red-coloured), while, for $k_r = [0.03, 0.04]$, some states are added to $\mathrm{Sat}_\perp(\Psi)$ (shown green-coloured). Note that, although $\mathrm{Sat}_\perp(\Psi) \neq \mathrm{Sat}_\top(\Psi)$, the approximation is precise enough to decide that the latter parameter subspace is false, since $\Lambda_{\phi',\max}(s) < 0.1$. In the third refinement step, we manage to refine only $\mathrm{Sat}_\top(\Psi)$ for region $k_r = [0.02, 0.025]$. The region remains still undecided as well as region $k_r = [0.025, 0.03]$. Finally, in the fourth step, the satisfaction sets for $k_r = [0.02, 0.0225]$ collapse, i.e. $\mathrm{Sat}_\perp(\Psi) = \mathrm{Sat}_\top(\Psi)$, which allows us to conclude that the region is true. Region $k_r = [0.0275, 0.03]$ can also be decided, despite the fact that $\mathrm{Sat}_\perp(\Psi) \neq \mathrm{Sat}_\top(\Psi)$ here.

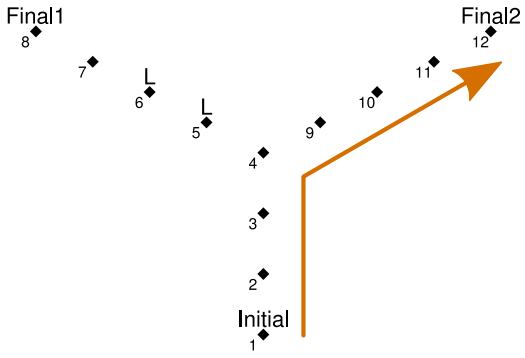


Fig. 8 Single-junction DNA walker circuit. In *orange*: the walker starts on the Initial anchorage and moves *right* at the junction, eventually quenching the fluorophore at the Final2 anchorage

This example demonstrates the key aspects of the parameter synthesis method for nested CSL formulas. In particular, it shows that the refinement of the parameter space yields the refinement of the satisfaction sets and can result in parameter subspaces where, for a given state formula Φ, $\mathrm{Sat}_{\perp}(\Phi) = \mathrm{Sat}_{\top}(\Phi)$. Note that, if additional refinements are needed for such subspaces, the corresponding probability bounds can be computed in the same way as non-nested formulas. This example also demonstrates that a region can be decided even when $\mathrm{Sat}_{\perp}(\Phi) \neq \mathrm{Sat}_{\top}(\Phi)$. This considerably reduces the number of required refinements for the subformula Φ. In general, the synthesis algorithms for the nested formulas have a higher complexity, since the nesting of probabilistic operators increases the number of regions to analyse.

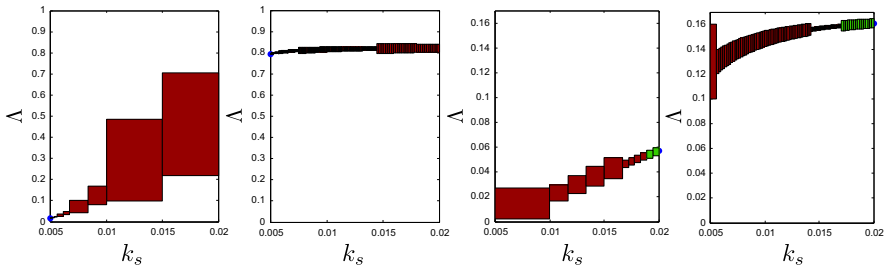
6.2 DNA walkers

We revisit models of a DNA walker, a man-made molecular motor that traverses a track of anchorages and can take directions at junctions in the track [43], which can be used to create circuits that evaluate Boolean functions. `PRISM` models of the walker stepping behaviour were developed previously [17] based on rate estimates in the experimental work. The walker model is modified here to allow uncertainty in the stepping rate, and we consider its behaviour over a single-junction circuit, see Fig. 8. Following [17], the stepping rate k is parameterised by d, the distance between the walker-anchorage complex and an uncut anchorage, and d_a, the distance between consecutive anchorages, and is defined as:

$$k = \begin{cases} k_s & \text{when } d \leq 1.5d_a \\ c \cdot k_s/50 & \text{when } 1.5d_a < d \leq 2.5d_a \\ c \cdot k_s/100 & \text{when } 2.5d_a < d \leq 24\text{nm} \\ 0 & \text{otherwise.} \end{cases} \tag{43}$$

where the base stepping rate $k_s \in [0.005, 0.020]$ is now defined as an interval, as opposed to the original value of 0.009. We have also added factor c for steps between anchorages that are not directly adjacent, but we will assume $c = 1$ for now. The base stepping rate may depend on buffer conditions and temperature, and we want to verify the robustness of the walker with respect to the uncertainty in the value of k_s.

We compute the minimal probability of the walker making it onto the correct final anchorage by time T (min synthesis for the formula $\phi = \mathrm{F}^{[T,T]}$ finish-correct) and the maximum probability of the walker making it onto the incorrect anchorage by time T (max synthesis for the formula $\phi = \mathrm{F}^{[T,T]}$ finish-incorrect). In Fig. 9, we list the probabilities at $T = 15, 30, 45, 200$ min and depict the solutions and the parameter space decompositions



Time bound	Min. correct	Max. incorrect	Runtime ∅	Runtime Sampling	Subspaces ∅	Subspaces Sampling
$T = 15$	1.68%	5.94%	0.55 s	0.51 s	22	11
$T = 30$	14.86%	10.15%	1.43 s	1.35 s	35	15
$T = 45$	33.10%	12.25%	3.53 s	2.14 s	61	21
$T = 200$	79.21%	16.47%	213.57s	88.97 s	909	329

Fig. 9 Experiments for the DNA walker model: min-synthesis for $\phi = \mathrm{F}^{[T,T]}$ finish-correct and max-synthesis for $\phi = \mathrm{F}^{[T,T]}$ finish-incorrect using $k_s \in [0.005, 0.020]$, $c = 1$ and probability tolerance $\epsilon = 1$ %. Plots show the solutions and the decompositions for the min (**a**, **b**) and max (**c**, **d**) synthesis and $T = 15{,}200$ min. Colour code is as in Fig. 4. In the table, results are reported also for $T = 30, 45$. The runtime and subspaces are listed only for min-synthesis (the results for max-synthesis are similar). **a** Min, $T = 15$. **b** Min, $T = 200$. **c** Max, $T = 15$. **d** Max, $T = 200$

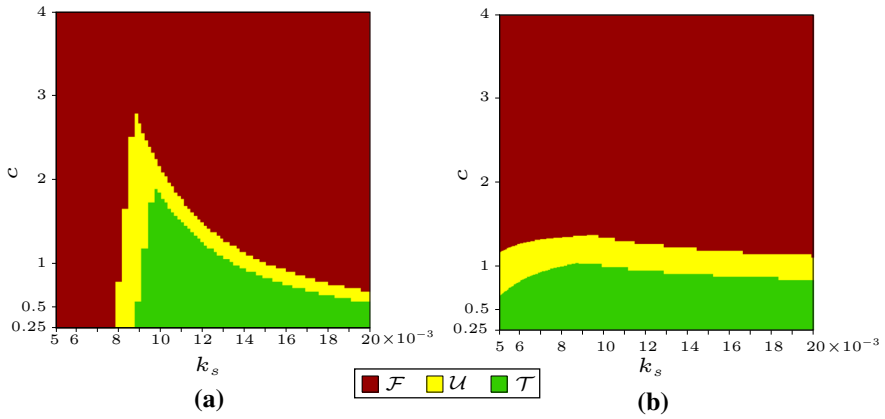
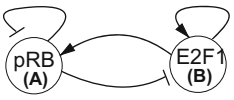


Fig. 10 The computation and results of the threshold synthesis for the DNA walker model, $k_s \times c \in [0.005, 0.020] \times [0.25, 4]$ and different formulas, using volume tolerance $\varepsilon = 10$ %. **a** $P_{\geq 0.4}[\mathrm{F}^{[30,30]}$ finish-correct$] \wedge P_{\leq 0.08}[\mathrm{F}^{[30,30]}$ finish-incorrect$]$, runtime 282.5 s, 3489 subspaces$*$. **b** $P_{\geq 0.8}[\mathrm{F}^{[200,200]}$ finish-correct$] \wedge P_{\leq 0.16}[\mathrm{F}^{[200,200]}$ finish-incorrect$]$, runtime 8.2 h, 47229 subspaces$*$. Results are obtained by solving the synthesis problem for each sub-formula and by combining the output regions as in Eq. 6. $*$: derived as the sum of runtimes and subspaces for each sub-formula

for both experiments at $T = 15, 200$ min. For time $T = 30, 45, 200$, we note that the walker is robust, as the minimal guaranteed probability for the correct outcome is greater than the maximum possible probability for the incorrect outcome. For time $T = 15$ this is not the case. From plots (a–d), we observe that minimum and maximum probability values are obtained for minimum and maximum values of k_s, respectively.

We also consider a property that provides bounds on the *ratio* between the walker finishing on the correct versus the incorrect anchorage. The rates $c \cdot k_s/50$ and $c \cdot k_s/100$ correspond



(a)

$A < M_A \wedge B > 0 \rightarrow$	$k_1 \frac{B}{(K_{m1}+B)} \frac{J_{11}}{J_{11}+A} : A' = A + 1$
$B < M_B \rightarrow$	$k_2 \frac{a^2+B^2}{(K_{m2}^2+B^2)} \frac{J_{12}}{J_{12}+A} : B' = B + 1$
$B < M_B \rightarrow$	$k_p : B' = B + 1$
$A > 0 \rightarrow$	$\gamma_A \cdot A : A' = A - 1$
$B > 0 \rightarrow$	$\gamma_B \cdot B : B' = B - 1$

(b)

Fig. 11 **a** Two-gene regulatory circuit controlling G_1/S transition in mammalian cell cycle ($\dashv$ indicates inhibition, $\rightarrow$ activation). **b** Stochastic Michaelis–Menten model of the G_1/S regulatory circuit adapted from [42] in the guarded command notation ($guard \rightarrow rate : update$). A and B are the number of proteins pRB and E_2F_1, respectively. Rates are expressed in s^{-1}. Production rate parameters are as in [42]: $k_1 = 1$, $k_2 = 1.6$, $k_p = 0.05$, $K_{m1} = 0.5$, $K_{m2} = 4$, $a = 0.04$, $J_{11} = 0.5$ and $J_{12} = 5$. $M_A = M_B = 30$ are the bounds on A and B, estimated through stochastic simulations. Synthesis parameters are: $\gamma_A \in [0.005, 0.5]$ (degradation rate of pRB), and $\gamma_B \in [0.005, 0.5]$ (degradation rate of E_2F_1)

to the walker stepping onto anchorages that are not directly adjacent, which affects the probability for the walker to end up on the unintended final anchorage. For higher values of c, the walker is more likely to reach the unintended final anchorage more often. Now we add uncertainty on the value of c, so that $c \in [0.25, 4]$, and define the performance related property by $P_{\geq 0.4}[\mathrm{F}^{[30,30]}\ \text{finish-correct}] \wedge P_{\leq 0.08}[\mathrm{F}^{[30,30]}\ \text{finish-incorrect}]$, that is, the probability of the walker to make it onto the correct anchorage is at least 40 % by time $T = 30$ min, while the probability for it to make it onto the incorrect anchorage is no greater than 8 %. In other words, we require a correct signal of at least 40 % and a correct-to-incorrect ratio of at least 5 by time $T = 30$ min. We define a similar property at time $T = 200$ min, this time requiring a signal of at least 80 %: $P_{\geq 0.8}[\mathrm{F}^{[200,200]}\ \text{finish-correct}] \wedge P_{\leq 0.16}[\mathrm{F}^{[200,200]}\ \text{finish-incorrect}]$. The synthesized ranges of k_s and c where the properties hold are shown in Fig. 10. Note that in this case the rate function is a multi-affine polynomial and for fixed c (Fig. 9) the function is linear.

6.3 Gene regulation of mammalian cell cycle

We consider the gene regulation model published in [42]. The model is shown in Fig. 11a and explains the regulation of a transition between the early phases of the mammalian cell cycle. In particular, it targets the transition from the control G_1-phase to S-phase (the synthesis phase). The G_1-phase makes an important checkpoint controlled by a *bistable regulatory circuit*, based on an interplay of the retinoblastoma protein pRB, denoted by A (the so-called tumour suppressor, HumanCyc:HS06650) and the retinoblastoma-binding transcription factor E_2F_1, denoted by B (a central regulator of a large set of human genes, HumanCyc:HS02261). At high concentration levels (*high mode*), B activates the G_1/S transition mechanism. On the other hand, a low concentration of B (*low mode*) prevents committing to S-phase. Depending on the parameters, the positive autoregulation of B causes bistability of its concentration, i.e. the existence of two stable states. Of specific interest are the degradation rates of A (γ_A) and B (γ_B). When mitogenic stimulation increases under conditions of active growth, rapid phosphorylation of A starts and makes the degradation of unphosphorylated A stronger (i.e. the rate γ_A increases). This causes B to lock in the high stable mode implying the cell cycle commits to the S-phase. Since mitogenic stimulation influences the degradation rate of A, our goal is to study the population distribution around the low and high steady states and to explore the effect of γ_A and γ_B.

The ODE model in [42] describes this system using Michaelis–Menten (MM) kinetics [35], which provides a deterministic approximation for enzyme-catalysed reactions. In this work, we derive a discrete stochastic translation of the ODE model by directly using the MM rates. The model is illustrated in Fig. 11b. The corresponding CTMC has 961 states and 3690 transitions. For details on the adequacy of the MM approximation in the stochastic settings, we refer the reader to [37,39].

We apply threshold synthesis to find the degradation rates that lead to bistability. In particular, synthesis parameters are $\gamma_A \in [0.005, 0.5]$ and $\gamma_B \in [0.005, 0.5]$ and thus the rate functions are linear with respect to the parameters. We formalize stability using time-bounded properties with time horizon 1000 s, which reflects the time scale of the gene regulation response. The stabilization of the model's dynamics within this time horizon was also confirmed by a steady-state analysis for different values of the parameters. We use atomic propositions H and L to denote the high and low mode of B, respectively. In the following experiments, we assume L is true if $B < 2$ and H is true if $B > 4$.

Bistability is commonly expressed as the property of reaching and staying in either one of two different regions of the state space. In time-bounded CSL, this can be expressed using the formula $P_{\geq r_L}[\mathrm{G}^I L] \wedge P_{\geq r_H}[\mathrm{G}^I H]$, where I spans the final time window and the probability of resting in the two modes is at least r_L and r_H, respectively. However, threshold synthesis for this property, $I = [900, 1000]$ and different combinations of r_L and r_H resulted in empty $\mathcal{T}$ regions and negligible $\mathcal{U}$ regions, indicating that no parameters can be found that meet this formulation of bistability. Note that this result agrees with the analysis performed in [11] using a different stochastic translation of the ODE model and parameter exploration techniques.

Therefore, we analyse whether there is at least a weaker type of bistability in the form of a bi-modal distribution at time $t = 1000$. The existence of the bi-modal distribution is formalized using the future operator F as:

$$P_{\geq r_L}[\mathrm{F}^t L] \wedge P_{\geq r_H}[\mathrm{F}^t H].$$

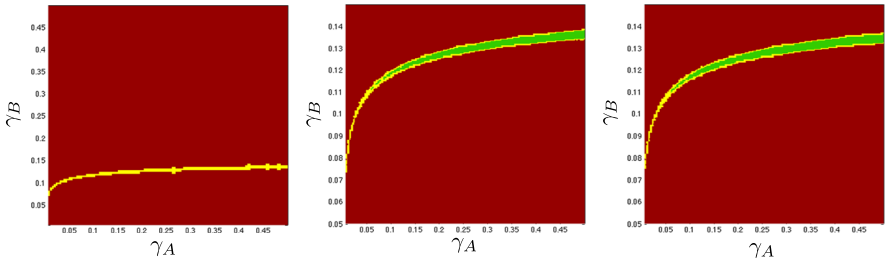
Results for $r_L = r_H = 0.4$ are summarised in Fig. 12. In plot (a), we observe that most of the degradation rates violate the formula, with the exception of an undecided region that spans the domain of γ_A and $\gamma_B \in [0.05, 0.15]$. Although this result is consistent with input volume tolerance ε, it does not provide a precise answer to whether satisfiable parameters exist or not.

In order to resolve the undecided region, we perform the same experiment by restricting the parameter space to the area of interest (plot b). Indeed, note that, by the volume tolerance criterion, a lower volume of $\mathcal{P}$ implies a lower volume of $\mathcal{U}$. An alternative (less efficient) solution would be keeping $\mathcal{P}$ unchanged and decreasing ε. The results demonstrate that a bi-modal distribution is reached for a relevant set of parameters. In particular, we show that, for any value of γ_A approximately ranging in $[0.08, 0.5]$, there exists a value of γ_B that satisfy the property.

To complement the above analysis, we further analyse how long the system remains in the low and high mode. To this purpose, we synthesise parameters such that the following cumulative reward property is satisfied:

$$\mathrm{R}_{\geq 400}\left[\mathrm{C}^{\leq 1000}\right](B < 2) \wedge \mathrm{R}_{\geq 400}\left[\mathrm{C}^{\leq 1000}\right](B > 4)$$

where, for subformula $\mathrm{R}_{\geq 400}[\mathrm{C}^{\leq 1000}](B \sim X)$, the state reward ρ is such that: $\forall s \in S.\, \rho(s) = 1 \Leftrightarrow B \sim X$ in s. Figure 12c illustrates the results of parameter synthesis restricted to the parameter space as in the previous experiment. The parameters that satisfy the reward property



	$\gamma_A \times \gamma_B$	Runtime∗	Subspaces∗
a)	$[0.005, 0.5] \times [0.005, 0.5]$	618 s	809
b)	$[0.005, 0.5] \times [0.05, 0.15]$	1256 s	1817
c)	$[0.005, 0.5] \times [0.05, 0.15]$	1882 s	1742

Fig. 12 Bistability analysis in the G_1/S regulatory circuit model: threshold synthesis for property $P_{\geq 0.4}[\mathrm{F}^I\, L] \wedge P_{\geq 0.4}[\mathrm{F}^I\, H]$ (**a**, **b**) and $\mathrm{R}_{\geq 400}[\mathrm{C}^{\leq 1000}](B < 2) \wedge \mathrm{R}_{\geq 400}[\mathrm{C}^{\leq 1000}](B > 4)$ (**c**) with volume tolerance $\varepsilon = 1$ %. In **b** and **c** we improve precision by restricting the range of γ_B. Colour code is as in Fig. 3a. Results are obtained by solving the synthesis problem for each sub-formula and by combining the output regions as in Eq. 6. *Asterisks* derived as the sum of runtimes and subspaces for each sub-formula

are aligned with the parameters leading to the bimodal distribution. This experiment confirms the existence of a certain form of the bistability in the stochastic version of the model. Moreover, our results are comparable with those obtained using numerical simulations of the ODE model and bifurcation analysis in [42], where bistability is registered[2] for $\gamma_A \in [0.007, 0.03]$ and fixed $\gamma_B = 0.1$.

7 Related work

The work by Brim et al. [11] first introduced extensions of CSL model checking and of the uniformisation method for computing safe probability bounds in parametric models of stochastic reaction networks. The methods are applied to the problem of parameter exploration, i.e. finding bounds that approximate the satisfaction function arbitrarily well. Our work extends [11] with the following five main contributions. (1) Definition of the threshold and max synthesis problems. (2) Synthesis algorithms that combine iterative refinement of the parameter space with sampling of probability values. In contrast to [11], where every parameter region needs to be analyzed, our algorithms focus only on the regions relevant to solve the synthesis problem, so avoiding unnecessary computation. (3) More general class of supported rate functions. Specifically, we allow multi-affine dependency of the rate functions in the parameters, while the method in [11] supports only rate functions linear in the parameters. (4) Convergence analysis of the approximation error (see Proposition 2). (5) Theorem 1, which provides a precise mathematical characterization for the satisfaction function of generic time-bounded and finitely-nested CSL formulas. Furthermore, the gene regulation model in Sect. 6.3 is also studied by Brim et al. [11]. They translate the original ODE model [42] into the framework of stochastic mass action kinetics, while we consider a stochastic Michaelis–Menten approximation. In [11], no parameters are found that meet

[2] In [42], we believe that there is a typo in the figure illustrating bistability (Figure 2B). The range of ϕ_{pRB} (corresponding to γ_A) should read 0.005–0.035.

the bistability requirement, expressed using 'globally' and cumulative reward formulas. Our analyses confirm that the model does not exhibit bistability if defined with 'globally' formulas, but that bistability occurs if formulated as a bounded reachability or a cumulative reward formula. This is explained by the different stochastic encodings of the original ODE model.

Parameter synthesis for CTMCs and bounded reachability specifications is considered in [23]. The authors show that the problem can be reduced to the analysis of the polynomial function describing the reachability probability of a given target state. As illustrated in Sect. 4.1, the main limitation is the high degree of the polynomials, which is determined by the number of uniformisation steps. Thus, in contrast to our work, only an approximate solution can be obtained using discretization of the parameter space and the experimental evaluation is limited to one simple case study (a CTMC with 19 states and 54 transitions).

When considering linear-time specifications, specific restrictions can be placed on the rate function to result in a smooth satisfaction function (i.e. having derivatives of all orders). In that case the function can be approximated using statistical methods such as Gaussian Process regression, which leverage smoothness [9]. Such restrictions require the rate function to be smooth in the parameters and polynomial in the state vector. Instead, the synthesis method we presented imposes limitations only on the parameter dependency. In contrast to our approach, the statistical estimation of the satisfaction function cannot provide guaranteed results. Moreover, the precision of the estimation strongly depends on a number of experimental design choices, e.g. how many and which parameter values to sample, while our algorithms provide results with arbitrary precision and do not require any information except the inputs to the synthesis problem.

A concept similar to smoothness, uniform continuity, can be used to obtain an unbiased statistical estimator for the satisfaction function [25]. Inference of parameter values in probabilistic models from time-series measurements is a well studied area of research [2,10], but different from the problem we consider.

Parameter synthesis problems have been studied for discrete-time Markovian models in [14,22]. These approaches apply to unbounded temporal properties and are based on constructing a rational function by performing state elimination [22].

Interval CTMCs, where transition rates are given as intervals, have been employed to obtain a three-valued abstraction for CTMCs [26]. In contrast to the parametric models we work with, the transition rates in interval CTMCs are chosen nondeterministically and schedulers are introduced to compute lower and upper probability bounds. Along the same lines, [41] introduce models and model checking algorithms for interval DTMCs and MDPs.

Synthesis approaches for non-stochastic biological models include the work by Batt et al. [6], where parameters of ODE-based gene regulatory network (GRN) models are synthesized using discrete abstractions and LTL model checking. In [5], the method of [18] based on sensitivity analysis is applied to the automated design of synthetic biological devices from basic ODE modules. In [28], parameter synthesis for discrete GRNs is reduced to coloured LTL model checking and solved through a distributed algorithm. Methods based on SMT are presented in [29,36] to synthesize Boolean network models from time-series and perturbation experiments data.

8 Conclusion and discussion

We have developed efficient algorithms for synthesising rate parameters for biochemical networks so that a given requirement, expressed as a time-bounded CSL formula, is guaranteed to be satisfied. The techniques are based on the computation of lower and upper probability bounds of [11], in conjunction with iterative refinement and sampling of parameters. In this work we focus on biological systems that, being characterised by complex and non-linear dynamics, are typically hard to analyse. However, our synthesis algorithms can be equivalently applied also to the performance and reliability analysis of computer systems.

We remark that improved performance can be easily achieved through parallel processing of individual subspaces and, within each subspace, of the parametric uniformisation method. Other techniques can also be integrated to speed up the synthesis process, including fast adaptive uniformisation [16,34], state aggregation [1,44], and abstraction [33]. Finally, we plan to include the synthesis algorithms in the `param` module of the `PRISM` model checker [14,32], and to extend the method to general non-linear rate functions.

Acknowledgements We thank David Šafránek for useful discussions about the stochastic models of gene regulation of mammalian cell cycle and Nicolas Basset for helping with the termination of the threshold synthesis algorithm. We also thank Andrej Tokarčík and Petr Pilař for implementing the prototype version of the synthesis algorithms. We finally thank the anonymous reviewers for their insightful feedback.

Appendix: Proofs

Proposition 2

Consider a parameter region $\mathcal{R} = [x_1^\perp, x_1^\top] \times \cdots \times [x_n^\perp, x_n^\top] \subseteq \mathcal{P}$. Fix an arbitrary state s and let the maximizing argument of the transient probability in s be (cf. Eq. 8):

$$p^* = \arg\max_{p \in \mathcal{R}} \pi_t(s) \tag{44}$$

and let $\tau_i^* = \pi_0 \mathbf{P}^i_{p^*}$. The local error introduced during a single discrete time step for state s, in the multi-affine case, is given by

$$e_i(s) = \frac{1}{q} \left| \text{flux}\left(\tau_{i-1}^*, s\right)(p^*) - \max_{p \in V_\mathcal{R}} \text{flux}\left(\tau_{i-1}^*, s\right)(p) \right|. \tag{45}$$

Note that in this analysis the local error is expressed for the discrete distribution $\tau_i^{\max}$, from which the solution $\pi_i^{\max}$ is derived as a linear combination. We now analyse the global error, which for $i > 1$ is given as:

$$g_i(s) = \left| \tau_i^*(s) - \tau_i^{\max}(s) \right| \tag{46}$$

$$= \left| \tau_{i-1}^*(s) + \frac{1}{q} \text{flux}\left(\tau_{i-1}^*, s\right)(p^*) - \tau_{i-1}^{\max}(s) - \frac{1}{q} \max_{p \in V_\mathcal{R}} \text{flux}\left(\tau_{i-1}^{\max}, s\right)(p) \right| \tag{47}$$

where we now use the definition of $g_i(s)$ and employ triangle inequality:

$$\leq g_{i-1}(s) + \frac{1}{q} \left| \text{flux}\left(\tau_{i-1}^*, s\right)(p^*) - \max_{p \in V_\mathcal{R}} \text{flux}\left(\tau_{i-1}^{\max}, s\right)(p) \right| \tag{48}$$

$$\leq g_{i-1}(s) + \frac{1}{q} \left| \text{flux}\left(\tau_{i-1}^*, s\right)(p^*) - \max_{p \in V_\mathcal{R}} \text{flux}\left(\tau_{i-1}^* + g_{i-1}, s\right)(p) \right| \tag{49}$$

where the overall global error g_i is the vector-wise equivalent of $g_i(s)$ and we continue

$$g_i(s) \leq g_{i-1}(s) + \frac{1}{q} \max_{p \in V_{\mathcal{R}}} \text{flux}(g_{i-1}, s)(p) \tag{50}$$

$$+ \frac{1}{q} \left| \text{flux}\left(\tau^*_{i-1}, s\right)(p^*) - \max_{p \in V_{\mathcal{R}}} \text{flux}\left(\tau^*_{i-1}, s\right)(p) \right| \tag{51}$$

$$\leq g_{i-1}(s) + \frac{1}{q} \max_{p \in V_{\mathcal{R}}} \text{flux}(g_{i-1}, s)(p) + e_i(s) \tag{52}$$

The form of $g_i(s)$ is understood as follows. The error in the current step is less than the error in the previous step plus the maximal local error plus the worst-case additional flux resulting from the approximation error in the previous step. Let the width of the parameter space $\mathcal{R}$ be given as $w_{\mathcal{R}} = \max_j (x_j^\top - x_j^\perp)$.

We now show how to derive bounds $M_1, M_2 < \infty$ such that

$$e_i \leq \frac{M_1}{q} w_{\mathcal{R}} \tag{53}$$

$$\max_{p \in V_{\mathcal{R}}} \text{flux}(g_i, s)(p) \leq M_2 \cdot \max_{s \in S} g_{i-1}(s) \tag{54}$$

Observe that, if the two inequalities above hold, we get the same over-approximation as in Eq. 42, which would prove the proposition true. For the local error we find

$$e_i(s) \leq \frac{1}{q} \max_{p \in V_{\mathcal{R}}} \left| \text{flux}\left(\tau^*_{i-1}, s\right)(p^*) - \text{flux}\left(\tau^*_{i-1}, s\right)(p) \right| \tag{55}$$

and in this case a Lipschitz constant $M_{1,s}$ exists such that

$$e_i(s) \leq \frac{M_{1,s}}{q} w_{\mathcal{R}} \tag{56}$$

Thus, $M_1 = \max_{s \in S} M_{1,s}$ is such that Eq. 53 holds. The maximum flux that propagates due to the approximation error in the previous step is given as

$$\max_{p \in V_{\mathcal{R}}} \text{flux}(g_{i-1}, s)(p). \tag{57}$$

Now regard $\text{flux}(g_{i-1}, s)(p)$ as a function of g_{i-1}, with p, s fixed. Note that the domain of the function is given by $\mathcal{R}' = \times_{s \in S}[0, g_{i-1}(s)]$, and thus $w_{\mathcal{R}'} = \max_{s \in S} g_{i-1}(s)$. Also, $\text{flux}(\mathbf{0}, s)(p) = 0$ for $\mathbf{0} : S \to 0$. Thus, another Lipschitz constant $M_{2,p,s}$ exists such that

$$\text{flux}(g_{i-1}, s)(p) = \text{flux}(g_{i-1}, s)(p) - \text{flux}(\mathbf{0}, s)(p) \leq M_{2,p,s} \cdot \max_{s \in S} g_{i-1}(s) \tag{58}$$

By taking $M_2 = \max_{p \in \mathcal{R}, s \in S} M_{2,p,s}$, Eq. 54 holds. Summarizing, the global error on $\tau_i^{\max}(s)$ is bounded by $\overline{g}_i$, under the assumption of multi-affine rate functions, as

$$\overline{g}_i = \begin{cases} 0 & \text{if } i = 0 \\ \overline{g}_{i-1} \cdot \left(1 + \frac{M_2}{q}\right) + \frac{M_1}{q} w_{\mathcal{R}} & \text{if } i > 0, \end{cases} \tag{59}$$

Theorem 1

The probability of satisfying an unnested and time-bounded CSL property given a CTMC $\mathcal{C}_p = (S, s_0, \mathbf{R}_p, L)$ reduces to the computation of transient-state probabilities over a CTMC

$\mathcal{C}'_p$ for which the transition relation $\mathbf{R}'_p$ is easily derived from the original CTMC [4,31]. Recalling Definition 4, the transient-state probability is given by standard uniformisation:

$$\hat{\pi}_{t,p} = \pi_0 \sum_{i=0}^{k_\epsilon} \gamma_{i,qt} \mathbf{P}^i_p. \tag{60}$$

Provided the entries in $\mathbf{P}$ are polynomials of finite degree, the expression $\hat{\pi}_t(s)$ is also a finite-degree polynomial over domain $\mathcal{P}$. We now prove that the approximate satisfaction function $\hat{\Lambda}_\phi$ of any time-bounded finitely-nested path formula can be expressed as a piecewise polynomial function with a finite number of subdomains, using the above as the base case in our induction. Note that only the path operators until (U) and next (X) allow nesting. We prove the induction step only for the until operator, since the proof for the next operator can be obtained in a similar way.

Let ϕ_1, ϕ_2 be time-bounded CSL path formulas such that $P_{=?}[\phi_1]$, $P_{=?}[\phi_2]$ are piecewise polynomial with a finite number of subdomains. Consider the nested formula:

$$P_{=?}\left[P_{\sim_1 r_1}[\phi_1]\ \mathrm{U}^I\ P_{\sim_2 r_2}[\phi_2]\right] \tag{61}$$

for a subdomain $I \in \mathbb{R}_{\geq 0}$, bounds r_1, r_2 and $\sim_i \in \{<, \leq, \geq, >\}$.

Observe that, if the satisfaction sets $v_i = \{s \in S \mid s \models P_{\sim_i r_i}[\phi_i]\}$ for $i = 1, 2$ are constant over a subspace $\mathcal{R} \subseteq \mathcal{P}$, then the expression in Eq. 61 is given by a polynomial function over $\mathcal{R}$, cf. Eq. 60. We will demonstrate that $\hat{\Lambda}_{P_{\sim_1 r_1}[\phi_1]\ \mathrm{U}^I\ P_{\sim_2 r_2}[\phi_2]}$ is piecewise polynomial over finitely many subdomains by constructing a partition of $\mathcal{P}$ that is conditioned on the truth assignment of each state. Given a state s, allow the partition $\mathcal{T}_1(s) \cup \mathcal{F}_1(s) = \mathcal{P}$ where

$$\forall p \in \mathcal{T}_1(s) : s \models P_{\sim_1 r_1}[\phi_1] \tag{62}$$

$$\forall p \in \mathcal{F}_1(s) : s \not\models P_{\sim_1 r_1}[\phi_1]. \tag{63}$$

By the induction hypothesis $P_{=?}[\phi_1] - r_1$ is piecewise polynomial over finite many subspaces, so that $\mathcal{T}_1(s)$, $\mathcal{F}_1(s)$ are unions of finitely many subspaces of $\mathcal{P}$. Assume a similar partition $\mathcal{T}_2(s) \cup \mathcal{F}_2(s) = \mathcal{P}$. We now wish to construct a partition $\bigcup_{v_1, v_2 \in 2^S} \mathcal{R}(v_1, v_2) = \mathcal{P}$ that is conditioned on the truth assignment of all states, so that $\forall p \in \mathcal{R}(v_1, v_2)$:

$$s \in v_1 \Leftrightarrow s \models P_{\sim_1 r_1}[\phi_1] \quad \wedge \quad s \in v_2 \Leftrightarrow s \models P_{\sim_2 r_2}[\phi_2] \tag{64}$$

in which case the expression of Eq. 61 is a polynomial function over $\mathcal{R}(v_1, v_2)$ because the truth-assignment of the nested formulas is fixed. We provide a constructive definition for $\mathcal{R}(v_1, v_2)$ by finite intersection of the sets $\mathcal{T}_i(s)$ and $\mathcal{F}_i(s)$, that is:

$$\mathcal{R}(v_1, v_2) = \left[\cap_{s \in v_1} \mathcal{T}_1(s)\right] \cap \left[\cap_{s \notin v_1} \mathcal{F}_1(s)\right] \cap \left[\cap_{s \in v_2} \mathcal{T}_2(s)\right] \cap \left[\cap_{s \notin v_2} \mathcal{F}_2(s)\right]. \tag{65}$$

Then $\mathcal{R}(v_1, v_2)$ is a union of a finite number of subdomains of $\mathcal{P}$.

Proposition 3

We first show termination of the algorithm for an unnested property $P_{\geq r}[\phi]$. Define

$$f = \hat{\Lambda}_\phi(\cdot)(s_0) - r$$

and let the zero-set of f be given as $Z(f) = \{p \in \mathcal{P} \mid f(p) = 0\}$. In other words, $Z(f)$ is the set of parameters p yielding satisfaction probability equal to r, i.e. $\hat{\Lambda}_\phi(p)(s_0) = r$. Now note that, at a generic step of the algorithm, the undecided space is composed by those regions $\mathcal{R}_i$ such that $\Lambda^{\mathcal{R}_i}_{\min} < r$ and $\Lambda^{\mathcal{R}_i}_{\max} \geq r$. Assuming infinite precision, $\Lambda^{\mathcal{R}_i}_{\min}$ and $\Lambda^{\mathcal{R}_i}_{\max}$ can be

made arbitrarily tight, i.e. the approximation error made arbitrarily small (cf Proposition 2). Therefore, any such $\mathcal{R}_i$ intersects the zero-set of f and the undecided space covers the zero-set. Formally, given a decomposition $\cup_i \mathcal{R}_i = \mathcal{P}$, the undecided region is given as:

$$\mathcal{U} = \cup_i \mathcal{R}_i \text{ s.t. } \mathcal{R}_i \cap Z(f) \neq \emptyset. \tag{66}$$

Excluding the trivial case that f is identically zero ($Z(f) = \mathcal{P}$), we prove termination in two steps. First, we show that $Z(f)$ can be covered by finitely many rectangular regions whose total volume can be made arbitrarily small. Call such cover C. Second, we show that our algorithm can reach in a finite number of steps a decomposition where $\mathcal{U}$ covers C and has volume no larger than the tolerance ε. Finally, we extend the termination proof to nested CSL properties.

1. We state now two useful properties of zero-sets: (i) the zero-set of a multi-variate polynomial is negligible, i.e. it has Lebesgue measure (volume) 0 (a proof is available in [12]). (ii) the zero-set $Z(f')$ of any continuous function f' is closed. This follows from the fact that $Z(f')$ is the pre-image of f' on the closed set $\{0\}$. Now note that the parameter space $\mathcal{P}$ is compact (bounded and closed). It follows that $Z(f) \subset \mathcal{P}$ is compact too. $Z(f)$ meeting condition i) corresponds to saying that for each $\epsilon' > 0$ there exists a finite or countable collection $R_1, R_2, \ldots$ of (possibly overlapping) open rectangles such that

$$Z(f) \subseteq \bigcup_{k \in K} R_k \text{ and } \sum_{k \in K} \text{vol}(R_k) < \epsilon' \text{ (see e.g. [8], Section 1).}$$

   Finally, by compactness every open cover of $Z(f)$ has finite a subcover, i.e. there exists a finite index set $K' \subseteq K$ such that:

$$Z(f) \subseteq \bigcup_{k' \in K'} R_{k'} \subseteq \bigcup_{k \in K} R_k.$$

   It follows that:

$$\text{vol}\left(\bigcup_{k' \in K'} R_{k'}\right) \leq \sum_{k' \in K'} \text{vol}(R_{k'}) \leq \sum_{k \in K} \text{vol}(R_k) < \epsilon'.$$

   The first inequality holds since rectangles in the cover can overlap. Thus, $C \stackrel{\text{def}}{=} \bigcup_{k' \in K'} R_{k'}$ is the required finite rectangular cover of $Z(f)$ with arbitrarily small volume. Note that these properties hold also if we replace in C each rectangle $R_{k'}$ with its closure.
2. Since any finite union of overlapping rectangles can be rewritten as a finite union of almost disjoint rectangles (i.e. intersecting only at their extrema) [15], we rewrite the cover C as

$$C = \bigcup_{j \in J} R_j, \text{ such that } \forall i, j \in J.\ \mathsf{int}(R_i) \cap \mathsf{int}(R_j) = \emptyset$$

   where $\mathsf{int}(R)$ is the interior of rectangle R. In particular, this transformation can be done in a such way that each R_j is a box of width δ, for some $\delta > 0$. We can hence derive the following:

$$|J| \cdot \delta^n = \sum_{j \in J} \text{vol}(R_j) = \text{vol}(C) < \epsilon' \tag{67}$$

   where n is the number of dimensions/parameters.

Without loss of generality assume that the parameter space $\mathcal{P}$ is the unit cube, meaning that the algorithm terminates for $\text{vol}(\mathcal{U}) \leq \varepsilon$. Assume also that at each step undecided parameter regions are decomposed by bisection. Consider a number of refinement steps i such that each undecided region at the i-th step has width $w \in [\delta, 2\delta]$, yielding $i = \lfloor -\log_2 \delta \rfloor$. From Eq. 66 and C being a cover of $Z(f)$, we derive that:

$$\mathcal{U} \subseteq \cup_i \mathcal{R}_i \text{ s.t. } \mathcal{R}_i \cap C \neq \emptyset.$$

Observe that each rectangle in the cover C is intersected by at most 2^n undecided regions of width w. Let $N = |J|$ be the number of boxes in the cover C. Then,

$$\text{vol}(\mathcal{U}) \leq 2^n \cdot N \cdot w^n \leq 2^n \cdot N \cdot (2\delta)^n < \epsilon' \cdot 4^n \tag{68}$$

where the last inequality holds by Eq. 67.
Since the bound ϵ' on the volume of C is arbitrary, termination follows. Indeed, to satisfy the termination condition ($\text{vol}(\mathcal{U}) \leq \varepsilon$), we can chose $\epsilon' = \varepsilon \cdot 4^{-n}$ which implies the existence of a suitable δ and, in turn, of a finite number of steps $i = \lfloor -\log_2 \delta \rfloor$.

3. By Theorem 1, we know that the satisfaction function of a nested CSL property is piecewise polynomial, with a finite number of (bounded) subdomains. We proceed by borrowing from steps (1) and (2) of this proof. Let D be an index set identifying the subdomains and f_d be the satisfaction function at the d-th subdomain. Then, there exist $\delta > 0$, arbitrary positive constants $\{\epsilon'_d\}_{d \in D}$ and a set of finite covers $\{C_d\}_{d \in D}$ such that for all $d \in D$, $\text{vol}(C_d) < \epsilon'_d$ and C_d is composed of pairwise-disjoint boxes of width δ. We can now prove termination by showing that Eq. 68 holds also for the nested case if we set $N = \sum_{d \in D} |C_d|$ to the total number of boxes and $\epsilon' = \sum_{d \in D} \epsilon'_d$.

However, there is an important caveat to discuss. In this case, the satisfaction function might exhibit jump discontinuities characterized by coarse probability bounds that cannot be "mitigated" by the iterative refinement procedure. It follows that we need to take into account all the additional undecided regions that contain jump discontinuity points. Fortunately, since piecewise continuous functions are continuous almost everywhere, the set of such discontinuities has measure 0, and hence, by a similar argument to the above proof, the total volume of the undecided regions containing discontinuities can be made arbitrarily small in a finite number of steps. □

Proposition 4

We prove the proposition by first showing that the safe bounds can be made arbitrarily small in a finite number of steps. Second, we derive the number of steps for which the termination condition is met.

1. Define $f = \hat{\Lambda}_\phi(\cdot)(s_0)$ and let D^k be the set of regions at the k-th decomposition step. Fix a region $\mathcal{R}_j^k \in D^k$. Let $[f(\mathcal{R}_j^k)]$ be the interval describing the range of f over $\mathcal{R}_j^k$. Since f is a polynomial function over a bounded domain, then it is also Lipschitz continuous, implying that there exists a constant m_j^k s.t. $\mathbf{w}([f(\mathcal{R}_j^k)]) \leq m_j^k \cdot \mathbf{w}(\mathcal{R}_j^k)$, where $\mathbf{w}$ denotes the width of the rectangle. The safe approximation we compute is such that $[f(\mathcal{R}_j^k)] \subset [\Lambda_{\min}^{\mathcal{R}_j^k}, \Lambda_{\max}^{\mathcal{R}_j^k}]$ and, in particular, $\Lambda_{\max}^{\mathcal{R}_j^k} - \Lambda_{\min}^{\mathcal{R}_j^k} = \mathbf{w}([f(\mathcal{R}_j^k)]) + e^\top + e^\perp$, where $e^\top$ and $e^\perp$ are the global approximation errors for the upper and lower bound of $\hat{\Lambda}_\phi$.
We now derive a closed-form expression for $e^\top$, based on Proposition 2. Let $e^\top = \overline{g}_{k_\epsilon}$, where k_ϵ is the number of uniformisation steps and, by Eq. 42, there exist positive

constants M_1 and M_2 such that

$$\overline{g}_{k_\epsilon} = \overline{g}_{k_\epsilon - 1} \cdot \left(1 + \frac{M_2}{q}\right) + \frac{M_1}{q}\left(\mathbf{w}\left(\mathcal{R}_j^k\right)\right). \tag{69}$$

Let $c_2 = 1 + \frac{M_2}{q}$ and $c_1 = \frac{M_1}{q}$. Solving the recurrence at Eq. 69, we get the following expression:

$$\overline{g}_{k_\epsilon} = c_j^{k,\top} \cdot \mathbf{w}(\mathcal{R}_j^k)) \tag{70}$$

where $c_j^{k,\top} = \dfrac{c_1 \cdot \left(c_2^{k_\epsilon} - 1\right)}{c_2 - 1}$. Following the same derivation for $e^\perp$, we conclude that there exist positive constants $c_j^{k,\top}$ and $c_j^{k,\perp}$ such that $e^\top = c_j^{k,\top} \cdot \mathbf{w}(\mathcal{R}_j^k)$ and $e^\perp = c_j^{k,\perp} \cdot \mathbf{w}(\mathcal{R}_j^k)$. Then, for all $\mathcal{R}_j^k \in D^k$:

$$\Lambda_{\max}^{\mathcal{R}_j^k} - \Lambda_{\min}^{\mathcal{R}_j^k} \leq \left(m + c^\top + c^\perp\right) \cdot \mathbf{w}\left(\mathcal{R}_j^k\right) \tag{71}$$

where $m = \max\{m_j^k \mid \mathcal{R}_j^k \in D_k\}$, $c^\top = \max\{c_j^{k,\top} \mid \mathcal{R}_j^k \in D_k\}$ and $c^\perp = \max\{c_j^{k,\perp} \mid \mathcal{R}_j^k \in D_k\}$. Since regions are decomposed by sectioning at the mid-point of each parameter interval, the width of a region is halved at every step, so the equation is expressed as:

$$\Lambda_{\max}^{\mathcal{R}_j^k} - \Lambda_{\min}^{\mathcal{R}_j^k} \leq \left(m + c^\top + c^\perp\right) \cdot \frac{\mathbf{w}(\mathcal{P})}{2^k}. \tag{72}$$

Then, for an arbitrary precision ϵ', there exists a finite number of decomposition steps k such that $\Lambda_{\max}^{\mathcal{R}_j^k} - \Lambda_{\min}^{\mathcal{R}_j^k} \leq \epsilon'$ for all $\mathcal{R}_j^k \in D^k$.

2. Now we show how to determine ϵ' s.t. the termination condition $\Lambda_\phi^\top - \Lambda_\phi^\perp \leq \epsilon$ is met. Let $\mathcal{R}_\top^k$ be one of the regions with the highest upper probability bound, thus $\Lambda_\phi^\top = \Lambda_{\max}^{\mathcal{R}_\top^k}$. Note that the highest lower bound M is at least $\Lambda_{\min}^{\mathcal{R}_\top^k}$, and thus every region $\mathcal{R}_j^k$ in $\mathcal{T}$ is such that

$$\Lambda_{\max}^{\mathcal{R}_j^k} \geq M \geq \Lambda_{\min}^{\mathcal{R}_\top^k} \geq \Lambda_{\max}^{\mathcal{R}_\top^k} - \epsilon' = \Lambda_\phi^\top - \epsilon'.$$

Then, the smallest lower bound in the true region, $\Lambda_\phi^\perp$, is at least $M - \epsilon'$ and so, $\Lambda_\phi^\perp \geq \Lambda_\phi^\top - 2 \cdot \epsilon'$. This implies that termination is achieved with $\epsilon' = \frac{\epsilon}{2}$ and, according to Eq. 72, in a number of steps equal to:

$$k = \left\lceil \log_2 \left(\frac{2 \cdot (m + c^\top + c^\perp) \cdot \mathbf{w}(\mathcal{P})}{\epsilon} \right) \right\rceil.$$

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.) Computer Aided Verification (CAV), LNCS, vol. 9206, pp. 195–213. Springer (2015)
2. Andreychenko, A., Mikeev, L., Spieler, D., Wolf, V.: Parameter identification for Markov models of biochemical reactions. In: Gopalakrishnan, G., Qadeer, S. (eds.) Computer Aided Verification (CAV), LNCS, pp. 83–98. Springer (2011)

3. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds.) Computer Aided Verification (CAV), LNCS, vol. 1102, pp. 269–276. Springer (1996)
4. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans Softw Eng **29**(6), 524–541 (2003)
5. Bartocci, E., Bortolussi, L., Nenzi, L.: A temporal logic approach to modular design of synthetic biological circuits. In: Gupta, A., Henzinger, T.A. (eds.) Computational Methods in Systems Biology (CMSB), pp. 164–177. Springer (2013)
6. Batt, G., Yordanov, B., Weiss, R., Belta, C.: Robustness analysis and tuning of synthetic gene networks. Bioinformatics **23**(18), 2415–2422 (2007)
7. Belta, C., Habets, L.: Controlling a class of nonlinear systems on rectangles. IEEE Trans Autom Control **51**(11), 1749–1759 (2006)
8. Billingsley, P.: Probability and Measure. Wiley, Hoboken (2008)
9. Bortolussi, L., Milios, D., Sanguinetti, G.: Smoothed model checking for uncertain continuous time markov chains. CoRR. arXiv:1402.1450 (2014)
10. Bortolussi, L., Sanguinetti, G.: Learning and designing stochastic processes from logical constraints. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) Quantitative Evaluation of Systems (QEST), LNCS, vol. 8054, pp. 89–105. Springer (2013)
11. Brim, L., Češka, M., Dražan, S., Šafránek, D.: Exploring parameter space of stochastic biochemical systems using quantitative model checking. In: Sharygina, N., Veith, H. (eds.) Computer Aided Verification (CAV), LNCS, vol. 8044, pp. 107–123. Springer (2013)
12. Caron, R., Traynor, T.: The zero set of a polynomial. Technical report, University of Windsor (2005)
13. Češka, M., Dannenberg, F., Kwiatkowska, M., Paoletti, N.: Precise parameter synthesis for stochastic biochemical systems. In: Mendes, P., Dada, J.O., Smallbone, K. (eds.) Computational Methods in Systems Biology (CMSB), pp. 86–98. Springer (2014)
14. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: Theoretical Aspects of Software Engineering (TASE), pp. 85–92. IEEE (2013)
15. Courant, R., John, F.: Introduction to Calculus and Analysis, vol. 2. Springer, Berlin (2012)
16. Dannenberg, F., Hahn, E.M., Kwiatkowska, M.: Computing cumulative rewards using fast adaptive uniformisation. In: Gupta, A., Henzinger, T.A. (eds.) Computational Methods in Systems Biology (CMSB), LNCS, vol. 8130, pp. 33–49. Springer (2013)
17. Dannenberg, F., Kwiatkowska, M., Thachuk, C., Turberfield, A.: DNA walker circuits: computational potential, design, and verification. Nat. Comput. **14**, 195–211 (2014)
18. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification (CAV), LNCS, vol. 6174, pp. 167–170. Springer (2010)
19. Fox, B.L., Glynn, P.W.: Computing Poisson probabilities. CACM **31**(4), 440–445 (1988)
20. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2381 (1977)
21. Grassmann, W.: Transient solutions in Markovian queueing systems. Comput. Oper. Res. **4**(1), 47–53 (1977)
22. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. Int. J. Softw. Tools Technol. Transf. (STTT) **13**(1), 3–19 (2011)
23. Han, T., Katoen, J., Mereacre, A.: Approximate parameter synthesis for probabilistic time-bounded reachability. In: Real-Time Systems Symposium (RTSS), pp. 173–182. IEEE (2008)
24. Jensen, A.: Markoff chains as an aid in the study of Markoff processes. Skand. Aktuarietidskr. **36**, 87–91 (1953)
25. Jha, S.K., Langmead, C.J.: Synthesis and infeasibility analysis for stochastic models of biochemical systems using statistical model checking and abstraction refinement. Theor. Comput. Sci. **412**(21), 2162–2187 (2011)
26. Katoen, J.P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time markov chains. In: Damm, W., Hermanns, H. (eds.) Computer Aided Verification (CAV), LNCS, vol. 4590, pp. 311–324. Springer (2007)
27. Kermack, W., McKendrick, A.: Contributions to the mathematical theory of epidemicsii. The problem of endemicity. Bull. Math. Biol. **53**(1), 57–87 (1991)
28. Klarner, H., Streck, A., Šafránek, D., Kolčák, J., Siebert, H.: Parameter identification and model ranking of thomas networks. In: Gilbert, D., Heiner, M. (eds.) Computational Methods in Systems Biology (CMSB), pp. 207–226. Springer (2012)
29. Koksal, A.S., Pu, Y., Srivastava, S., Bodik, R., Fisher, J., Piterman, N.: Synthesis of biological models from mutation experiments. SIGPLAN Not. **48**(1), 469–482 (2013)

30. Kwiatkowska, M., Norman, G., Pacheco, A.: Model checking expected time and expected reward formulae with random time bounds. Comput. Math. Appl. **51**, 305–316 (2006)
31. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) Formal Methods for Performance Evaluation (SFM), LNCS, vol. 4486, pp. 220–270. Springer (2007)
32. 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 (2011)
33. Madsen, C., Myers, C., Roehner, N., Winstead, C., Zhang, Z.: Utilizing stochastic model checking to analyze genetic circuits. In: Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp. 379–386. IEEE (2012)
34. Mateescu, M., Wolf, V., Didier, F., Henzinger, T.A.: Fast adaptive uniformization of the chemical master equation. IET Syst. Biol. **4**(6), 441–452 (2010)
35. Michaelis, L., Menten, M.L.: Die kinetik der invertinwirkung. Biochem. z **49**(333–369), 352 (1913)
36. Paoletti, N., Yordanov, B., Hamadi, Y., Wintersteiger, C.M., Kugler, H.: Analyzing and synthesizing genomic logic functions. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification (CAV), pp. 343–357. Springer (2014)
37. Rao, C.V., Arkin, A.P.: Stochastic chemical kinetics and the quasi-steady-state assumption: application to the gillespie algorithm. J. Chem. Phys. **118**(11), 4999–5010 (2003)
38. Reibman, A.: Numerical transient analysis of Markov models. Comput. Oper. Res. **15**(1), 19–36 (1988)
39. Sanft, K.R., Gillespie, D.T., Petzold, L.R.: Legitimacy of the stochastic Michaelis-Menten approximation. Syst. Biol., IET 5(1), 58–69 (2011)
40. Sassi, B., Amin, M., Girard, A.: Control of polynomial dynamical systems on rectangles. In: European Control Conference (ECC), pp. 658–663. IEEE (2013)
41. Sen, K., Viswanathan, M., Agha, G.: Model-checking markov chains in the presence of uncertainties. In: Hermanns, H., Palsberg, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS), LNCS, vol. 3920, pp. 394–410. Springer (2006)
42. Swat, M., Kel, A., Herzel, H.: Bifurcation analysis of the regulatory modules of the mammalian G1/S transition. Bioinformatics **20**(10), 1506–1511 (2004)
43. Wickham, S.F.J., Bath, J., Katsuda, Y., Endo, M., Hidaka, K., Sugiyama, H., Turberfield, A.J.: A DNA-based molecular motor that can navigate a network of tracks. Nat. Nanotechnol. **7**, 169–173 (2012)
44. Zhang, J., Watson, L.T., Cao, Y.: Adaptive aggregation method for the chemical master equation. Int. J. Comput. Biol. Drug Des. **2**(2), 134–148 (2009)

Approximating Complex Arithmetic Circuits with Formal Error Guarantees: 32-bit Multipliers Accomplished

Milan Češka, Jiří Matyáš, Vojtech Mrazek, Lukas Sekanina, Zdenek Vasicek and Tomas Vojnar
Faculty of Information Technology, Centre of Excellence IT4Innovations
Brno University of Technology, Brno, Czech Republic
{ceskam, imatyas, imrazek, sekanina, vasicek, vojnar}@fit.vutbr.cz

***Abstract*—We present a novel method allowing one to approximate complex arithmetic circuits with formal guarantees on the approximation error. The method integrates in a unique way formal techniques for approximate equivalence checking into a search-based circuit optimisation algorithm. The key idea of our approach is to employ a novel search strategy that drives the search towards promptly verifiable approximate circuits. The method was implemented within the ABC tool and extensively evaluated on functional approximation of multipliers (with up to 32-bit operands) and adders (with up to 128-bit operands). Within a few hours, we constructed a high-quality Pareto set of 32-bit multipliers providing trade-offs between the circuit error and size. This is for the first time when such complex approximate circuits with formal error guarantees have been derived, which demonstrates an outstanding performance and scalability of our approach compared with existing methods that have either been applied to the approximation of multipliers limited to 8-bit operands or statistical testing has been used only. Our approach thus significantly improves capabilities of the existing methods and paves a way towards an automated design process of provably-correct circuit approximations.**

***Index Terms*—approximate computing, logical synthesis, genetic programming, formal methods**

I. Introduction

As many important applications are inherently error resilient, precision of the involved computations can be traded for improved energy efficiency, performance, and/or chip area. Various approaches exploiting this fact have been developed in recent years and presented under the umbrella of the so-called *approximate computing* [1]. These approximations can be conducted at different system levels with circuit approximation being one of the most popular.

Circuit approximation techniques can be classified into two main groups: (1) *Frequency/voltage over-scaling* where timing-induced errors can appear as the circuit is operated on a higher frequency or lower voltage than the nominal value. (2) *Functional approximation* where the original circuit is replaced by a less complex one which exhibits some errors but improves non-functional circuit parameters such as power consumption or chip area. We only deal with the latter approach in this paper. Circuit approximation can be formulated as a multi-objective optimization problem where the error and non-functional circuit parameters are conflicting design objectives. Since the resulting approximate circuits are common circuits, they can be implemented using the standard circuit design flow.

We focus on *approximate arithmetic circuits* (AACs) because they are frequently used in key applications relevant for approximate computing. Prominent examples are signal, image, and video processing circuits (such as filters, discrete transforms, and motion estimation blocks [2]), or the multiply-accumulate-transform structures of artificial neurons in neural networks (consuming about 50% of the total power in neural network accelerators [3]).

Various *error metrics*, such as the worst-case relative error or the mean absolute error, for evaluating approximate circuits have been proposed (cf. Sect. III). A crucial question is then how the error of a given approximation is derived. For that, as discussed in more details in the related work section, methods based on *simulating* the circuit on given inputs are often used. However, such approaches suffer from low scalability (exhaustive simulation), lack of strong guarantees (when simulating the circuit for a random subset of the possible inputs only), and/or specialization to certain circuits only (statistical models). Alternatively, as in our case, the error can be derived using *formal verification*. The main advantages of this approach lie in that (1) formal error bounds can be given as a part of the input and (2) the approach is more scalable than exhaustive circuit simulation.

While formal methods of (exact) equivalence checking have been studied for decades, only a few formal approximate checking methods have been used in circuit approximation tools. Depending on the particular error metric, the error calculation is transformed to a decision problem and solved by means of SAT solving or binary decision diagrams (BDDs). Despite of enormous progress in the area of SAT solvers and BDD libraries, approximation of arithmetic circuits with formal error guarantees was so far limited to circuits no more complex than 16-bit adders and 8-bit multipliers [4], [5], [6].

In this paper, we present a new method for designing complex approximate arithmetic circuits with formal bounds on the approximation error. The method uniquely integrates new formal techniques for approximate equivalence checking into search-based circuit optimization by means of Cartesian genetic programming (CGP). The key idea is to employ a novel search strategy driving the search towards promptly verifiable approximate circuits. We have implemented the strategy within the ABC tool and extended the underlying equivalence checking algorithm to support queries on the worst-case error. This extension builds on a new effective construction of miters, i.e. auxiliary circuits interconnecting the original correct circuit and its approximation such that their approximate equivalence can be checked.

We decided to optimize for the *worst-case error* since its exact value can be important in time-critical and dependable

978-1-5386-3093-8/17/$31.00 ©2017 IEEE

systems (e.g., inverse kinematics in robot control [7]) or when complex approximate arithmetic circuits are constructed using less complex approximate building (circuit) blocks. The final error then depends on how the worst case error is propagated from low-level blocks to the result. Moreover, even in not so critical applications such as image processing, low average error but excessive worst-case error can produce unacceptable results [8]. Finally, our results suggest that there is also a high correlation between the worst-case error and the mean absolute error (Sect. V).

While our primary motivation is to automatically approximate complex multipliers, our method is directly applicable to other arithmetic circuits too. The method is capable of providing Pareto fronts showing high-quality compromises between the circuit error and non-functional circuit parameters. Results are presented for approximate multipliers (with up to 32-bit operands) and adders (with up to 128-bit operands) and compared with several approximate circuits available in literature. This is for the first time when such complex approximate arithmetic circuits with formally guaranteed error bounds have been presented.

Contributions: We propose a new miter construction allowing for efficient approximate equivalence checking tailored to search-based approximation of complex arithmetic circuits. We design a novel search strategy for synthesis of approximate circuits with formal error guarantees that integrates Cartesian genetic programming and the proposed approximate equivalence checking. Using a resource-limited verifier, the strategy drives the search towards promptly verifiable candidates and thus provides scalable approximation of complex circuits. We develop an implementation of the miter construction and the search strategy within the ABC tool and perform extensive experimental evaluation of our approach on large circuits including approximation of 128-bit adders and 32-bit multipliers. Within several hours, we are able to construct high-quality Pareto sets of 128-bit adders and 32-bit multipliers that represent the trade-offs between the circuit error and non-functional circuit parameters.

II. Related Work

This section presents a brief survey of the most important approaches developed for functional approximation of multipliers and adders. We restrict our attention to these two arithmetic operations because they represent the key components of more complex circuits and thus their approximation has been intensively studied. Moreover, multipliers—due to their complex structure—represent one of the most difficult arithmetic circuits from the perspective of both approximation as well as verification.

A. Approximation Methods

The approximation process usually starts with a fully functional circuit and a target error. *Circuit-dependent approximation methods* then take the structure of the arithmetic circuit at the input and (manually or algorithmically) introduce modifications to carefully preselected parts of the circuit. In the case of adders, it is possible to approximate elementary 1-bit adders, modify the carry propagation chain, or introduce segments of adders and generate the carry using different methods [9]. In the case of multipliers, generation of partial products, the summation tree, counters, or compressors are approximated [10]. In addition to that, the simple bit-width reduction belongs to this category of methods too.

More complex approximate circuits can be constructed by a smart *composition* of approximate elementary blocks. For example, a 2-bit multiplier was approximated in [11] and then used as a building block of more complex multipliers. This strategy can be improved, e.g., by configurable lossy compression of the partial product rows based on their progressive bit significance [12].

The concept of *quality configurable circuits* uses elementary circuits composed in such a way that their error can be modified online using several configuration bits in order to dynamically reduce the power consumption. The configuration bits can (dis)connect some preselected parts of the circuit. As the source codes of quality configurable adders [13] and multipliers [2] are available online, we compare them with approximate circuits obtained using our approach.

General-purpose methods, such as SALSA [14] or SASIMI [15], aim at automatically approximating circuits independently of their structure. These methods operate with different circuit representations and employ various heuristics to identify circuit parts suitable for approximation. Evolutionary algorithms have been recently applied to accomplish desired approximations in a holistic scenario [16], [17]. A comprehensive library of 8-bit adders and multipliers was built using multi-objective CGP [18].

B. Simulation-Based Error Computation

Conceptually, the simplest approach to obtain precise error bounds of an AAC is to simulate its function on all possible inputs. However, even on state-of-the-art computer architectures, this approach has principal scalability limitations causing that it cannot be used to synthesize approximate circuits with more than 12-bit operands [19].

Due to that, the error is commonly estimated using a subset of input vectors only, e.g. 10^8 inputs were used to evaluate 16-bit adders in [9]. Of course, the main drawback of this approach is that no formal guarantees on the error bound can be provided. Alternatively, the circuit error can be calculated using a statistical model constructed for elementary circuit components and their compositions [20], [21]. However, reliable and general statistical models can only be constructed in some specific situations.

C. Formal Error Computation

Recently, various applications of formal methods have been intensively studied in order to improve the scalability of the design process of correct as well as approximate circuits. For designing correct circuits (where one insists on preserving the original functionality but tries to optimize non-functional parameters), one can consider combinational

equivalence checking based on modern SAT solvers, efficient BDD representations of circuits, or algebraic computation techniques combining polynomial representation of circuits with logic reductions [22], [23]. For designing AACs, a more challenging notion of *relaxed* or *approximate equivalence checking* is needed. This notion requires to quantify the approximation error or, alternatively, prove whether the error is below a certain threshold.

To quantify the approximation error using formal verification techniques, a use of auxiliary circuits, called *miters*, combining the original circuit and the approximate circuit was proposed in [24]. In order to check whether a predefined worst-case error is violated by the candidate approximate circuit, a pseudo-Boolean SAT solver combining a SAT solver with integer linear programming was then employed.

The number of inputs for which an approximate circuit returns an incorrect result can be quantified using *SAT counting methods* (so-called #SAT solvers). However, despite the recent progress in the area of #SAT solvers (see, e.g., [25]), our preliminary experiments indicate that #SAT problems encoding the error quantification are currently beyond the capabilities of state-of-the-art #SAT tools even for 12-bit multipliers.

An efficient BDD-based approach allowing one to guarantee the worst-case and the average-case arithmetic error of approximate adders up to 16-bit operands was proposed in [5]. An alternative approach that uses BDDs representing characteristic functions was employed in [4]. Compared to our approach, this approximation method lags behind in scalability, which is demonstrated by the fact that it has been applied to the approximation of multipliers limited to 8-bit operands and adders limited to 16-bit operands only.

III. Error Metrics for AACs

Various metrics describing the error of AACs have been proposed and shown suitable for different application domains. The most popular error metrics relevant especially to arithmetic circuits are the *worst-case absolute error* (WCAE) and the *mean absolute error* (MAE). For a correct circuit G, further denoted as the *golden circuit*, which computes a function f_G, and its approximation C, computing a function f_C, where $f_G, f_C : \{0,1\}^n \to \{0,1\}^m$, these metrics, relativized by the range of the output, are defined as follows:

$$\text{WCAE}(G,C) = \frac{\max_{x\in\{0,1\}^n} |\text{int}(f_G(x)) - \text{int}(f_C(x))|}{2^m},$$

$$\text{MAE}(G,C) = \frac{\sum_{x\in\{0,1\}^n} |\text{int}(f_G(x)) - \text{int}(f_C(x))|}{2^m},$$

where $\text{int}(x)$ denotes the integer representation of a bit vector x and $|i|$ denotes the absolute value of an integer i.

A. Checking Worst Case Errors

To compute whether the WCAE is violated, we can adopt the concept of approximation miter introduced in [24]. The general configuration of the approximation miter is shown in Fig. 1. The miter consists of the inspected approximate circuit C, the golden circuit G which serves as the specification, a subtractor, and a comparator which checks whether the error introduced by the approximation is greater than a given threshold $\mathcal{T}$. The output of the miter is a single bit which evaluates to 1 if and only if the error is violated, i.e. $\text{WCAE}(G,C) > \mathcal{T}$.

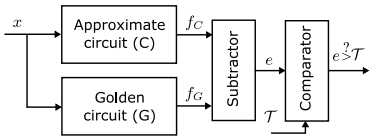


Fig. 1. Approximation miter for the worst-case error analysis, typically $e(x) = |f_G(x) - f_C(x)|$.

For a given input vector x, the subtractor calculates the difference between the output of the golden circuit, i.e. $f_G(x)$, and the output of the approximate circuit, i.e. $f_C(x)$. Let $d = \text{int}(f_G(x)) - \text{int}(f_C(x))$ be the error magnitude. A direct computation of the WCAE according to its definition leads to evaluating the expression $e = |d|$, i.e. the absolute difference of the error magnitude. The absolute difference is typically calculated by means of a common two's complement subtractor (implemented using m full-adders with the first carry-in set to 1 and inverting each bit of the subtrahend) followed by a circuit determining the absolute value (computed using m half-adders and m XOR gates).

B. The Proposed Miter Construction

Miters used in the literature compute the absolute value of the difference between f_G and f_C. The computation is usually performed in two steps. Firstly, a subtractor with a signed output evaluates $f_G - f_C$. Secondly, the absolute value has to be computed. The circuit performing such a task contains XOR chains which are a known cause of poor performance of the state-of-the-art SAT solvers [26]. The main reasons are that unlike AND/OR gates, the Boolean constraint propagation over XOR gates is limited, and the XOR operations cause the CNF form of the formulae to grow rapidly.

In order to avoid long XOR chains at the output of the miter which slowdown the decision process, we propose to employ a different approach. The key idea is to compare the result of the subtractor with both the positive and negative value of the threshold and thus avoid the expensive evaluation of the absolute value. For a given threshold $\mathcal{T}$ on the worst-case absolute error WCAE, it holds that $e > \mathcal{T}$ is satisfied iff d is positive and $d > \mathcal{T}$, or d is negative and $-d > \mathcal{T}$. As we typically deal with numbers in the two's complement, the second condition is equal to $\neg d > (\mathcal{T} - 1)$. Hence, we can use the two's complement representation and examine the positive and negative values separately to avoid usage of the absolute difference of the output.

Since the threshold $\mathcal{T}$ is fixed during the design process, we can easily avoid the standard comparator consisting of a long chain of XOR gates. This helps us to further simplify the miter and improve the performance of the decision procedure.

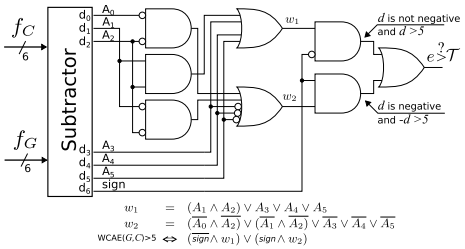


Fig. 2. The proposed approximation miter for the worst-case error analysis: an example for $\mathcal{T} = 5$, $N = 6$.

In particular, we replace the sequential comparison of the particular bits of the operands implemented as

$$A > B \equiv \bigvee_{0 \le i \le N-1} \left(A_i \wedge \neg B_i \bigwedge_{i < j \le N-1} \overline{A_j \oplus B_j} \right),$$

for B being a constant bit vector representing the threshold $\mathcal{T}$, by a simpler procedure implemented as

$$A > B \equiv \bigvee_{0 \le i \le N-1 \ \wedge \ B_i = 0} \left(A_i \bigwedge_{i < j \le N-1 \ \wedge \ B_j = 1} A_j \right).$$

As is evident, the resulting formula does not contain any XOR gate. Note that d is represented as an $m+1$ bit number in the two's complement—hence, A corresponds to the N least significant bits of d where $N = m$. The $(m+1)$-th bit is reserved for the sign and employed for determining whether d encodes a positive or negative number. The miter for $\mathcal{T} = 5$, f_C and f_G with 6-bit outputs is illustrated in Fig. 2.

The proposed construction, compared to the construction using the absolute value and full comparators, allows us to obtain smaller and structurally less complex miters. Such miters can be efficiently used in the SAT-based CEC procedures, resulting in a significant acceleration of the candidate circuit evaluation. Our experiments show that, in the case of arithmetic circuits having 64 output bits (e.g. 32-bit multipliers), the proposed construction improves the size of the miters (in terms of the number of And-Inverter Graph (AIG) nodes representing the circuit) by about 25–35% depending on the value of $\mathcal{T}$, where $\mathcal{T}$ ranged from 0.0001% to 0.5% of the maximal value at the output (i.e. 2^{64}) in our experiment.

IV. Search-Based Design of AACs

In this section, we present our novel approach to the search-based design of AACs combining principles of CGP with a verifiability-driven search strategy that employs a fitness function based on the approximate equivalence checking.

A. Cartesian Genetic Programming

CGP is a form of genetic programming where the candidate solutions are represented as a string of integers of a fixed length that is mapped to a directed acyclic graph [27]. This integer representation is called a *chromosome*. CGP can efficiently represent common computational structures including

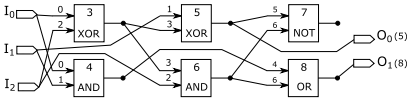


Fig. 3. Full adder represented by CGP. Chromosome: (0, 2, 2) (0, 1, 0) (1, 3, 2) (3, 2, 0) (5, 6, 3) (4, 6, 1) (5, 8), node functions: AND (0), OR (1), XOR (2), NOT (3).

mathematical equations, computer programs, neural networks, and digital circuits. The candidate circuits are typically represented in a two-dimensional array of programmable two-input nodes. Every node is encoded by three integers in the chromosome representation where the first two numbers denote the node's inputs, the third represents the node's function (see the illustration in Fig. 3).

In circuit approximation, the evolution loop starts with a *parent* representing a correctly working circuit. New candidate circuits are obtained from the parent using a *mutation operator* which performs random changes in the candidate's chromosome in order to obtain a new, possibly better candidate solution. In the next step, the algorithm evaluates the quality of each solution using a specified metric, called the *fitness function*. This function assesses important correctness and performance aspects of circuits. The candidate with the best fitness value is chosen as the parent of the next generation, the other solutions are removed and the evolution continues with generating new candidate circuits. The whole loop is repeated until a termination criterion is met. For details of CGP, see [27].

The most critical and time consuming part of the CGP loop is the fitness evaluation, which principally limits the scalability of the search-based design. To alleviate this problem, we propose below a novel search strategy.

B. Verifiability-Driven Search Strategy

The verifiability-driven search strategy can be seen as a general concept improving the scalability of evolutionary design methods. We demonstrate its key idea on the below problem.

Problem: *For a given golden circuit G and a threshold $\mathcal{T}$, our goal is to find a circuit C^* with the minimal size such that the error WCAE$(G, C^*) \le \mathcal{T}$.*

This problem formulation allows us to define the fitness function f in the following way:

$$f(C) = \begin{cases} \text{size}(C) & \text{if WCAE}(G, C) \le \mathcal{T}, \\ \infty & \text{otherwise} \end{cases}$$

where $\text{size}(C)$ denotes the size of the circuit C. Since the procedure deciding whether $\text{WCAE}(G, C) \le \mathcal{T}$ (further denoted as SAT solver) represents the most time consuming part of the design loop, we avoid calling the procedure as much as possible. Therefore, we only call SAT solver for circuits C satisfying $\text{size}(C) < \text{size}(B)$ where B is the best solution with an acceptable error (i.e., $\text{WCAE}(G, B) \le \mathcal{T}$) that we have found so far. Our experiments show that, during the evolution process, a significant set of candidate designs C does not satisfy the condition $\text{size}(C) < \text{size}(B)$ and thus their fitness can be easily assessed without SAT solver.

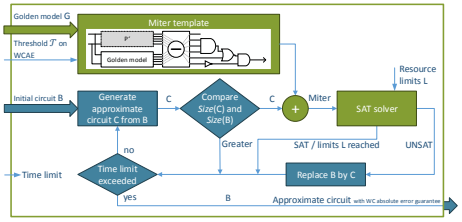


Fig. 4. The main steps of the proposed verifiability-driven search scheme.

Our experiments further indicate that a long sequence of candidate circuits B_i improving the size and having an acceptable error has to be typically explored to obtain a solution that is sufficiently close to C^*. Therefore, both the SAT and the UNSAT queries to SAT solver have to be short. To this end, we use an additional criterion for the evaluation of the circuit C, namely, the ability of SAT solver to prove that $\text{WCAE}(G, C) \leq \mathcal{T}$ with a given limit L on the resources available to the underlying decision procedure. If the procedure fails to prove $\text{WCAE}(G, C) \leq \mathcal{T}$ within the limit L, we set $f(C) = \infty$ and generate a new candidate. The design loop using the verifiability-driven search is illustrated in Fig. 4.

The inputs of the design process include: (1) the golden model G, (2) the threshold on the worst case absolute error $\mathcal{T}$, (3) the initial circuit B having an acceptable error (it can be either the golden model or a suitable approximation we want to start with), and (4) the time limit on the overall design process. The loop exploits the CGP principles; namely, it uses mutations to generate new candidate circuits C from the candidate circuit B representing the best approximation of the circuit C^* that we have found so far. The circuit C is then evaluated using the fitness function f as described above. If the candidate C belongs to the improving sequence (i.e., $\text{size}(C) < \text{size}(B)$ and $\text{WCAE}(G, C) \leq \mathcal{T}$), we replace B by C. The design loop terminates if the time limit is reached and B is returned as the output of the design process.

In our verifiability-driven search scheme, we use the resource limit L (as a parameter of the design loop) to drive the search towards candidates that can be promptly evaluated. We intentionally throw away improving candidates B_i that require greater resources and thus longer, but still feasible, verification time. The reason for this is the fact that by mutating these candidates we would most likely obtain solutions that would require the same or even longer verification times and thus finding the whole improving sequence would become time-infeasible. Instead, we require that every improving candidate B_i has to be verifiable using the resource limit L and thus drive the search towards candidates B_i that, for a given time limit on the overall design process, lead to longer improving sequences. Our experiments indicate that these sequences lead to candidate circuits that are closer to C^*. Since we are able to evaluate a much larger set of candidate circuits, we have a better chance to find a long improving sequence within the given time provided that it exists for the limit L.

The obvious disadvantage is that we possibly cut improving sequences that would lead to good solutions within the given design time. It can also happen that, for the limit L, no improving sequence exists, while it exists for a slightly greater resource limit. Despite of this limitation, our results clearly show that the proposed verifiability-driven search strategy allows us to utilise the given design time in a more efficient way compared to the standard evolution schemes.

C. Integration to the ABC Tool

The proposed approach performs the approximation at the level of the CGP problem representation (i.e., on acyclic oriented graphs with arbitrary two-input logic functions in the nodes). The green part of Fig. 4 shows the position of ABC in our methodology. ABC is primarily used to construct the miter and decide whether the maximal arithmetic error of the candidate circuit is not above $\mathcal{T}$. The proposed miter construction allows us to reduce the problem of approximate equivalence checking to the Boolean satisfiability (SAT) problem. In order to evaluate a candidate circuit, (1) a candidate chromosome is used to construct a corresponding AIG, (2) another AIG, representing the golden circuit, is constructed (just once at the beginning of the evolution), and (3) the miter is built. The state-of-the-art techniques used for CEC in the ABC tool—the `iprove` engine—are then applied to decide the equivalence. An important feature of the mix of techniques used in `iprove` is that one can control the time needed for one query, which is the key feature we exploit in our verifiability-driven search strategy. In particular, the satisfiability checking can be controlled by fine-tuning various resource limits for the different techniques used, such as the number of simulations performed to prove non-equivalence, the number of conflicts in structural hashing, or the number of logic-reduction steps. We so far used solely a limit on the maximal number of conflicts in which a single variable (representing an AIG node) can be involved during the backtracking process. Our experiments show that this resource limit allows us to effectively control the time needed for particular `iprove` queries and thus to drive the search towards promptly verifiable circuits.

A similar approach has recently been used in circuit approximation exploiting the approximate-aware rewriting of an AIG representation of circuits [4]. Principally, our approach differs in the candidate circuit representation (the gate-level CGP encoding), its evaluation, and in using the verifiability-driven evolution instead of a simple greedy algorithm for AIG pruning. The gate-level representation is an important feature of our approach which allows us to efficiently capture XOR-intensive structures existing in arithmetic circuits.

V. Results

To evaluate the proposed method, we primarily focused on complex approximate multipliers as they are the most challenging benchmark problems. Since only 8-bit multipliers with guaranteed error bounds were presented in the literature so far, there are no solutions available for a direct comparison in the case of 16-bit and more complex approximate multipliers.

Hence, (1) we compare the 16-bit approximate multipliers that we generated using our method with 16-bit multipliers (available in the literature) whose error was determined using simulation, and then (2) we present Pareto fronts (the error and key circuit parameters) for 20-bit, 24-bit, 28-bit, and 32-bit approximate multipliers and up to 128-bit approximate adders to demonstrate the scalability of the proposed method.

A. Experimental Setup

We implemented our approach, including the miter construction and verifiability-driven evolution, within the ABC tool [28]. Array multipliers and ripple carry adders composed of 2-input gates were employed as the initial (golden) circuits for CGP. The number of nodes in the CGP's grid is equal to the number of gates of the initial circuit. The set of functions consists of the common two-input logic gates, the buffer, and the inverter. We used 2 circuits in the population and 5 integers were modified by the mutation operator.

For each target WCAE, we performed 30 independent runs of CGP to obtain statistically significant results. Each CGP was executed for 2 hours on an Intel Xeon X5670 2.4 GHz processor using a single core. The individual CGP runs are independent and thus we executed them in parallel using a cluster of these processors to accelerate the design process.

For purposes of the fitness evaluation, the circuit size is estimated as the sum of the relative area of the two-input gates used, where the sizes of each gate are taken from the technology library. At the end of the evolution, the 5 most fitting circuits for each WCAE were synthetized using the Synopsys Design Compiler (high-effort compiling for a better quality of the results) for a 45 nm technology library in order to obtain non-functional parameters like the area and power-delay product (PDP). The accurate implementations were created by means of Verilog $*$ and $+$ operators and synthesized in the same way as approximate circuits.

B. 16-bit Approximate Multipliers

An evaluation of the verifiability-driven search: In the first experiment, we approximated the golden 16-bit multiplier for 9 target values of WCAE from the set $\{0.1, 0.2, 0.5, 1, 2, 5, 10, 15$ and $20\%\}$ and evaluated the proposed method with three different settings of the resource limit L controlling the maximal number of conflicts for one AIG node: (1) no limits, i.e., L=∞, (2) L=$160K$, and (3) L=$20K$. The limits L=160K and L=20K roughly correspond to the time limit of 120 sec. and 3 sec., respectively, on 16-bit multipliers.

Fig. 5 shows that, for WCAE $\geq 2\%$, the resource limit L has a marginal impact on the PDP and area. However, with a decreasing target WCAE, the limit L=20K provides significantly better results. For example, if WCAE = 0.1% and L=20K, 22,050 SAT calls were produced and 11% of them were terminated on average because of the termination condition. In the case of L=160K, 856 SAT calls were produced only (15% terminated). The average number of SAT calls (across all target errors) that were forced to terminate is 6.28% (for L=160K) and 8.84% (for L=20K). If L=∞, 170 SAT calls were evaluated for WCAE = 0.1% only. Despite the fact that some potentially good candidate circuits are quickly rejected, the aggressive resource limits allowed us to generate and evaluate significantly more candidate circuits and thus to substantially improve the quality of results. Box plots in Fig. 5 also show that independent runs with L=20K lead to circuits having very similar parameters (low inter-quartile distances) and thus this limit is be used in the following experiments.

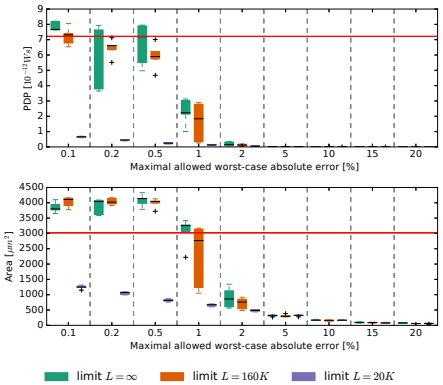


Fig. 5. PDP and area of approximate 16-bit multipliers for 9 target errors obtained using 3 different resource limits L on the SAT solver. The red line shows the PDP and area of the accurate multiplier.

Note that the parameters of some approximate multipliers shown in Fig. 5 are worse than for the accurate multiplier. The reason is that the relative area is the only non-functional circuit parameter optimized by CGP while the PDP and area are computed at the end of the optimization using the Synopsys Design Compiler. We have never observed this discrepancy for the limit L=20K.

A comparison with other multipliers: Next, we generated 16-bit approximate multipliers using the setup described in the previous section and compared them with approximate multipliers available in the literature. In order to perform a fair comparison (the error of the published multipliers was originally estimated using simulation), we modified our method and applied a binary search strategy to determine the WCAE exactly. In addition to WCAE, we also provide MAE obtained using simulation (10^9 vectors).

We considered the following 16-bit approximate multipliers:

M1 Approximate configurable multipliers from the lpACLib library [13], where the multiplication is recursively simplified using two different variants (denoted as *Lit* and *V1*) of an elementary block representing a 2-bit multiplier. The partial results are summed using accurate adders. We implemented 32 different architectures consisting of four 8-bit multipliers where each of these multipliers is configurable as exact/approximate (2^4 configurations) and can be built using either *Lit* (M1Lit) or *V1* (M1V1) blocks.

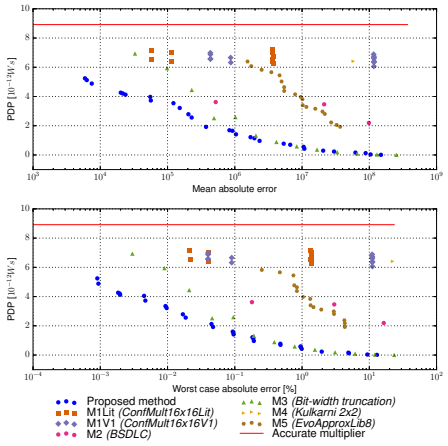


Fig. 6. Parameters of 16-bit approximate multipliers considered in our study.

M2 The approximate multiplier employing the bit-significance-driven logic compression as introduced in [12].

M3 Approximate multipliers obtained from exact multipliers using the bit-width reduction. The reduction replaces 16-bit multipliers by accurate x-bit multipliers (for $x < 16$). It ignores the LSBs of the operands and leaves the LSBs of the result zero.

M4 The approximate multiplier composed of approximate 2-bit multipliers as proposed in [11].

M5 Approximate multipliers composed of 8-bit multipliers that are available in the EvoApproxLib library [18]. The construction principle is taken from [11].

For all considered multipliers, the value of PDP is plotted against WCAE and MAE in Fig. 6 (only Pareto fronts are visualized). While the bit-width reduction provides the same quality of results as our method for large target errors (up to 20% WCAE), it is significantly outperformed by our approach for small target errors. Despite that the existing approximate multipliers typically exhibit good tradeoffs between the error and PDP in specific applications (as demonstrated in the relevant literature), Fig. 6 clearly shows that these multipliers are considerably Pareto-dominated by the multipliers obtained using our approach. These results were, in fact, expected as the proposed method is based on a global holistic optimization approach while the other approximate multipliers were composed of smaller ones and the composition procedure always introduces some overhead. Finally, it is an interesting observation that MAE follows the trend of WCAE. It seems that WCAE can be used as a good indicator of MAE.

C. Complex Multipliers

The aim of our further experiments is to show that the proposed method is scalable and can approximate complex multipliers. We present the results of the approximation process on 12-bit, 16-bit, 20-bit, 24-bit, 28-bit, and 32-bit multipliers. The target WCAEs were adapted accordingly to respect the range of values in the different considered bit widths. We used the same setup as in the previous sections but increased the time of optimization to 4 hours for the 24-bit multiplier and 6 hours for larger multipliers. The reason is that the search space becomes much bigger. While the exact 12-bit multiplier contains 850 two-input gates, the 32-bit exact multiplier requires over 6,300 gates. We obtained (as the result of evolution) over 1190 unique multipliers. Because of this huge number and for the sake of clarity, Fig. 7 shows parameters of approximate multipliers occupying the Pareto fronts only.

In the experiments, we observed that, in the case of 12-bit multipliers, 2.4% of SAT calls were terminated on average due to the resource limit L=20K only. However, this number increased to 36.9% in the case of approximate 32-bit multipliers. For all bit widths, the MAE is around 30% of the worst-case error, which again demonstrates that WCAE is a good indicator of MAE. Fig. 7 also shows that the obtained approximations cover the whole range (up to 100%) of the Area axis. However, this is not the case for PDP. The reason is that we optimize the relative area and PDP is computed after the synthesis.

Since Pareto fronts shown in Fig. 7 follow the trend of the highly competitive fronts for the 16-bit multipliers presented before, we believe that the tradeoffs between the circuit error and size obtained for more complex multipliers are also very good and thus the corresponding circuits represent the cutting edge of approximate multipliers and can serve as a new benchmark set for approximate computing.

D. Approximate Adders

In order to demonstrate that the proposed method is applicable for other complex arithmetic circuits, we constructed Pareto fronts for approximate adders with 20-bit to 128-bit operands. Approximation of adders is much easier than approximation of multipliers since adders are structurally less complicated and the number of outputs is lower. For example, the exact 20-bit adder requires 140 two-input gates and the 128-bit adder consists of 1,000 gates.

The approximate adders were constructed using the same setup as in the previous section. A single CGP run took 2 hours (for all bit widths). Fig. 8 shows parameters of approximate adders occupying the corresponding Pareto fronts. We report 16 to 18 non-dominated implementations of 24-bit, 28-bit, and 32-bit adders in terms of PDP and WCAE. For 64-bit and 128-bit adders, 12 tradeoffs are reported only because we have restricted the number of target error levels. Similarly to the evolved multipliers, the proposed approximate adders are also good candidates for including into a new benchmark suite.

VI. Conclusion

Automated design of approximate circuits with formal error guarantees is a landmark of provably-correct construction of energy-efficient systems. We present a solution to this problem, introducing a novel verifiability-driven search strategy

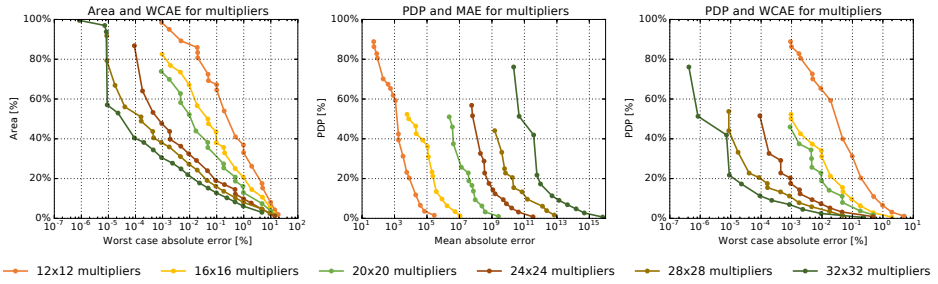


Fig. 7. Pareto fronts showing parameters of evolved approximate multipliers. 100% refers to parameters of the accurate multiplier for a given bit width.

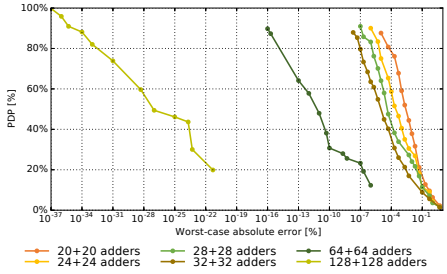


Fig. 8. Pareto fronts showing parameters of evolved approximate adders. 100% refers to parameters of the accurate adder for a given bit width.

that uniquely integrates approximate equivalence checking into a search-based circuit optimisation algorithm. Able to construct high-quality Pareto sets of 32-bit multipliers and 128-bit adders, our method shows excellent scalability and paves the way for design automation of complex approximate circuits.

In the future, we will thoroughly explore relationships between resource limits on the underlying SAT solvers and the structure of the resulting circuits. This will allow us to further improve the performance of our method and thus to go beyond the approximation of 32-bit multipliers. We will also integrate the constructed circuits into real-world energy-aware systems to demonstrate practical impacts of our work.

Acknowledgments: This work has been supported by the Czech Science Foundation grant No. GA16-17538S.

References

[1] S. Mittal, "A survey of techniques for approximate computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, 2016.
[2] M. Shafique, R. Hafiz *et al.*, "Invited: Cross-layer approximate computing: From logic to architectures," in *Proc. of DAC'16*, 2016, pp. 1–6.
[3] P. Judd, J. Albericio *et al.*, "Proteus: Exploiting numerical precision variability in deep neural networks," in *ICS'16*, 2016, pp. 1–12.
[4] A. Chandrasekharan, M. Soeken *et al.*, "Approximation-aware rewriting of AIGs for error tolerant applications," in *Proc. of ICCAD'16*, 2016, pp. 83:1–83:8.
[5] Z. Vasicek, V. Mrazek, and L. Sekanina, "Towards low power approximate DCT architecture for HEVC standard," in *Proc. of DATE'17*, 2017, pp. 1576–1581.
[6] C. Yu and M. Ciesielski, "Analyzing imprecise adders using BDDs – a case study," in *Proc. of ISVLSI'16*, 2016, pp. 152–157.
[7] B. Grigorian and G. Reinman, "Dynamically adaptive and reliable approximate computing using light-weight error analysis," in *Proc. of AHS'14*, 2014, pp. 248–255.
[8] D. S. Khudia, B. Zamirai *et al.*, "Rumba: An online quality management system for approximate computing," in *ISCA'15*, 2015, pp. 554–566.
[9] H. Jiang, J. Han, and F. Lombardi, "A comparative review and evaluation of approximate adders," in *Proc. of GLVLSI'15*, 2015, pp. 343–348.
[10] H. Jiang, C. Liu *et al.*, "A comparative evaluation of approximate multipliers," in *Int. Symp. Nanoscale Architectures*, 2016, pp. 191–196.
[11] P. Kulkarni, P. Gupta, and M. D. Ercegovac, "Trading accuracy for power in a multiplier architecture," *J. Low Power Electronics*, vol. 7, no. 4, pp. 490–501, 2011.
[12] I. Qiqieh, R. Shafik *et al.*, "Energy-efficient approximate multiplier design using bit significance-driven logic compression," in *Proc. of DATE'17*, 2017, pp. 7–12.
[13] M. Shafique, W. Ahmad *et al.*, "A low latency generic accuracy configurable adder," in *Proc. of DAC'15*, 2015, pp. 86:1–86:6.
[14] S. Venkataramani, A. Sabne *et al.*, "SALSA: systematic logic synthesis of approximate circuits," in *Proc. of DAC'12*, 2012, pp. 796–801.
[15] S. Venkataramani, K. Roy, and A. Raghunathan, "Substitute-and-simplify: a unified design paradigm for approximate and quality configurable circuits," in *Proc. of DATE'13*, 2013, pp. 1367–1372.
[16] Z. Vasicek and L. Sekanina, "Evolutionary approach to approximate digital circuits design," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 432–444, 2015.
[17] K. Nepal, S. Hashemi *et al.*, "Automated high-level generation of low-power approximate computing circuits," *IEEE Trans. Emerg. Topics Comput.*, pp. 1–13, 2017.
[18] V. Mrazek, R. Hrbacek *et al.*, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. of DATE'17*, 2017, pp. 258–261.
[19] V. Mrazek, S. S. Sarwar *et al.*, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *Proc. of ICCAD'16*, 2016, pp. 81:1–81:7.
[20] C. Li, W. Luo *et al.*, "Joint precision optimization and high level synthesis for approximate computing," in *DAC'15*, 2015, pp. 1–6.
[21] S. Mazahir, O. Hasan *et al.*, "Probabilistic error modeling for approximate adders," *IEEE Trans. Comput.*, vol. 66, no. 3, pp. 515–530, 2017.
[22] M. Ciesielski, C. Yu *et al.*, "Verification of gate-level arithmetic circuits by function extraction," in *Proc. of DAC '15*, 2015, pp. 52:1–52:6.
[23] A. Sayed-Ahmed, D. Große *et al.*, "Formal verification of integer multipliers by combining Gröbner basis with logic reduction," in *Proc. of DATE'16*, 2016, pp. 1048–1053.
[24] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan, "Macaco: Modeling and analysis of circuits for approximate computing," in *Proc. of ICCAD'11*, 2011, pp. 667–673.
[25] S. Chakraborty, K. S. Meel *et al.*, "Approximate probabilistic inference via word-level counting," in *Proc. of AAAI'16*, 2016, pp. 3218–3224.
[26] C.-S. Han and J.-H. R. Jiang, "When boolean satisfiability meets gaussian elimination in a simplex way," in *CAV'12*, 2012, pp. 410–426.
[27] J. F. Miller, *Cartesian Genetic Programming*, 2011.
[28] R. Brayton and A. Mishchenko, "ABC: An academic industrial-strength verification tool," in *Proc. of CAV'10*, ser. LNCS, 2010, pp. 24–40.

Syntax-Guided Optimal Synthesis for Chemical Reaction Networks

Luca Cardelli[1,2], Milan Češka[3], Martin Fränzle[4], Marta Kwiatkowska[2], Luca Laurenti[2], Nicola Paoletti[5], and Max Whitby[2(✉)]

[1] Microsoft Research Cambridge, Cambridge, UK
[2] Department of Computer Science, University of Oxford, Oxford, UK
max.whitby@keble.ox.ac.uk
[3] Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
[4] Department of Computer Science, Carl von Ossietzky Universität Oldenburg, Oldenburg, Germany
[5] Department of Computer Science, Stony Brook University, Stony Brook, USA

Abstract. We study the problem of optimal syntax-guided synthesis of stochastic Chemical Reaction Networks (CRNs) that plays a fundamental role in design automation of molecular devices and in the construction of predictive biochemical models. We propose a sketching language for CRNs that concisely captures syntactic constraints on the network topology and allows its under-specification. Given a sketch, a correctness specification, and a cost function defined over the CRN syntax, our goal is to find a CRN that simultaneously meets the constraints, satisfies the specification and minimizes the cost function. To ensure computational feasibility of the synthesis process, we employ the Linear Noise Approximation allowing us to encode the synthesis problem as a satisfiability modulo theories problem over a set of parametric Ordinary Differential Equations (ODEs). We design and implement a novel algorithm for the optimal synthesis of CRNs that employs almost complete refutation procedure for SMT over reals and ODEs, and exploits a meta-sketching abstraction controlling the search strategy. Through relevant case studies we demonstrate that our approach significantly improves the capability of existing methods for synthesis of biochemical systems and paves the way towards their automated and provably-correct design.

1 Introduction

Chemical Reaction Networks (CRNs) are a versatile language widely used for modelling and analysis of biochemical systems. The power of CRNs derives from the fact that they provide a compact formalism equivalent to Petri nets [42], Vector Addition Systems (VAS) [36] and distributed population protocols [4].

This work has been partially supported by the Czech Grant Agency grant No. GA16-17538S (M. Češka), Royal Society professorship, and EPSRC Programme on Mobile Autonomy (EP/M019918/1).

R. Majumdar and V. Kunčak (Eds.): CAV 2017, Part II, LNCS 10427, pp. 375–395, 2017.
DOI: 10.1007/978-3-319-63390-9_20

CRNs also serve as a high-level *programming language* for molecular devices [14,49] in systems and synthetic biology. Motivated by numerous potential applications ranging from smart therapeutics to biosensors, the construction of CRNs that exhibit prescribed dynamics is a major goal of synthetic biology [17,21,52]. Formal verification methods are now commonly embodied in the design process of biological systems [32,34,40] in order to reason about their correctness and performance. However, there is still a costly gap between the design and verification process, exacerbated in cases where stochasticity must be considered, which is typically the case for molecular computation. Indeed, automated synthesis of *stochastic CRNs* is generally limited to the estimation or synthesis of rate parameters [20,53], which neglect the network structure, and suffers from scalability issues [23].

Current research efforts in design automation aim to eliminate this gap and address the problem of *program synthesis* – automatic construction of programs from high-level specifications. The field of syntax-guided program synthesis [1] has made tremendous progress in recent years, based on the idea of supplementing the specification with a syntactic template that describes a high-level structure of the program and constrains the space of allowed programs. Applications range from bit-streaming programming [47] and concurrent data structures [46], to computational biology [37]. Often not only the correctness of synthesized programs is important, but also their optimality with respect to a given cost [8].

In this paper we consider the problem of optimal syntax-guided synthesis of CRNs. We work in the setting of *program sketching* [47], where the template is a partial program with holes (incomplete information) that are automatically resolved using a constraint solver. We define a sketching language for CRNs that allows designers to not only capture the high-level topology of the network and known dependencies among particular species and reactions, but also to compactly describe parts of the CRN where only limited knowledge is available or left unspecified (partially specified) in order to examine alternative topologies. A *CRN sketch* is therefore a parametric CRN, where the parameters can be unknown species, (real-valued) rates or (integer) stoichiometric constants. Our sketching language is well-suited for biological systems, where partial knowledge and uncertainties due to noisy or imprecise measurements are very common. We associate to a sketch a *cost* function that captures the structural complexity of the CRNs and reflects the cost of physically implementing it using DNA [14].

Traditionally, the dynamical behaviour of a CRN is represented as a *deterministic* time evolution of average species concentrations, described by a set of Ordinary Differential Equations (ODEs), or as a discrete-state *stochastic* process solved through the Chemical Master Equation (CME) [51]. Given the importance of faithfully modelling stochastic noise in biochemical systems [5,27], we focus on the (continuous) Linear Noise Approximation (LNA) of the CME [28,51]. It describes the time evolution of expectation and variance of the species in terms of ODEs, thus capturing the stochasticity intrinsic in CRNs, but, in contrast to solving the CME, scales well with respect to the molecular counts.

We can therefore represent the stochastic behaviour of a sketch as a set of parametric ODEs, which can be adequately solved as a satisfiability modulo theories (SMT) problem over the reals with ODEs. For this purpose, we employ the SMT solver iSAT(ODE) [26] that circumvents the well-known undecidability of this theory by a procedure generating either a certificate of unsatisfiability, or a solution that is precise up to an arbitrary user-defined precision.

To specify the desired *temporal behaviour* of the network, we support constraints about the expected number and variance of molecules, and, crucially, their derivatives over time. This allows us, for instance, to formalise that a given species shows a specific number of oscillations or has higher variability than another species, thus providing greater expressiveness compared to simple reachability specifications or temporal logic.

We therefore formulate and provide a solution to the following problem. For a given CRN sketch, a formal specification of the required temporal behaviour and a cost function, we seek a sketch instantiation (a concrete CRN) that satisfies the specification and minimizes the cost. The optimal solution for a given sketch is computed using the *meta-sketch* abstraction for CRNs inspired by [8]. It combines a representation of the syntactic search space with the cost function and defines an ordered set of sketches. This cost-based ordering allows us to effectively prune the search space during the synthesis process and guide the search towards the minimal cost.

In summary, this paper makes the following contributions:

- We propose the first sketching language for CRNs that supports partial specifications of the topology of the network and structural dependencies among species and reactions.
- We formulate a novel optimal synthesis problem that, thanks to the LNA interpretation of stochastic dynamics, can be solved as an almost complete decision/refutation problem over the reals involving parametric ODEs. In this way, our approach offers superior scalability with respect to the size of the system and the number of parameters and, crucially, supports the synthesis of the CRN structure and not just of rate parameters.
- We design a new synthesis algorithm that builds on the meta-sketch abstraction, ensuring the optimality of the solution, and the SMT solver iSAT.
- We develop a prototype implementation of the algorithm and evaluate the usefulness and performance of our approach on three case studies, demonstrating the feasibility of synthesising networks with complex dynamics in a matter of minutes.

We stress that CRNs provide not just a programming language for bio-systems, but a more general computational framework. In fact, CRNs are formally equivalent to population protocols and Petri nets. As a consequence, our methods enable effective program synthesis also in other non-biological domains [3].

Related Work. In the context of syntax-guided program synthesis (SyGuS) and program sketching, SMT-based approaches such as counter-example guided inductive synthesis [48] were shown to support the synthesis of deterministic

programs for a variety of challenging problems [8,46]. Sketching for probabilistic programs is presented in [43], together with a synthesis algorithm that builds on stochastic search and approximate likelihood computation. A similar approach appears in [11,31], where genetic algorithms and probabilistic model checking are used to synthesise probabilistic models from model templates (an extension of the PRISM language [38]) and multi-objective specifications. SyGuS has also been used for data-constrained synthesis, as in [24,37,45], where (deterministic) biological models are derived from gene expression data.

A variety of methods exist for estimating and synthesising rate parameters of CRNs, based on either the deterministic or stochastic semantics [2,6,10,20,35,41,53]. In contrast, our approach supports the synthesis of network structure and (uniquely) employs LNA.

Synthesis of CRNs from input-output functional specifications is considered in [23], via a method comprising two separate stages: (1) SMT-based generation of qualitative CRN models (candidates), and (2) for each candidate, parameter estimation of a parametric continuous time Markov chain (pCTMC). In contrast to our work, [23] do not consider solution optimality and require solving an optimisation problem for each concrete candidate on a pCTMC whose dimension is exponential in the number of molecules, making synthesis feasible only for very small numbers of molecules. On the other hand, our approach has complexity independent of the initial molecular population.

In [18], authors consider the problem of comparing CRNs of different size. They develop notions of bisimulations for CRNs in order to map a complex CRN into a simpler one, but with similar dynamical behaviour. Our optimal synthesis algorithm automatically guarantees that the synthesized CRN has the minimal size among all the CRNs consistent with the specification and the sketch.

2 Sketching Language for Chemical Reaction Networks

In this section, we introduce CRNs and the sketching language for their design.

2.1 Chemical Reaction Networks

CRN Syntax. A *chemical reaction network (CRN)* $\mathcal{C} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where Λ is a set of *species*, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is a set of reactions. Species in Λ interact according to the reactions in $\mathcal{R}$. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the coefficient associated with the rate of the reaction. r_τ and p_τ represent the stoichiometry of reactants and products. Given a reaction $\tau_1 = ([1,1,0],[0,0,2],k_1)$, we often refer to it as $\tau_1 : \lambda_1 + \lambda_2 \xrightarrow{k_1} 2\lambda_3$. The *state change* associated to τ is defined by $\upsilon_\tau = p_\tau - r_\tau$. For example, for τ_1 as above, we have $\upsilon_{\tau_1} = [-1,-1,2]$. The initial condition of a CRN is given by a vector of initial populations $x_0 \in \mathbb{N}^{|\Lambda|}$. A *chemical reaction system* (CRS) $C = (\Lambda, \mathcal{R}, x_0)$ is a tuple where (Λ, R) is a CRN and $x_0 \in \mathbb{N}^{|\Lambda|}$ represents its initial condition.

CRN Semantics. Under the usual assumption of mass action kinetics, the *stochastic* semantics of a CRN is generally given in terms of a discrete-state, continuous-time Markov process (CTMC) $(X(t), t \geq 0)$ [28], where the states, $x \in \mathbb{N}^{|\Lambda|}$, are vectors of molecular counts. Such a representation is accurate, but not scalable in practice because of the state space explosion problem [34,39]. An alternative *deterministic* model describes the evolution of the concentrations of the species as the solution $\Phi : \mathbb{R}_{\geq 0} \to \mathbb{R}^{|\Lambda|}$ of the following ODEs (the so called rate equations) [13]:

$$\frac{d\Phi(t)}{dt} = F(\Phi(t)) = \sum_{\tau \in \mathcal{R}} \upsilon_\tau \cdot (k_\tau \prod_{S \in \Lambda} \Phi_S^{r_{S,\tau}}(t)) \tag{1}$$

where Φ_S and $r_{S,\tau}$ are the components of vectors Φ and r_τ relative to species S. However, such a model does not take into account the stochasticity intrinsic in molecular interactions. In this paper, we work with the Linear Noise Approximation (LNA) [16,28,51], which describes the stochastic behaviour of a CRN in terms of a Gaussian process Y converging in distribution to X [9,28]. For a CRS $\mathcal{C} = (\Lambda, \mathcal{R}, x_0)$ contained in a system of volume N, we define $Y = N \cdot \Phi + \sqrt{N} \cdot Z$, where Φ is the solution of the rate equations (Eq. 1) with initial condition $\Phi(0) = \frac{x_0}{N}$. Z is a zero-mean Gaussian process with variance $C[Z(t)]$ described by

$$\frac{\mathrm{d}C[Z(t)]}{\mathrm{d}t} = J_F(\Phi(t))C[Z(t)] + C[Z(t)]J_F^T(\Phi(t)) + W(\Phi(t)) \tag{2}$$

where $J_F(\Phi(t))$ is the Jacobian of $F(\Phi(t))$, $J_F^T(\Phi(t))$ its transpose version, and $W(\Phi(t)) = \sum_{\tau \in \mathcal{R}} \upsilon_\tau {\upsilon_\tau}^T k_\tau \prod_{S \in \Lambda} (\Phi_S)^{r_{S,\tau}}(t)$. Y is a Gaussian process with expectation $E[Y(t)] = N\Phi(t)$ and covariance matrix $C[Y(t)] = NC[Z(t)]$. As a consequence, for any $t \in \mathbb{R}_{\geq 0}$, the distribution of $Y(t)$ is fully determined by its expectation and covariance. These are computed by solving the ODEs in Eq. 1–2, and thus avoiding the state space exploration. We denote by $[\![\mathcal{C}]\!]_N = (E[Y], C[Y])$ the solution of these equations for CRS $\mathcal{C}$ in a system of size N, henceforth called the *LNA model*. By using the LNA we can consider stochastic properties of CRNs whilst maintaining scalability comparable to that of the deterministic model [16]. In fact, the number of ODEs required for LNA is quadratic in the number of species and independent of the molecular counts.

2.2 CRN Sketching Language

CRN sketches are defined in a similar fashion to concrete CRNs, with the main difference being that species, stoichiometric constants and reaction rates are specified as unknown *variables*. The use of variables considerably increases the expressiveness of the language, allowing the modeller to specify *additional constraints* over them. Constraints facilitate the representation of key background knowledge of the underlying network, e.g. that a reaction is faster than another, or that it consumes more molecules than it produces.

Another important feature is that reactants and products of a reaction are lifted to *choices* of species (and corresponding stoichiometry). In this way, the modeller can explicitly incorporate in the reaction a set of admissible alternatives, letting the synthesiser resolve the choice.

Further, a sketch distinguishes between *optional* and *mandatory* reactions and species. These are used to express that some elements of the network *might* be present and that, on the other hand, other elements *must* be present. Our sketching language is well suited for synthesis of biological networks: it allows expressing key domain knowledge about the network, and, at the same time, it allows for network under-specification (holes, choices and variables). This is crucial for biological systems, where, due to inherent stochasticity or noisy measurements, the knowledge of the molecular interactions is often partial.

Definition 1 (Sketching language for CRNs). *A* CRN sketch *is a tuple* $\mathcal{S} = (\Lambda, \mathcal{R}, \mathsf{Var}, \mathsf{Dec}, \mathsf{Ini}, \mathsf{Con})$*, where:*

- $\Lambda = \Lambda_m \cup \Lambda_o$ *is a finite set of* species*, where* Λ_m *and* Λ_o *are sets of mandatory and optional species, respectively.*
- $\mathsf{Var} = \mathsf{Var}_\Lambda \cup \mathsf{Var}_c \cup \mathsf{Var}_r$ *is a finite set of* variable names*, where* Var_Λ*,* Var_c *and* Var_r *are sets of* species, coefficient and rate variables*, respectively.*
- Dec *is a finite set of* variable declarations*. Declarations bind variable names to their respective domains of evaluation and are of the form* $x : D$*, where* $x \in \mathsf{Var}$ *and* D *is the domain of* x*. Three types of declaration are supported:*
  - Species*, where* $x \in \mathsf{Var}_\Lambda$ *and* $D \subseteq \Lambda$ *is a finite non-empty set of species.*
  - Stoichiometric coefficients*, where* $x \in \mathsf{Var}_c$ *and* $D \subseteq \mathbb{N}$ *is a finite non-empty set of non-negative integers.*
  - Rate parameters*, where* $x \in \mathsf{Var}_r$ *and* $D \subseteq \mathbb{R}_{\geq 0}$ *is a bounded set of non-negative reals.*
- Ini *is the* set of initial states*, that is, a predicate on variables* $\{\lambda_0\}_{\lambda \in \Lambda}$ *describing the initial number of molecules for each species.*
- Con *is a finite set of* additional constraints*, specified as quantifier-free formulas over* Var*.*
- $\mathcal{R} = \mathcal{R}_m \cup \mathcal{R}_o$ *is a finite set of* reactions*, where* $\mathcal{R}_m$ *and* $\mathcal{R}_o$ *are sets of mandatory and optional reactions, respectively. As for a concrete CRNs, each* $\tau \in \mathcal{R}$ *is a triple* $\tau = (r_\tau, p_\tau, k_\tau)$*, where in this case* $k_\tau \in \mathsf{Var}_r$ *is a rate variable; the reaction complex* r_τ *and the product complex* p_τ *are* sets of reactants and products*, respectively. A* reactant $R \in r_\tau$ (product $P \in p_\tau$) is a finite choice of species and coefficients*, specified as a (non-empty) set* $R = \{c_i \lambda_i\}_{i=1,\ldots,|R|}$*, where* $c_i \in \mathsf{Var}_c$ *and* $\lambda_i \in \mathsf{Var}_\Lambda$*. We denote with* f_{r_τ} *the* uninterpreted choice function *for the reactants of* τ*, that is, a function* $f_{r_\tau} : r_\tau \to \mathsf{Var}_c \times \mathsf{Var}_\Lambda$ *such that* $f_{r_\tau}(R) \in R$ *for each* $R \in r_\tau$*. The choice function for products,* f_{p_τ}*, is defined equivalently.*

As an example, reaction $\tau = (\{\{c_1\lambda_1, c_2\lambda_2\}, \{c_3\lambda_3\}\}, \{\{c_4\lambda_4, c_5\lambda_5\}\}, k)$ is preferably written as $\{c_1\lambda_1, c_2\lambda_2\} + c_3\lambda_3 \xrightarrow{k} \{c_4\lambda_4, c_5\lambda_5\}$, using the shortcut $c_3\lambda_3$ to indicate the single-option choice $\{c_3\lambda_3\}$. A possible concrete choice function for the reactants of τ is the function $\overline{f_{r_\tau}} = \{\{c_1\lambda_1, c_2\lambda_2\} \mapsto c_1\lambda_1, \{c_3\lambda_3\} \mapsto c_3\lambda_3\}$ that chooses option $c_1\lambda_1$ as first reactant.

Holes and Syntactic Sugar. Unknown information about the network can be also expressed using *holes*, i.e. portions of the model left "unfilled" and resolved by the synthesiser. Holes, denoted with ?, are implicitly encoded through sketch variables. To correctly interpret holes, we assume default domains, $D_r \subseteq \mathbb{R}$ bounded and $D_c \subseteq \mathbb{N}$ finite, for rate and coefficient variables, respectively. We also support the implicit declaration of variables, as shown in Example 1.

The following example illustrates the proposed sketching language and the optimal solution obtained using our synthesis algorithm introduced in Sect. 4.

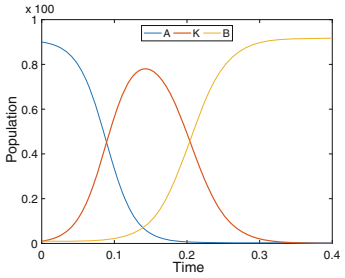
Example 1 (Bell shape generator). *For a given species K, our goal is to synthesize a CRN such that the evolution of K, namely the expected number of molecules of K, has a bell-shaped profile during a given time interval, i.e. during an initial interval the population K increases, then reaches the maximum, and finally decreases, eventually dropping to 0. Table 1 (left) defines a sketch for the bell-shape generator inspired by the solution presented in [12].*

Table 1. Left: the sketch for bell-shape generator, with volume $N = 100$. Right: CRN producing the bell-shape profile (species K) synthesized by our algorithm

$\Lambda_m = \{K\}, \Lambda_o = \{A, B\}, \mathcal{R}_m = \{\tau_1, \tau_2\},$
$\mathcal{R}_o = \{\tau_3\}$, $\mathsf{Dec} = \{c_1, \ldots, c_4 : [0, 2],$
$k_1, k_2, k_3 : [0, 0.1]$, $\lambda_1, \lambda_2 : \{A, B\}\},$
$\mathsf{Con} = \{\lambda_1 \neq \lambda_2,\ c_1 < c_2,\ c_3 > c_4\},$
$\mathsf{Ini} = \{K_0 = 1 \wedge A_0 \in [0, 100] \wedge B_0 \in [0, 100]\}$

$\tau_1 = \lambda_1 + c_1 K \xrightarrow{k_1} c_2 K$
$\tau_2 = \{0, 1\}\lambda_2 + c_3 K \xrightarrow{k_2} ?\lambda_2 + c_4 K$
$\tau_3 = \emptyset \xrightarrow{k_3} \{\lambda_2, [1, 2]K\}$

$A + K \rightarrow^{56} 2K; \quad K + B \rightarrow^{43} 2B$



This sketch reflects our prior knowledge about the control mechanism of the production/degradation of K. It captures that the solution has to have a reaction generating K (τ_1) and a reaction where K is consumed (τ_2). We also know that τ_1 requires a species, represented by variable λ_1, that is consumed by τ_1, and thus τ_1 will be blocked after the initial population of the species is consumed. An additional species, λ_2, different from λ_1, may be required. However, the sketch does not specify its role exactly: reaction τ_2 consumes either none or one molecule of λ_2 and produces an unknown number of λ_2 molecules, as indicated by the hole ?. There is also an optional reaction, τ_3, that does not have any reactants and produces either 1 molecule of λ_2 or between 1 and 2 molecules of K. The sketch further defines the mandatory and optional sets of species, the domains of the variables, and the initial populations of species. We assume the default domain $D_c = [0, 2]$, meaning that the hole ? can take values from 0 to 2. Note that many sketch variables are implicitly declared, e.g. term $[1, 2]K$ corresponds to $c'\lambda'$ with fresh variables $c' : [1, 2]$ and $\lambda' : \{K\}$.

Table 1 (right) shows the optimal CRN computed by our algorithm for the cost function given in Definition 3 and the bell-shape profile produced by the CRN.

We now characterise when a concrete network is a valid instantiation of a sketch.

Definition 2 (Sketch instantiation). *A CRS* $\mathcal{C} = (\Lambda_\mathcal{C}, \mathcal{R}_\mathcal{C}, x_0)$ *is a valid instantiation of a sketch* $\mathcal{S} = (\Lambda, \mathcal{R}, \mathsf{Var}, \mathsf{Dec}, \mathsf{Ini}, \mathsf{Con})$ *if:* $\mathsf{Ini}(x_0)$ *holds; there exists an interpretation* I *of the variables in* Var *and choice functions such that:*

1. *all additional constraints are satisfied:* $I \models \bigwedge_{\phi \in \mathsf{Con}} \phi$,
2. *for each* $\tau \in \mathcal{R}_m$ *there is* $\tau' \in \mathcal{R}_\mathcal{C}$ *that* realises τ, *i.e.,* τ' *is obtained from* τ *by replacing variables and choice functions with their interpretation*[1], *and*
3. *for each* $\tau' \in \mathcal{R}_\mathcal{C}$ *there is* $\tau \in \mathcal{R}$ *such that* τ' *realises* τ;

and the following conditions hold:

4. *for each* $\tau' = (r_{\tau'}, p_{\tau'}, k_{\tau'}) \in \mathcal{R}_\mathcal{C}$: $k_{\tau'} > 0$ *and* $r_{\tau'} + p_{\tau'} > 0$
5. $\Lambda_m \subseteq \Lambda_\mathcal{C}$ *and* $\Lambda_\mathcal{C} \subseteq \Lambda_m \cup \Lambda_o$ *and*
6. *for each species* $A \in \Lambda_\mathcal{C}$ *there is* $r \in \mathcal{R}_\mathcal{C}$ *such that* A *appears in* r *as reactant or product.*

Such an interpretation is called consistent *for* $\mathcal{S}$. *For sketch* $\mathcal{S}$ *and consistent interpretation* I, *we denote with* $I(\mathcal{S})$ *the* instantiation of $\mathcal{S}$ through I. *We denote with* $L(\mathcal{S})$ *the set of valid instantiations of* $\mathcal{S}$.

Condition 4. states that there are no void reactions, i.e. having null rate ($k_{\tau'} = 0$), or having no reactants and products ($r_{\tau'} + p_{\tau'} = 0$). Further, condition 6. ensures that the concrete network contains only species occurring in some reactions.

Example 2. *A CRS* $\mathcal{C}_1 = \{\{A, B, K\}, \{\tau'_1, \tau'_2, \tau'_3\}, x_0\}$ *where*

$$\tau'_1 = A + K \xrightarrow{0.01} 2K \quad \tau'_2 = B + K \xrightarrow{0.1} 2B \quad \tau'_3 = \emptyset \xrightarrow{0.001} K,$$

with $x_0 = (A_0 = 100, B_0 = K_0 = 1)$ *is a valid instantiation of the bell shape sketch* $\mathcal{S}$ *from Example 1. Reactions* τ'_1, τ'_2 *and* τ'_3 *realise respectively reaction sketches* τ_1, τ_2 *and* τ_3. *The corresponding consistent interpretation is* $I = \{\lambda_1 \mapsto A, c_1 \mapsto 1, k_1 \mapsto 0.01, c_2 \mapsto 2, c'_1 \mapsto 1, \lambda_2 \mapsto B, c_3 \mapsto 1, k_2 \mapsto 0.1, H \mapsto 2, c_4 \mapsto 0, k_3 \mapsto 0.001, f_{p_{\tau_3}} \mapsto \{\{\lambda_2, [1,2]K\} \mapsto [1,2]K\}, c'_2 \mapsto 1\}$, *where* c'_i *is the* i*-th implicit stoichiometric variable and* H *is the only hole. The interpretation of* $f_{p_{\tau_3}}$ *indicates that the choice* $\{\lambda_2, [1,2]K\}$ *is resolved as* $[1,2]K$.

Since a sketch instantiation corresponds to a CRS, we remark that its behaviour is given by the LNA model. Similarly, as we will show in Sect. 4, the SMT encoding of a sketch builds on a symbolic encoding of the LNA equations.

[1] When τ' realises sketch reaction τ, its reactants $r_{\tau'}$ is a set of the form $\{c_R \lambda_R\}_{R \in r_\tau}$, i.e. containing a concrete reactant for each choice R. Then, this is readily encoded in the reactant vector form $r_{\tau'} \in \mathbb{N}^{|\Lambda|}$ as per CRN definition (see Sect. 2.1). Similar reasoning applies for products $p_{\tau'}$.

3 Specification Language

We are interested in checking whether a CRN exhibits a given temporal profile. For this purpose, our specification language supports constraints about the expected number and variance of molecules, and, importantly, about their derivatives over time. This allows us, for instance, to synthesise a network where a given species shows a bell-shape profile (as in Example 1), or has variance greater than its expectation (considered in Sect. 5). Before explaining the specification language, we introduce the logical framework over which properties, together with CRN sketches, will be interpreted and evaluated.

3.1 Satisfiability Modulo ODEs

In syntax-guided synthesis, the synthesis problem typically reduces to an SMT problem [1]. Since we employ LNA, which generally involves non-linear ODEs, we resort to the framework of *satisfiability modulo ODEs* [25,26,30], which provides solving procedures for this theory that are sound and complete up to a user-specified precision. We stress that this framework allows for continuous encoding of the LNA equations, thus avoiding discrete approximations of its dynamics. Crucially, we can express arbitrary-order derivatives of the LNA variables, as these are smooth functions, and hence admit derivatives of all orders.

We employ the SMT solver iSAT(ODE) [25] that supports arithmetic constraint systems involving non-linear arithmetic and ODEs. The constraints solved are quantifier-free Boolean combinations of Boolean variables, arithmetic constraints over real- and integer-valued variables with bounded domains, and ODE constraints over real variables plus flow invariants. *Arithmetic constraints* are of the form $e_1 \sim e_2$, where $\sim \in \{<, \leq, =, \geq, >\}$ and $e_{1,2}$ are expressions built from real- and integer-valued variables and constants using functions from $\{+, -, \cdot, \sin, \cos, \mathrm{pow}_{\mathbb{N}}, \exp, \min, \max\}$. *ODE constraints* are time-invariant and given by $\frac{\mathrm{d}x}{\mathrm{d}t} = e$, where e is an expression as above[2] containing variables themselves defined by ODE constraints. *Flow invariant constraints* are of the form $x \leq c$ or $x \geq c$, with x being an ODE-defined variable and c being a constant. ODE constraints have to occur under positive polarity and are interpreted as first-order constraints on pre-values x and post-values x' of their variables, i.e., they relate those pairs (x, x') being connected by a trajectory satisfying $\frac{\mathrm{d}x}{\mathrm{d}t} = e$ and, if present, the flow invariant throughout.

Due to undecidability of the fragment of arithmetic addressed, iSAT(ODE) implements a sound, yet quantifiably incomplete, unsatisfiability check based on a combination of interval constraint propagation (ICP) for arithmetic constraints, safe numeric integration of ODEs, and conflict-driven clause learning (CDCL) for manipulating the Boolean structure of the formula. This procedure investigates "boxes", i.e. Cartesian products of intervals, in the solution space until it either finds a proof of unsatisfiability based on a set of boxes covering the original domain or finds some hull-consistent box [7], called a *candidate*

[2] Where we can additionally use non-total functions $/$, $\sqrt{}$ and ln.

solution box, with edges smaller than a user-specified width $\delta > 0$. While the interval-based unsatisfiability proof implies unsatisfiability over the reals, thus rendering the procedure sound, the report of a candidate solution box only guarantees that a slight relaxation of the original problem is satisfiable. Within this relaxation, all original constraints are first rewritten to equi-satisfiable inequational form $t \sim 0$, with $\sim \in \{>, \geq\}$, and then relaxed to the strictly weaker constraint $t \sim -\delta$. In that sense, iSAT and related algorithms [30,50] provide reliable verdicts on either unsatisfiability of the original problem or satisfiability of its aforementioned δ-relaxation, and do in principle[3] always terminate with one of these two verdicts. Hence the name "δ-decidability" used by Gao et al. in [29].

3.2 Specification for CRNs

The class of properties we support are formulas describing a dynamical profile composed as a finite sequence of phases. Each phase i is characterised by an arithmetic predicate $\mathsf{pre} - \mathsf{post}_i$, describing the system state at its start and end points (including arithmetic relations between these two), as well as by flow invariants (formula inv_i) pertaining to the trajectory observed during the phase. Formally, a specification φ comprising $M \geq 1$ phases is defined by

$$\varphi = \bigwedge_{i=1}^{M} \mathsf{inv}_i \wedge \mathsf{pre} - \mathsf{post}_i \tag{3}$$

Note that entry as well as target conditions of phases can be expressed within $\mathsf{pre} - \mathsf{post}_i$. Initial conditions are not part of the specification but, as explained in Sect. 4, the sketch definition.

CRS Correctness. For a CRS $\mathcal{C}$, Volume N, and property φ, we are interested in checking whether $\mathcal{C}$ is *correct* with respect to φ, written $[\![\mathcal{C}]\!]_N \models \varphi$, i.e., whether $\mathcal{C}$ at Volume N exhibits the dynamic behavior required by φ. Since $[\![\mathcal{C}]\!]_N$ is a set of ODEs, this corresponds to checking whether $\hat{\varphi} \wedge \varphi_{[\![\mathcal{C}]\!]_N}$ is satisfiable, where $\varphi_{[\![\mathcal{C}]\!]_N}$ is an SMT formula encoding the set of ODEs given by $[\![\mathcal{C}]\!]_N$ and their higher-order derivatives[4] by means of the corresponding ODE constraints, and $\hat{\varphi}$ is the usual bounded model checking (BMC) unwinding of the step relation $\bigwedge_{i=1}^{M}(\mathit{phase} = i \Rightarrow \mathsf{inv}_i \wedge \mathsf{pre} - \mathsf{post}_i) \wedge \mathit{phase}' = \mathit{phase} + 1$ encoding the phase sequencing and the pertinent phase constraints, together with the BMC target $\mathit{phase} = M$ enforcing all phases to be traversed. As this satisfiability problem is undecidable in general, we relax it to checking whether $\hat{\varphi} \wedge \varphi_{[\![\mathcal{C}]\!]_N}$ is δ-satisfiable in the sense of admitting a candidate solution box of width δ. In that case, we write $[\![\mathcal{C}]\!]_N \models^{\delta} \varphi$.

[3] i.e., when considering the abstract algorithms using unbounded precision rather than the safe rounding employed in their floating-point based actual implementations.

[4] Only the derivatives appearing in φ are included. These are encoded using the Faà di Bruno's formula [33].

Example 3 (Specification for the bell-shape generator). *The required bell-shaped profile for Example 1 can be formalized using a 2-phase specification as follows:*

$$\mathsf{inv}_1 \equiv E^{(1)}[K] \geq 0, \quad \mathsf{pre\text{-}post}_1 \equiv E^{(1)}[K]' = 0 \ \wedge \ E[K]' > 30,$$
$$\mathsf{inv}_2 \equiv E^{(1)}[K] \leq 0, \quad \mathsf{pre\text{-}post}_2 \equiv E[K]' \leq 1 \ \wedge \ T' = 1$$

where $E[K]$ is the expected value of species K and $E^{(1)}[K]$ its first derivative. T is the global time. Primed notation $(E[K]', E^{(1)}[K]', T')$ indicates the variable value at the end of the respective phase. Constraints inv_1 and inv_2 require, respectively, that $E[K]$ is not decreasing in the first phase, and not increasing in the second (and last) phase. $\mathsf{pre\text{-}post}_1$ states that, at the end of phase 1, $E[K]$ is a local optimum ($E^{(1)}[K]' = 0$), and has an expected number of molecules greater than 30. $\mathsf{pre\text{-}post}_2$ states that, at the final phase, the expected number of molecules of K is at most 1 and that the final time is 1.

This example demonstrates that we can reason over complex temporal specifications including, for instance, a relevant fragment of bounded metric temporal logic [44].

4 Optimal Synthesis of Chemical Reaction Networks

In this section we formulate the optimal synthesis problem where we seek to find a concrete instantiation of the sketch (i.e. a CRN) that satisfies a given property and has a minimal cost. We further show the encoding of the problem using satisfiability modulo ODEs and present an algorithm scheme for its solution.

4.1 Problem Formulation

Before explaining our optimal synthesis problem, we first need to introduce the class of cost functions considered. A cost function G for a sketch $\mathcal{S}$ has signature $G : L(\mathcal{S}) \to \mathbb{N}$ and maps valid instantiations of $\mathcal{S}$ to a natural cost. A variety of interesting cost functions fit this description, and, depending on the particular application, the modeller can choose the most appropriate one. A special case is, for instance, the overall number of species and reactions, a measure of CRN complexity used in e.g. CRN comparison and reduction [18,19]. Importantly, cost functions are defined over the structure of the concrete instantiation, rather than its dynamics. As we shall see, this considerably simplifies the optimisation task, since it leads to a finite set of admissible costs. In the rest of the paper, we consider the following cost function, which captures the structural complexity of the CRN and the cost of physically implementing it using DNA [14,49].

Definition 3 (Cost function). *For a sketch $\mathcal{S} = (\Lambda, \mathcal{R}, \mathsf{Var}, \mathsf{Dec}, \ \mathsf{Ini}, \mathsf{Con})$, we consider the cost function $G_{\mathcal{S}} : L(\mathcal{S}) \to \mathbb{N}$ that, for any CRS instantiation $\mathcal{C} = (\Lambda, \mathcal{R}) \in L(\mathcal{S})$, is defined as:*

$$G_{\mathcal{S}}(\mathcal{C}) = 3 \cdot (|\Lambda \cap \Lambda_o|) + \sum_{\tau \in \mathcal{R}_{\mathcal{C}}} \sum_{S \in \Lambda} 6 \cdot r_{S,\tau} + 5 \cdot p_{S,\tau}$$

where $r_{S,\tau}$ ($p_{S,\tau}$) is the stoichiometry of species S as reactant (product) of τ.

This cost function penalizes the presence of optional species (Λ_o) and the number of reactants and products in each reaction. It does not explicitly include a penalty for optional reactions, but this is accounted for through an increased total number of reactants and products. We stress that different cost functions can be used, possibly conditioned also on the values of reaction rates.

Problem 1 *(Optimal synthesis of CRNs). Given a sketch S, cost function $G_{\mathcal{S}}$, property φ, Volume N and precision δ, the optimal synthesis problem is to find CRS $\mathcal{C}^* \in L(\mathcal{S})$, if it exists, such that $[\![\mathcal{C}^*]\!]_N \vDash^{\delta} \varphi$ and, for each CRS $\mathcal{C} \in L(\mathcal{S})$ such that $G_{\mathcal{S}}(\mathcal{C}) < G_{\mathcal{S}}(\mathcal{C}^*)$, it holds that $[\![\mathcal{C}]\!]_N \nvDash^{\delta} \varphi$.*

An important characteristic of the sketching language and the cost function is that for each sketch $\mathcal{S}$ the set $\{G_{\mathcal{S}}(\mathcal{C}) \mid \mathcal{C} \in L(\mathcal{S})\}$ is finite. This follows from the fact that $\mathcal{S}$ restricts the maximal number of species and reactions as well as the maximal number of reactants and products for each reaction. Therefore, we can define for each sketch $\mathcal{S}$ the minimal cost $\mu_{\mathcal{S}}$ and the maximal cost $\nu_{\mathcal{S}}$.

Example 4. *It is easy to verify that the cost of the CRS $\mathcal{C}$ of Example 2, a valid instantiation of the bell-shape generator sketch $\mathcal{S}$, is $G_{\mathcal{S}}(\mathcal{C}) = 3 \cdot 2 + 6 \cdot 4 + 5 \cdot 5 = 55$, and that minimal and maximal costs of sketch $\mathcal{S}$ are, respectively, $\mu_{\mathcal{S}} = 3 \cdot 1 + 6 \cdot 2 + 5 \cdot 2 = 25$ and $\nu_{\mathcal{S}} = 3 \cdot 2 + 6 \cdot 5 + 5 \cdot 7 = 71$.*

We now define a meta-sketch abstraction for our sketching language that allows us to formulate an efficient optimal synthesis algorithm.

Definition 4 (Meta-sketch for CRNs). *Given a sketch $\mathcal{S}$ and a cost function $G_{\mathcal{S}}$, we define the meta-sketch $\mathcal{M}_{\mathcal{S}} = \{\mathcal{S}(i) \mid \mu_{\mathcal{S}} \leq i \leq \nu_{\mathcal{S}}\}$, where $\mathcal{S}(i)$ is a sketch whose instantiations have cost smaller than i, i.e. $L(\mathcal{S}(i)) = \{C \in L(\mathcal{S}) \mid G_{\mathcal{S}}(C) < i\}$.*

A meta-sketch $\mathcal{M}_{\mathcal{S}}$ establishes a hierarchy over the sketch $\mathcal{S}$ in the form of an ordered set of sketches $\mathcal{S}(i)$. The ordering reflects the size of the search space for each $\mathcal{S}(i)$ as well as the cost of implementing the CRNs described by $\mathcal{S}(i)$. In contrast to the abstraction defined in [8], the ordering is given by the cost function and thus it can be directly used to guide the search towards the optimum.

4.2 Symbolic Encoding

Given a sketch of CRN $\mathcal{S} = (\Lambda, \mathcal{R}, \mathsf{Var}, \mathsf{Dec}, \mathsf{Ini}, \mathsf{Con})$, we show that the dynamics of $L(\mathcal{S})$, set of possible instantiations of $\mathcal{S}$, can be described symbolically by a set of parametric ODEs, plus additional constraints. These equations depend on the sketch variables and on the choice functions of each reaction, and describe the time evolution of mean and variance of the species.

For $S \in \Lambda, \lambda \in \mathsf{Var}$, we define the indicator function $\mathcal{I}_S(\lambda) = 1$ if $\lambda = S$, and 0 otherwise. For $S \in \Lambda$ and $\tau \in \mathcal{R}$, we define the following constants:

$$r_{S,\tau} = \sum_{\substack{R \in r_\tau \\ (c,\lambda) = f_{r_\tau}(R)}} c \cdot \mathcal{I}_S(\lambda), \qquad p_{S,\tau} = \sum_{\substack{P \in p_\tau \\ (c,\lambda) = f_{p_\tau}(P)}} c \cdot \mathcal{I}_S(\lambda), \qquad v_{S,\tau} = p_{S,\tau} - r_{S,\tau}$$

Note that these are equivalent to the corresponding coefficients for concrete CRNs, but now are parametric as they depend on the sketch variables. As for the LNA model of Sect. 2.1, symbolic expectation and variance together characterise the symbolic behaviour of sketch $\mathcal{S}$, given as the set of parametric ODEs $[\![\mathcal{S}]\!]_N = (N \cdot \Phi, N \cdot C[Z])$, for some Volume N.

The functions $\Phi(t)$ and $C[Z(t)]$ describe symbolically the time evolution of expected values and covariance of all instantiations of $\mathcal{S}$, not just of valid instantiations. We restrict to valid instantiations by imposing the following formula:

$$\mathsf{consist} \equiv \mathsf{Ini}(x_0) \wedge \bigwedge_{\phi \in \mathsf{Con}} \phi \ \wedge \bigwedge_{\tau \in \mathcal{R}_m} \neg \mathsf{void}(\tau) \ \wedge \bigwedge_{S \in \Lambda_m} \mathsf{used}(S)$$

which, based on Definition 2, states that initial state and additional constraints have to be met, all mandatory reactions must not be void, and all mandatory species must be "used", i.e. must appear in some (non-void) reactions. Note that we allow optional reactions to be void, in which case they are not included in the concrete network. Formally, $\mathsf{void}(\tau) \equiv \ (k_\tau = 0) \vee \sum_{S \in \Lambda}(r_{S,\tau} + p_{S,\tau}) = 0$ and $\mathsf{used}(S) \equiv \ \bigvee_{\tau \in \mathcal{R}} \neg \mathsf{void}(\tau) \wedge (r_{S,\tau} + p_{S,\tau}) > 0$.

Sketch Correctness. Given an interpretation I consistent for $\mathcal{S}$, call Φ_I and $C[Z]_I$, the concrete functions obtained from Φ and $C[Z]$ by substituting variables and functions with their assignments in I. The symbolic encoding ensures that the LNA model $[\![I(\mathcal{S})]\!]_N$ of CRS $I(\mathcal{S})$ (i.e. the instantiation of $\mathcal{S}$ through I, see Definition 2) is equivalent to $(\Phi_I, C[Z]_I)$.

With reference to our synthesis problem, this implies that the synthesis of a CRS $\mathcal{C}^*$ that satisfies a correctness specification φ from a sketch $\mathcal{S}$ corresponds to finding a consistent interpretation for $\mathcal{S}$ that satisfies φ. Similarly to the case for concrete CRSs, this corresponds to checking if $\hat{\varphi} \wedge \mathsf{consist} \wedge \varphi_{[\![\mathcal{S}]\!]_N}$ is δ-satisfiable for some precision δ, where $\hat{\varphi}$ is the BMC encoding of ϕ (see Sect. 3.2) and $\varphi_{[\![\mathcal{S}]\!]_N}$ is the SMT encoding of the symbolic ODEs given by $[\![\mathcal{S}]\!]_N$ and the corresponding derivatives.

Cost Constraints. For a sketch $\mathcal{S}$ and cost $i \in \mathbb{N}$, the following predicate encodes the cost function of Definition 3 in order to restrict $\mathcal{S}$ into $\mathcal{S}(i)$, i.e. the sketch whose instantiations have cost smaller than i:

$$\mathsf{Con}_G(i) \equiv \left(3 \cdot \sum_{S \in \Lambda_o} \mathcal{I}(\mathsf{used}(S)) + \sum_{\tau \in \mathcal{R}} \mathcal{I}(\neg \mathsf{void}(\tau)) \cdot \sum_{S \in \Lambda} (6 \cdot r_{S,\tau} + 5 \cdot p_{S,\tau}) \right) < i$$

where $\mathcal{I}$ is the indicator function, and used and void are predicates defined above.

4.3 Algorithm Scheme for Optimal Synthesis

In Algorithm 1, we present an algorithm scheme for solving the optimal synthesis problem for CRNs. It builds on the meta-sketch abstraction described in Definition 4, which enables effective pruning of the search space through cost

Algorithm 1. Generalised synthesis scheme

Require: Meta-sketch $\mathcal{M}_{\mathcal{S}}$, property φ, precision δ and initial precision δ_{init}
Ensure: C^* is a solution of Problem 1 if $\exists\ C \in L(\mathcal{M}^{\omega}_{\mathcal{S}}) : C \vDash^{\delta} \varphi$, otherwise $C^* = \mathsf{null}$

1: $i^{\top} \leftarrow \nu_{\mathcal{S}}$; $i^{\perp} \leftarrow \mu_{\mathcal{S}}$; $i \leftarrow g(i^{\perp}, i^{\top})$; $C^* \leftarrow \mathsf{null}$
2: **repeat**
3: $\quad SAT_1 \leftarrow \delta\text{-}\mathsf{Solver}(\mathcal{S}(i), \varphi, \delta_{init})$; $SAT_2 \leftarrow \mathsf{false}$
4: $\quad$ **if** SAT_1 **then**
5: $\quad\quad (M, SAT_2) \leftarrow \delta\text{-}\mathsf{Solver}(\mathcal{S}(i), \varphi, \delta)$
6: $\quad\quad$ **if** SAT_2 **then** $C^* = \mathsf{getSoln}(\mathcal{S}(i), M)$
7: $\quad\quad$ **else** $\delta_{init} = (\delta_{init} - \delta)/2$
8: $\quad (i^{\perp}, i^{\top}) \leftarrow f(i, i^{\perp}, i^{\top}, SAT_2, \mathsf{G}_{\mathcal{S}}(C^*))$; $i \leftarrow g(i^{\perp}, i^{\top})$
9: **until** $i^{\perp} \leq i^{\top}$
10: **return** C^*

constraints, and the SMT-based encoding of Sect. 4.2, which allows for the automated derivation of meta-sketch instantiations (i.e. CRNs) that satisfy the specification and the cost constraints.

This scheme repeatedly invokes the SMT solver (δ-Solver) on the sketch encoding, and at each call the cost constraints are updated towards the optimal cost. We consider three approaches: (1) *top-down*: starting from the maximal cost $\nu_{\mathcal{S}}$, it solves meta-sketches with decreasing cost until no solution exists (UNSAT); (2) *bottom-up:* from the minimal cost $\mu_{\mathcal{S}}$, it increases the cost until a solution is found (SAT); (3) *binary search:* it bounds the upper estimate on the optimal solution using a SAT witness and the lower estimate with an UNSAT witness.

We further improve the algorithm by exploiting the fact that UNSAT witnesses can also be obtained at a lower precision δ_{init} ($\delta_{init} \gg \delta$), which consistently improves performance. Indeed, UNSAT outcomes are precise and thus valid for any precision. Note that the top-down strategy does not benefit from this speed-up since it only generates SAT witnesses.

At every iteration, variable i maintains the current cost. The solver is firstly called using the rough precision δ_{init} (line 3). If the solver returns SAT (potential false positive), we refine our query using the required precision δ (line 5). If this query is in turn satisfiable, then the solver also returns a candidate solution box M, where all discrete variables are instantiated to a single value and an interval smaller than δ is assigned to each real-valued variable. Function getSoln computes the actual sketch instantiation C^* as the centre point of M that δ-satisfies φ. The cost of C^* provides the upper bound on the optimal solution. If either query returns UNSAT, the current cost i provides the lower bound on the optimal solution. The second query being UNSAT implies that the rough precision δ_{init} produced a false positive, and thus it is refined for the next iteration (line 7).

The actual search strategy used in Algorithm 1 is given by the functions f controlling how the upper ($i^{\top}$) and lower ($i^{\perp}$) bounds on the cost are updated and by g determining the next cost to explore. Note that such bounds ensure the termination of the algorithm (line 9). In the bottom-up approach, f "terminates" the search (i.e. causes $i^{\perp} > i^{\top}$) if SAT_2 is true (i.e. when the first SAT witness

is obtained), otherwise f sets $(i^{\perp}, i^{\top}) \leftarrow (i+1, i^{\top})$ and g sets $i \leftarrow i^{\perp}$. In the top-down case, f terminates the search if SAT_2 is false (i.e. at the first UNSAT witness), otherwise it sets $(i^{\perp}, i^{\top}) \leftarrow (i^{\perp}, \mathsf{G}_{\mathcal{S}}(C^*)-1)$ and $i \leftarrow i^{\top}$, where $\mathsf{G}_{\mathcal{S}}(C^*)$ is the cost of CRN C^*. Binary search is obtained with f that updates $(i^{\perp}, i^{\top})$ to $(i^{\perp}, \mathsf{G}_{\mathcal{S}}(C^*) - 1)$ if $SAT_2 = \mathsf{true}$, to $(i+1, i^{\top})$ otherwise, and with g that updates i to $i^{\perp} + \lfloor (i^{\top} - i^{\perp})/2 \rfloor$.

5 Experimental Evaluation

We evaluate the usefulness and performance of our optimal synthesis method on three case studies, representative of important problems studied in biology: (1) the **bell-shape generator**, a component occurring in signaling cascades; (2) **Super Poisson**, where we synthesize CRN implementations of stochastic processes with prescribed levels of process noise; and (3) **Phosphorelay network**, where we synthesize CRNs exhibiting switch-like sigmoidal profiles, which is the biochemical mechanism underlying cellular decision-making, driving in turn development and differentiation.

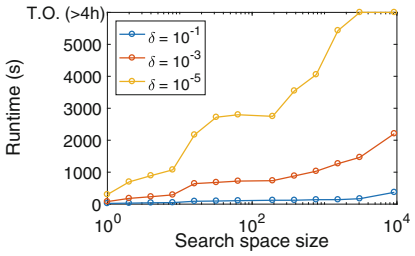
We employ the solver iSAT(ODE) [25,26][5], even if our algorithm supports any δ-solver. We ran preliminary experiments using the tool dReal [30], finding that iSAT performs significantly better on our instances. All experiments were run on a server with a Intel Xeon CPU E5645 @2.40 GHz processor (using a single core) and 24GB @1333 MHz RAM.

Bell-Shape Generator. We use the example described in Examples 1 and 3, resulting in 8 parametric ODEs, as the main benchmark. The synthesised CRN is shown in Fig. 1. In the first experiment, we evaluate the scalability of the solver with respect to precision δ and the size of the discrete search space, altered by changing the domains of species and coefficient variables of the sketch. We exclude cost constraints as they reduce the size of the search space. Runtimes, reported in Table 2 (left), correspond to a single call to iSAT with different δ values, leading to SAT outcomes in all cases. Note that the size of the continuous state space, given by the domains of rate variables, does not impose such a performance degradation, as shown in Table 3 (right) for a different model.

In the second experiment, we analyse how cost constraints and different variants of Algorithm 1 affect the performance of optimal synthesis. Table 2 (right) shows the number of iSAT calls with UNSAT/SAT outcomes (2nd column) and total runtimes without/with the improvement that attempts to obtain UNSAT witnesses at lower precision ($\delta_{init} = 10^{-1}$). Importantly, the average runtime for a single call to iSAT is significantly improved when we use cost constraints, since these reduce the discrete search space (between 216 s and 802 s with cost constraints, 1267 s without). Moreover, results clearly indicate that UNSAT cases are considerably faster to solve, because inconsistent cost constraints typically

[5] Version r2806. Parameters: `--maxdepth=k` (`k` is the BMC unrolling depth) and `--ode-opts=--continue-after-not-reaching-horizon`.

Table 2. Performance of bell-shape generator model. Left: runtimes for different precisions δ and discrete search space size. Right: optimal synthesis with different variants of Algorithm 1, fixed discrete search space size (1536) and $\delta = 10^{-3}$.



Search Strategy	# iSAT calls	Total time (s)
bottom-up	7/1	2671/1732
top-down	1/6	4863/5612
binary-search	2/4	3440/3121

lead to trivial UNSAT instances. This favours the bottom-up approach over the top-down. In this example, the bottom-up approach also outperforms binary-search, but we expect the opposite situation for synthesis problems with wider spectra of costs. As expected, we observe a speed-up when using a lower precision for UNSAT witnesses, except for the top-down approach.

Super Poisson. We demonstrate that our approach is able to synthesise a CRN that behaves as a stochastic process, namely, a super Poisson process having variance greater than its expectation. We formalise the behaviour on the interval $[0, 1]$ using a 1-phase specification as shown in Table 3 (left). For $N = 100$ we consider the sketch listed in Table 3 (center) where both reactions are mandatory, reflecting the knowledge that A is both produced and degraded.

Table 3. Left: the 1-phase specification of the super poisson process. Centre: the sketch. Right: runtimes for different precisions.

$\mathsf{inv}_1 \equiv C[A] > E[A]$	$\Lambda = \{A, B\}, \Lambda_o = \{B\}, \lambda_1, \lambda_2 : \Lambda,$	Rate interval | Time (s)
$\mathsf{pre\text{-}post}_1 \equiv T' = 1$	$\mathcal{R}_m = \{\tau_1, \tau_2\}, A_0 = B_0 = 0,$	$[0, 1]$ | 4
	$k_1, k_2 : [0, 100], c_1, c_2, c_3 : [0, 2]$	$[0, 10]$ | 18
	$\tau_1 :\rightarrow^{k_1} c_1 A + c_2 \lambda_1; \quad \tau_2 : A \rightarrow^{k_2} c_3 \lambda_2;$	$[0, 100]$ | 31

Using precision $\delta = 10^{-3}$, we obtained the optimal solution $\{\xrightarrow{23} 2A, A \xrightarrow{94}\}$ (cost 16) in 4 s. Notably, the synthesis without the cost constraints took 19 s. Moreover, the ability to reason over the variance allows the solver to discard solution $\{\rightarrow A, A \rightarrow\}$ (implementation of a Poisson process [15]), which would have led to a variance equal to expectation. Table 3 (right) demonstrates the scalability of our approach with respect to the size of the continuous parameter space. Despite its non-trivial size (10 ODEs and discrete search space of size 288), we obtain remarkable performance, with runtimes in the order of seconds.

Phosphorelay Network. In the last case study we present a rate synthesis problem (i.e. all discrete parameters are instantiated) for a three-layer phosphorelay network [22]. In this network, each layer Li $(i = 1, 2, 3)$ can be found in phosphorylated form, Lip, and there is the ligand B, acting as an input for the network. The authors of [22] were interested in finding rates such that the time dynamics of $L3p$ shows ultra-sensitivity – a sigmoid shape of the time evolution of $L3p$ – which they obtained by manually varying the parameters until the right profile was discovered. We show that our approach can automatically find these parameters, thus overcoming such a tedious and time-consuming task.

We formalise the required behaviour using the 2-phase specification as shown in Table 4 (left). In particular, we consider a time interval $[0, 1]$ during which $L3p$ never decreases $(E^{(1)}[L3p] \geq 0)$, and we require that an inflection point in the second derivative occurs in the transition between the two phases. At the final time we require that the population of $L3p$ is above 100, to rule out trivial solutions. For $N = 1000$ we consider the sketch listed in Table 4 (center), inspired by [22]. Figure 1 lists the rates synthesised for $\delta = 10^{-3}$ and illustrates the obtained sigmoid profile.

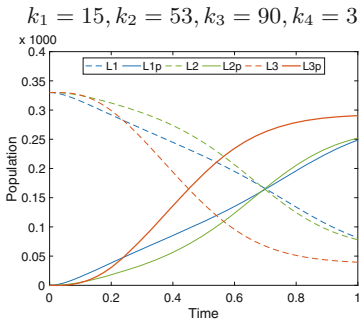


Fig. 1. The synthesised rates and the corresponding profile (without variance constraints).

We further consider a more complex variant of the problem, where we extend the specification to require that the variance of $L3p$ on its inflection point (the point where the variance is known to reach its maximum [22]) is limited by a threshold. This extension led to an encoding with 37 symbolic ODEs, compared to the 9 ODEs (7 species plus two ODEs for the derivatives of $L3p$) needed for the previous specification. Table 4 (right) shows the runtimes of the synthesis process for both variants of the model and different precisions δ. The results demonstrate that neither increasing the number of ODEs nor improving the precision leads to exponential slowdown of the synthesis process, indicating good scalability of our approach.

Table 4. Left: the 2-phase specification of the sigmoid profile (no variance constraints). Centre: the sketch. Right: runtimes for different precisions and the two variants (without and with covariances).

$\mathsf{inv}_1 \equiv E^{(1)}[L3p] \geq 0 \wedge E^{(2)}[L3p] \geq 0$	$L1 + B \xrightarrow{k_1} B + L1p$	ODEs	δ	Time (s)
$\mathsf{pre\text{-}post}_1 \equiv E^{(2)}[L3p]' = 0$	$L2 + L1p \xrightarrow{k_2} L1 + L2p$	9	10^{-1}	53
$\mathsf{inv}_2 \equiv E^{(1)}[L3p] \geq 0 \wedge E^{(2)}[L3p] \leq 0$	$L2p + L3 \xrightarrow{k_3} L2 + L3p$	9	10^{-3}	370
$\mathsf{pre\text{-}post}_2 \equiv E[L3p]' > 100 \wedge T' = 1$	$L3p \xrightarrow{k_4} L3; \quad \emptyset \xrightarrow{1} B$	9	10^{-5}	719
	$k_1, \ldots, k_4 : (0, 100],$	37	10^{-1}	1052
	$Li_0 = 330, Lip_0 = B_0 = 0$	37	10^{-3}	11276
		37	10^{-5}	39047

6 Conclusion

Automated synthesis of biochemical systems that exhibit prescribed behaviour is a landmark of synthetic and system biology. We presented a solution to this problem, introducing a novel method for SMT-based optimal synthesis of stochastic CRNs from rich temporal specifications and sketches (syntactic templates). By means of the LNA, we define the semantics of a sketch in terms of a set of parametric ODEs quadratic in the number of species, which allows us to reason about stochastic aspects not possible with the deterministic ODE-based semantics. Able to synthesize challenging systems with up to 37 ODEs and $\sim$10 K admissible network topologies, our method shows unprecedented scalability and paves the way for design automation for provably-correct molecular devices.

In future work we will explore alternative notions of optimality and encodings, and develop a software tool based on parallel search strategies.

References

1. Alur, R., et al.: Syntax-guided synthesis. Dependable Softw. Syst. Eng. **40**, 1–25 (2015)
2. Andreychenko, A., Mikeev, L., Spieler, D., Wolf, V.: Parameter identification for Markov models of biochemical reactions. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 83–98. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_8
3. Angluin, D., Aspnes, J., Eisenstat, D.: Fast computation by population protocols with a leader. Distrib. Comput. **21**(3), 183–199 (2008)
4. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
5. Arkin, A., Ross, J., McAdams, H.H.: Stochastic kinetic analysis of developmental pathway bifurcation in phage λ-infected Escherichia coli cells. Genetics **149**(4), 1633–1648 (1998)
6. Barnat, J., et al.: On parameter synthesis by parallel model checking. IEEE/ACM Trans. Comput. Biol. Bioinf. **9**(3), 693–705 (2012)
7. Benhamou, F., Goualard, F., Granvilliers, L., Puget, J.F.: Revising hull and box consistency. In: ICLP 1999. MIT Press, pp. 230–244 (1999)
8. Bornholt, J., Torlak, E., Grossman, D., Ceze, L.: Optimizing synthesis with metasketches. In: POPL 2016. ACM, pp. 775–788 (2016)
9. Bortolussi, L., Cardelli, L., Kwiatkowska, M., Laurenti, L.: Approximation of probabilistic reachability for chemical reaction networks using the linear noise approximation. In: Agha, G., Houdt, B. (eds.) QEST 2016. LNCS, vol. 9826, pp. 72–88. Springer, Cham (2016). doi:10.1007/978-3-319-43425-4_5
10. Bortolussi, L., Milios, D., Sanguinetti, G.: Smoothed model checking for uncertain Continuous-Time Markov Chains. Inf. Comput. **247**, 235–253 (2016)
11. Calinescu, R.C., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Designing robust software systems through parametric markov chain synthesis. In: IEEE International Conference on Software Architecture (ICSA 2017). IEEE (2017)
12. Cardelli, L.: Artificial biochemistry. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses, pp. 429–462. Springer, Heidelberg (2009)

13. Cardelli, L.: Morphisms of reaction networks that couple structure to function. BMC Syst. Biol. **8**(1), 84 (2014)
14. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. **23**(02), 247–271 (2013)
15. Cardelli, L., Kwiatkowska, M., Laurenti, L.: Programming discrete distributions with chemical reaction networks. In: Rondelez, Y., Woods, D. (eds.) DNA 2016. LNCS, vol. 9818, pp. 35–51. Springer, Cham (2016). doi:10.1007/978-3-319-43994-5_3
16. Cardelli, L., Kwiatkowska, M., Laurenti, L.: Stochastic analysis of chemical reaction networks using linear noise approximation. Biosystems **149**, 26–33 (2016)
17. Cardelli, L., Kwiatkowska, M., Whitby, M.: Chemical reaction network designs for asynchronous logic circuits. In: Rondelez, Y., Woods, D. (eds.) DNA 2016. LNCS, vol. 9818, pp. 67–81. Springer, Cham (2016). doi:10.1007/978-3-319-43994-5_5
18. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Comparing chemical reaction networks: a categorical and algorithmic perspective. In: LICS 2016, pp. 485–494. ACM (2016)
19. Cardelli, L., Tribastone, M., Tschaikowski, M., Vandin, A.: Symbolic computation of differential equivalences. ACM SIGPLAN Notices **51**(1), 137–150 (2016). (ACM)
20. Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., Brim, L.: Precise parameter synthesis for stochastic biochemical systems. Acta Inf. 1–35 (2016)
21. Chen, H.-L., Doty, D., Soloveichik, D.: Rate-independent computation in continuous chemical reaction networks. In: ITCS 2014, pp. 313–326. ACM (2014)
22. Csikász-Nagy, A., Cardelli, L., Soyer, O.S.: Response dynamics of phosphorelays suggest their potential utility in cell signalling. J. R. Soc. Interface **8**(57), 480–488 (2011)
23. Dalchau, N., Murphy, N., Petersen, R., Yordanov, B.: Synthesizing and tuning chemical reaction networks with specified behaviours. In: Phillips, A., Yin, P. (eds.) DNA 2015. LNCS, vol. 9211, pp. 16–33. Springer, Cham (2015). doi:10.1007/978-3-319-21999-8_2
24. Dunn, S.-J., Martello, G., Yordanov, B., Emmott, S., Smith, A.: Defining an essential transcription factor program for naive pluripotency. Science **344**(6188), 1156–1160 (2014)
25. Eggers, A., Fränzle, M., Herde, C.: SAT modulo ODE: a direct SAT approach to hybrid systems. In: Cha, S.S., Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 171–185. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88387-6_14
26. Eggers, A., Ramdani, N., Nedialkov, N.S., Fränzle, M.: Improving the SAT modulo ODE approach to hybrid systems analysis by combining different enclosure methods. Softw. Syst. Model. **14**(1), 121–148 (2015)
27. Eldar, A., Elowitz, M.B.: Functional roles for noise in genetic circuits. Nature **467**(7312), 167–173 (2010)
28. Ethier, S.N., Kurtz, T.G.: Markov Processes: Characterization and Convergence, vol. 282. Wiley, Hoboken (2009)
29. Gao, S., Avigad, J., Clarke, E.M.: δ-complete decision procedures for satisfiability over the reals. In: Gramlich, B., Miller, D., Sattler, U. (eds.) IJCAR 2012. LNCS, vol. 7364, pp. 286–300. Springer, Heidelberg (2012). doi:10.1007/978-3-642-31365-3_23
30. Gao, S., Kong, S., Clarke, E.M.: dReal: an SMT solver for nonlinear theories over the reals. In: Bonacina, M.P. (ed.) CADE 2013. LNCS (LNAI), vol. 7898, pp. 208–214. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38574-2_14

31. Gerasimou, S., Tamburrelli, G., Calinescu, R.: Search-based synthesis of probabilistic models for quality-of-service software engineering. In: ASE 2015, pp. 319–330 (2015)
32. Giacobbe, M., Guet, C.C., Gupta, A., Henzinger, T.A., Paixão, T., Petrov, T.: Model checking gene regulatory networks. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 469–483. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_47
33. Hardy, M.: Combinatorics of partial derivatives. Electron. J. Combin. **13**(1), 13 (2006)
34. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoret. Comput. Sci. **391**(3), 239–257 (2008)
35. Hoops, S., et al.: COPASI - a complex pathway simulator. Bioinformatics **22**(24), 3067–3074 (2006)
36. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
37. Koksal, A.S., Pu, Y., Srivastava, S., Bodik, R., Fisher, J., Piterman, N.: Synthesis of Biological Models from Mutation Experiments. In: POPL 2013, pp. 469–482. ACM (2013)
38. 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). doi:10.1007/978-3-642-22110-1_47
39. Kwiatkowska, M., Thachuk, C.: Probabilistic model checking for biology. In: Software Systems Safety, vol, 36, p. 165 (2014)
40. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. J. R. Soc. Interface **9**(72), 1470–1485 (2012)
41. Madsen, C., Shmarov, F., Zuliani, P.: BioPSy: an SMT-based tool for guaranteed parameter set synthesis of biological models. In: Roux, O., Bourdon, J. (eds.) CMSB 2015. LNCS, vol. 9308, pp. 182–194. Springer, Cham (2015). doi:10.1007/978-3-319-23401-4_16
42. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
43. Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Effcient Synthesis of Probabilistic Programs. In: PLDI 2014, pp. 208–217. ACM (2015)
44. Ouaknine, J., Worrell, J.: Some recent results in metric temporal logic. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 1–13. Springer, Heidelberg (2008). doi:10.1007/978-3-540-85778-5_1
45. Paoletti, N., Yordanov, B., Hamadi, Y., Wintersteiger, C.M., Kugler, H.: Analyzing and Synthesizing genomic logic functions. In: Biere, A., Bloem, R. (eds.) CAV 2014. LNCS, vol. 8559, pp. 343–357. Springer, Cham (2014). doi:10.1007/978-3-319-08867-9_23
46. Solar-Lezama, A., Jones, C.G., Bodik, R.: Sketching concurrent data structures. In: PLDI 2008, pp. 136–148. ACM (2008)
47. Solar-Lezama, A., Rabbah, R., Bodík, R., Ebcioğlu, K.: Programming by Sketching for bit-streaming programs. In: PLDI 2005, pp. 281–294. ACM (2005)
48. Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. In: ASPLOS 2006, pp. 404–415. ACM (2006)
49. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. U. S. A. **107**(12), 5393–5398 (2010)

50. Tung, V.X., Van Khanh, T., Ogawa, M.: raSAT: an SMT solver for polynomial constraints. In: Olivetti, N., Tiwari, A. (eds.) IJCAR 2016. LNCS (LNAI), vol. 9706, pp. 228–237. Springer, Cham (2016). doi:10.1007/978-3-319-40229-1_16
51. Van Kampen, N.G.: Stochastic Processes in Physics and Chemistry, Elsevier, vol. 1 (1992)
52. Yordanov, B., Kim, J., Petersen, R.L., Shudy, A., Kulkarni, V.V., Phillips, A.: Computational design of nucleic acid feedback control circuits. ACS Synth. Biol. **3**(8), 600–616 (2014)
53. Zimmer, C., Sahle, S.: Parameter estimation for stochastic models of biochemical reactions. J. Comput. Sci. Syst. Biol. **6**, 011–021 (2012)

The Journal of Systems & Software 143 (2018) 140–158

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Efficient synthesis of robust models for stochastic systems

Radu Calinescu[a], Milan Češka[b], Simos Gerasimou[*,a], Marta Kwiatkowska[c], Nicola Paoletti[d]

[a] Department of Computer Science, University of York, UK
[b] Faculty of Information Technology, Brno University of Technology, Czechie
[c] Department of Computer Science, University of Oxford, UK
[d] Department of Computer Science, Stony Brook University, USA

ARTICLE INFO

Keywords:
Software performance and reliability engineering
Probabilistic model synthesis
Multi-objective optimisation
Robust design

ABSTRACT

We describe a tool-supported method for the efficient synthesis of parametric continuous-time Markov chains (*p*CTMC) that correspond to *robust designs* of a system under development. The *p*CTMCs generated by our RObust DEsign Synthesis (RODES) method are resilient to changes in the system's operational profile, satisfy strict reliability, performance and other quality constraints, and are Pareto-optimal or nearly Pareto-optimal with respect to a set of quality optimisation criteria. By integrating sensitivity analysis at designer-specified tolerance levels and Pareto optimality, RODES produces designs that are potentially slightly suboptimal in return for less sensitivity—an acceptable trade-off in engineering practice. We demonstrate the effectiveness of our method and the efficiency of its GPU-accelerated tool support across multiple application domains by using RODES to design a producer-consumer system, a replicated file system and a workstation cluster system.

1. Introduction

Robustness is a key characteristic of both natural (Kitano, 2004) and human-made (Phadke, 1995) systems. Systems that cannot tolerate change are prone to frequent failures and require regular maintenance. As such, engineering disciplines like mechanical and electrical engineering treat robustness as a first-class citizen by designing their systems based on established tolerance standards (e.g. International Organization for Standardization, 2010; International Organization for Standardization, 2013). By comparison, software engineering is lagging far behind. Despite significant advances in software performance and reliability engineering (Balsamo et al., 2004; Bondy, 2014; Becker et al., 2009; Fiondella and Puliafito, 2016; Stewart, 2009; Woodside et al., 2014), the quality attributes of software systems are typically analysed for point estimates of stochastic system parameters such as component service rates or failure probabilities. Even the techniques that assess the sensitivity of quality attributes to parameter changes (e.g. Gokhale and Trivedi, 2002; Lo et al., 2005; Huang and Lyu, 2005; Kamavaram and Goseva-Popstojanova, 2003; Filieri et al., 2016) focus on the analysis of a given design at a time instead of systematically designing robustness into the system under development (SUD).

To address these limitations, we propose a tool-supported method for the efficient synthesis of parametric continuous-time Markov chains (*p*CTMCs) that correspond to robust SUD designs. Our RObust DEsign Synthesis (RODES) method generates sets of *p*CTMCs that:

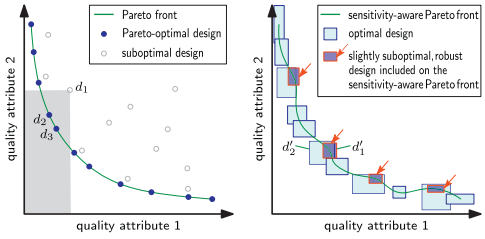
(i) are resilient to pre-specified *tolerances* in the SUD parameters, i.e., to changes in the SUD's operational profile;
(ii) satisfy strict performance, reliability and other quality constraints;
(iii) are Pareto-optimal or nearly Pareto optimal with respect to a set of quality optimisation criteria.

RODES comprises two steps. In the first step, the SUD design space is modelled as a *p*CTMC with discrete and continuous parameters corresponding to alternative system architectures and to ranges of possible values for the SUD parameters, respectively. In the second step, a multi-objective optimisation technique is used to obtain a set of low-sensitivity, Pareto-optimal or nearly Pareto-optimal SUD designs by fixing the discrete parameters (thus selecting specific architectures) and restricting the continuous parameters to bounded intervals that reflect the pre-specified tolerances. The designs that are slightly suboptimal have the advantage of a lower sensitivity than the optimal designs with similar quality attributes, achieving a beneficial compromise between optimality and sensitivity. A *sensitivity-aware Pareto dominance relation* is introduced in the paper to formally capture this trade-off.

Fig. 1 shows the differences between a traditional Pareto front, which corresponds to a fixed SUD operational profile, and a sensitivity-aware Pareto front generated by RODES, which corresponds to a SUD operational profile that can change within pre-specified bounds. Accordingly, the designs from the RODES sensitivity-aware Pareto front are bounded regions of quality-attribute values for the system. The size

* Corresponding author.
E-mail address: simos.gerasimou@york.ac.uk (S. Gerasimou).

https://doi.org/10.1016/j.jss.2018.05.013
Received 5 October 2017; Received in revised form 2 April 2018; Accepted 9 May 2018
Available online 16 May 2018



(a) Traditional Pareto front: design d_1 is suboptimal because designs d_2 and d_3 from the shaded area have better (i.e., lower) quality attributes than d_1

(b) Sensitivity-aware Pareto front: the slightly suboptimal design d'_1 belongs to the front because it has a much lower sensitivity than optimal design d'_2

Fig. 1. Traditional Pareto front (a) versus sensitivity-aware Pareto front (b) for two quality attributes that require minimisation (e.g., response time and probability of failure).

and shape of these regions convey the sensitivity of the synthesised designs to parameter changes within the pre-specified tolerances. Small quality-attribute regions correspond to particularly robust designs that cope with variations in the system parameters without exposing users to significant changes in quality attributes. These designs require reduced maintenance, and can be implemented using high-variability components that are cheaper to develop or obtain off-the-shelf than low-variability components. Large quality-attribute regions from a RODES Pareto front—while still the most robust for the quality attribute trade-offs they correspond to—are associated with designs that are sensitive to SUD parameters variations. These designs may involve high maintenance and/or development costs, so they should only be used if justified by their other characteristics (e.g. desirable quality attribute trade-offs).

To the best of our knowledge, RODES is the first solution that integrates multi-objective stochastic model synthesis and sensitivity analysis into an end-to-end, tool-supported design method. As we show in detail in Section 7, the existing research addresses the challenges associated with design synthesis (e.g. Gerasimou et al., 2015; Martens et al., 2010) and sensitivity analysis (e.g. Gokhale and Trivedi, 2002; Lo et al., 2005; Huang and Lyu, 2005; Kamavaram and Goseva-Popstojanova, 2003; Filieri et al., 2016) separately. The main contributions of our paper are:

1. The extension of the notion of *parameter tolerance* from other engineering disciplines for application to software architecture.
2. The definitions of the parametric Markov chain synthesis problem and of the sensitivity-aware Pareto dominance relation for the synthesis of robust models for stochastic systems.
3. The RODES method for the generation of sensitivity-aware Pareto fronts by integrating multi-objective probabilistic model synthesis and precise *p*CTMC parameter synthesis.
4. A GPU-accelerated tool that implements the RODES method and is available preinstalled on an easy-to-use VirtualBox instance from our project website https://www.github.com/gerasimou/RODES/wiki.
5. A repository of case studies demonstrating the successful application of RODES to a replicated file system used by Google's search engine, a cluster availability management system, and a producer-consumer system.

These contributions significantly extend our conference paper on robust model synthesis (Calinescu et al., 2017a) and the prototype probabilistic model synthesis tool (Calinescu et al., 2017b) in several ways. First, we provide a more detailed description of our solution, including a running example and new experimental results. Second, we greatly improve the scalability of RODES by integrating the GPU-accelerated analysis of candidate designs into our prototype tool (Calinescu et al., 2017b). Third, we extend the experimental evaluation to demonstrate the impact of the GPU acceleration. Finally, we present an additional case study in which we apply RODES to a producer-consumer system, and we use the systems and models from our experiments to assemble a repository of case studies available on our project website.

The remainder of the paper is organised as follows. Section 2 introduces the RODES design-space modelling language and the formalism to specify quality constraints and optimisation criteria. Section 3 defines the sensitivity-aware dominance relation and introduces the parametric Markov chain synthesis problem. We then present our method for synthesising robust designs in the form of a sensitivity-aware Pareto set, and the GPU-accelerated tool RODES implementing the method in Sections 4 and 5, respectively. Finally, we evaluate our method within three case studies in Section 6, discuss related work in Section 7, and conclude the paper with a summary and future work in Section 8.

2. Modelling and specification language for probabilistic systems

This section formalises three key elements underpinning the formulation of the robust design problem: 1) the modelling of the design space of a SUD, 2) the specification of quality attributes and requirements, and 3) the sensitivity of a design.

2.1. Design space modelling

We use a *parametric continuous-time Markov chain* (*p*CTMC) to define the design space of a SUD. To this end, we extend the original *p*CTMC definition (Han et al., 2008), where only real-valued parameters determining the transition rates of the Markov chain are considered, and assume that a *p*CTMC also includes discrete parameters affecting its state space. Our definition captures the need for both discrete parameters encoding architectural structural information (e.g. by selecting between alternative implementations of a software component) and continuous parameters encoding configurable aspects of the system (e.g. network latency or throughput). As such, a candidate system design corresponds to a fixed discrete parameter valuation and to continuous parameter values from a (small) region.

Definition 1 (*pCTMC*). Let K be a finite set of real-valued parameters such that the domain of each parameter $k \in K$ is a closed interval $[k^{\perp}, k^{\top}] \subset \mathbb{R}$, and D a finite set of discrete parameters such that the domain of each parameter $d \in D$ is a set $T^d \subset \mathbb{Z}$. Let also $\mathscr{P} = \times_{k \in K}[k^{\perp}, k^{\top}]$ and $\mathscr{D} = \times_{d \in D} T^d$ be the continuous and the discrete *parameter spaces* induced by K and D, respectively. A *p*CTMC over K and D is a tuple

$$\mathscr{C}(\mathscr{P}, \mathscr{D}) = (\mathscr{D}_S, \mathscr{D}_{init}, \mathscr{D}_{\mathbf{R}}, L), \tag{1}$$

where, for any discrete parameter valuation $q \in \mathscr{D}$:

- $\mathscr{D}_S(q) = S$ is a finite set of *states*, and $\mathscr{D}_{init}(q) \in S$ is the initial state;
- $\mathscr{D}_{\mathbf{R}}(q)$: $S \times S \to \mathbb{R}[K]$ is a parametric rate matrix, where $\mathbb{R}[K]$ denotes the set of polynomials over the reals with variables in K;
- $L(q)$: $S \to 2^{AP}$ is a labelling function mapping each state $s \in S$ to the set $L(q)(s) \subseteq AP$ of atomic propositions that hold true in s.

A *p*CTMC $C(\mathscr{P}, \mathscr{D})$ describes the uncountable set of continuous-time Markov chains (CTMCs) $\{\mathscr{C}(p, q) \mid p \in \mathscr{P} \wedge q \in \mathscr{D}\}$, where each $\mathscr{C}(p, q) = (\mathscr{D}_S(q), \mathscr{D}_{init}(q), \mathbf{R}(p, q), L(q))$ is the instantiated CTMC with transition matrix $\mathbf{R}(p, q)$ obtained by replacing the real-valued parameters in $\mathscr{D}_{\mathbf{R}}(q)$ with their valuation in p.

In our approach we operate with *p*CTMCs expressed in a high-level modelling language extending the PRISM language (Kwiatkowska et al., 2011) which models a system as the parallel composition of a set of

modules. The state of a module is encoded by a set of finite-range local variables, and its state transitions are defined by probabilistic guarded commands that change these variables, and have the general form:

$$[\text{action}]\text{guard} \rightarrow e_1 : update_1 + \cdots + e_n : update_n \quad (2)$$

In this command, *guard* is a Boolean expression over all model variables. If the guard evaluates to true, the arithmetic expression e_i, $1 \leq i \leq n$, gives the rate with which the $update_i$ change of the module variables occurs. When *action* is present, all modules comprising commands with this action have to synchronise (i.e., to carry out one of these commands simultaneously) and the resulting rate of such synchronised commands is equal to the multiplication of the individual command rates. Atomic propositions are encoded with label expressions of the form:

$$\text{label ``id'' } = b \quad (3)$$

where id is a string that identifies the atomic proposition and b is a Boolean expression over the state variables.

We extend the PRISM language with the following constructs (adopted from Gerasimou et al., 2015) for specifying the parameters $k \in K$ and $d \in D$ from Definition 1:

$$\begin{aligned} &\text{evolve double } k\ [min.\ max] \\ &\text{evolve int } d\ [min.\ max] \\ &\text{evolve module } ComponentName \end{aligned} \quad (4)$$

where $N > 1$ instances of the last construct (with the same component name) define N alternative architectures for a component, introducing the index (between 1 and N) of the selected architecture as an implicit discrete parameter.

As per Definition 1, continuous parameters can only appear in the transition rates (expressions $e_1, \ldots, e_n$ above).

Explicit discrete variables (declared using evolve int) can instead appear in any type-consistent expression.

The translation of models expressed in the extended PRISM language into the corresponding *p*CTMC is fully automatic and follows the probabilistic guarded command semantics described above. The discrete state space $\mathcal{Q}$ results from all possible valuations of explicit discrete variables and implicit discrete variables (different implementations of a module). For a fixed valuation $q \in \mathcal{Q}$, the parametric PRISM model describes a fixed set of modules with a fixed set of finite-range variables, and thus the state space $\mathcal{S}_S(q)$ is given by the Cartesian product of the value ranges for these variables. In contrast, q determines also the parametric rate matrix $\mathcal{S}_{\mathbf{R}}(q)$ and atomic propositions $L(q)$, as q can affect guards and updates of PRISM commands, as well as label expressions.

Example 1 (*Producer-consumer model*). As a running example, we consider a simple producer-consumer system with a two-way buffering, illustrated in Fig. 2. The *p*CTMC PRISM model, extended with the evolvable constructs from Definition 4 is shown in Fig. 3. The system comprises a producer generating requests with rate p_rate. Each request is being transferred to a consumer either via a slow buffer or via a fast buffer with probabilities 0.6 and 0.4, respectively (lines 14 and 15 in Fig. 3). The fast buffer transmits requests to the consumer faster than the slow buffer, but it has smaller capacity and is less reliable, as it loses packets with a 5% probability (line 20).We consider two alternative designs of the producer-consumer model that differ in the way that the two buffers manage the pending requests. More specifically we consider

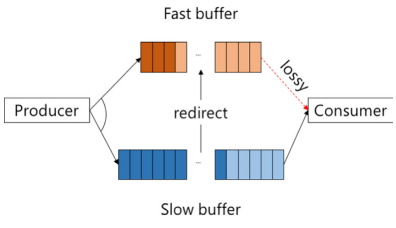


Fig. 2. Two-way producer-consumer system.

```
1  ctmc
   // buffer capacities
2  const int slow_max = 31;
3  const int fast_max = 21;
4  cons double p_rate = 40;          //request production rate
5  const double s_rate = 30;         //request transmission rate
   // packet transmission rates
   // trans. rate for the fast buffer is r_slow_rate+delta_rate
6  evolve double r_slow_rate [5..30]; //slow buffer
7  evolve double delta_rate [0..30];  //fast buffer
8  const double c_rate = 40;         //packet consumption rate

// no redirection
9  evolve module Buffer
     // is request sent to fast/slow buffer?
10   fast : [0..1] init 0;
11   slow : [0..1] init 0;
     // buffers
12   buffer_s : [0..slow_max] init 0;
13   buffer_f : [0..fast_max] init 0;
     // has consumer received the packet?
13   consumer : [0..1] init 0;
     //produce
14   [] (fast=0) → p_rate*0.4 : (fast'=1);
15   [] (slow=0) → p_rate*0.6 : (slow'=1);
     //send
16   [] (slow=1) & (buffer_s<slow_max) → s_rate :
                                (slow'=0) & (buffer_s' = buffer_s +1);
17   [] (fast=1) & (buffer_f<fast_max) → s_rate :
                                (fast'=0) & (buffer_f' = buffer_f +1);
     //receive
18   [] (consumer=0) & (buffer_s > 0) → r_slow_rate :
                                (consumer'=1) & (buffer_s' = buffer_s - 1);
19   [] (consumer=0) & (buffer_f > 0) → (r_slow_rate+delta_rate)*0.95 :
                                (consumer'=1) & (buffer_f' = buffer_f - 1);
     // fast buffer loses the packet
20   [lost] (consumer=0) & (buffer_f > 0) → (r_slow_rate+delta_rate)*0.05 :
                                (buffer_f' = buffer_f - 1);
     //consume
21   [consume] (consumer=1) → c_rate : (consumer'=0);
22 endmodule

// redirection
23 evolve module Buffer
     ⋮
     //send
24   [] (slow=1) & (buffer_s<slow_max) → s_rate *(1-buffer_s/(10.0*slow_max)) :
                                (slow'=0) & (buffer_s' = buffer_s +1);
25   [] (slow=1) & (buffer_s>0) & (buffer_f<fast_max) →
                                s_rate *buffer_s/(10.0*slow_max) :
                                    (slow'=0) & (buffer_f' = buffer_f +1);
26   [] (fast=1) & (buffer_f<fast_max) →
                                s_rate : (fast'=0) & (buffer_f' = buffer_f +1);
     ⋮
27 endmodule
```

Fig. 3. PRISM-RODES encoding of *p*CTMC model of a producer-consumer system with two-way buffering and redirection. In the second module only the commands that differ from the first module are reported.

1. a no-redirection design in which once a request is sent to either buffer, the packet is transmitted by that buffer to the consumer (lines 9–22);
2. a redirection design that enables the slow buffer to transmit requests to the fast buffer with a probability proportional to its occupancy (lines 23–27). In particular, redirection is disabled when the slow buffer is empty and has maximum rate when it is full and is equal to s_rate/10, where s_rate is the request transmission rate without redirection.In addition to these two alternative designs, the model has two continuous parameters, the packet transmission rate for the

slow buffer, r_slow_rate, and delta_rate, i.e. the transmission rate difference between fast and slow buffers. Notably, the rate of packet loss by the fast buffer is proportional to its transmission rate, meaning that the buffer becomes less reliable as its rate increases.We formally capture the above system model with its continuous parameters and alternative designs by a *p*CTMC $\mathscr{C}_{PC}(\mathscr{P}, \mathscr{Q})$, where $\mathscr{P} = [5, 30] \times [0, 30]$ defines the domains for the continuous parameters r_slow_rate, and delta_rate, respectively, and $\mathscr{Q} = \{1, 2\}$ defines the domain for the discrete parameter corresponding to the two alternative designs (i.e. modules).

Definition 2 (*Candidate design*). A candidate design of the *p*CTMC $\mathscr{C}(\mathscr{P}, \mathscr{Q})$ from (1) is a *p*CTMC

$$\mathscr{C}(\mathscr{P}', \{q\}) = (\mathscr{D}'_S, \mathscr{D}'_{init}, \mathscr{D}'_{\mathbf{R}}, L') \tag{5}$$

where $\mathscr{P}' = \times_{k\in K}[k'^{\perp}, k'^{\top}] \subseteq \mathscr{P}$, $q \in \mathscr{Q}$, $\mathscr{D}'_S(q) = \mathscr{D}_S(q)$, $\mathscr{D}'_{\mathbf{R}}(q) = \mathscr{D}_{\mathbf{R}}(q)$, $\mathscr{D}'_{init}(q) = \mathscr{D}_{init}(q)$ and $L'(q) = L(q)$. The *tolerance* of the candidate design with respect to the real-valued parameter $k \in K$ is defined as

$$\gamma_k = \frac{k'^{\top} - k'^{\perp}}{2(k^{\top} - k^{\perp})}, \tag{6}$$

in line with the fact that the design restricts the value domain of k to the interval $[\overline{k} - \gamma_k(k^{\top} - k^{\perp}), \overline{k} + \gamma_k(k^{\top} - k^{\perp})]$, $\overline{k} = \frac{k'^{\perp} + k'^{\top}}{2}$.[1] For convenience, we will use the shorthand notation $\mathscr{C}(\mathscr{P}', q) \equiv \mathscr{C}(\mathscr{P}', \{q\})$ in the rest of the paper.

Example 2 (*Candidate design*). Consider the *p*CTMC $C_{PC}(\mathscr{P}, \mathscr{Q})$ from Example 1 and a single tolerance value $\gamma = 0.005$ for both continuous parameters r_slow_rate and delta_rate. By (6), candidate designs have continuous parameter ranges of size $2\gamma(k^{\top} - k^{\perp}) = 0.25$ for r_slow_rate and of size 0.3 for delta_rate. Two examples of valid candidate designs for the second module (redirection), obtained using our RODES synthesis method (see also results in Fig. 7), are *p*CTMCs $d_1 = \mathscr{C}_{PC}(\mathscr{P}', 2)$ and $d_2 = \mathscr{C}_{PC}(\mathscr{P}'', 2)$ where $\mathscr{P}' = [15.02, 15.27] \times [1.93, 2.23]$, $\mathscr{P}'' = [13.2, 13.45] \times [3.51, 3.81]$. The *p*CTMCs $d_3 = \mathscr{C}_{PC}(\mathscr{P}'', 1)$ with is instead a valid candidate design for the first module (no redirection).

2.2. Quality attribute specification and requirements

We specify quality attributes over *p*CTMCs-defined design spaces using *continuous stochastic logic* (CSL) extended with reward operators (Kwiatkowska et al., 2007). Our focus is on timed properties of *p*CTMCs expressed by the time-bounded fragment of CSL with rewards comprising state formulae (Φ) and path formulae (ϕ) with the syntax:

$$\begin{aligned} &\Phi ::= \text{true} \mid a \mid \neg\Phi \mid \Phi \wedge \Phi \mid P_{\sim r}[\phi] \mid R_{\sim r}[C^{\leq t}] \\ &\phi ::= X\ \Phi \mid \Phi\ U^{I}\Phi \end{aligned}, \tag{7}$$

where a is an atomic proposition evaluated over states, $\sim \in \{<, \leq, \geq, >\}$ is a relational operator, r is a probability ($r \in [0, 1]$) or reward ($r \in \mathbb{R}_{\geq 0}$) threshold[2], $t \in \mathbb{R}_{\geq 0}$ is a time bound, and $I \subseteq \mathbb{R}_{\geq 0}$ is a bounded time interval. The 'future' operator, F, and 'globally' operator, G, are derived from U in the standard way[3]. As briefly discussed in Section 4.2, our approach can be extended to unbounded CSL.

Traditionally, the CSL semantics is defined for CTMCs using a satisfaction relation $\models$. Intuitively, a state $s \models P_{\sim r}[\phi]$ iff the probability of the set of paths starting in s and satisfying ϕ meets $\sim r$. A path $\omega = s_0 t_0 s_1 t_1 \ldots$ satisfies the formula $\Phi\ U^{I}\ \Psi$ iff there exists a time $t \in I$ such that $(\omega@t \models \Psi \wedge \forall t' \in [0, t).\omega@t' \models \Phi)$, where $\omega@t$ denotes the state in ω at time t. A state $s \models R_{\sim r}[C^{=t}]$ iff the expected rewards over the path starting in s and cumulated within t time units satisfies $\sim r$, where the rates with which reward is acquired in each state and the reward acquired at each transition are defined by a *reward structure*.

In line with our previous work (Češka et al., 2017), we introduce a *satisfaction function* $\Lambda_{\phi}: \mathscr{P} \times \mathscr{Q} \rightarrow [0, 1]$ that quantifies how the satisfaction probability associated with a path CSL formula ϕ relates to the parameters of a *p*CTMC $\mathscr{C}(\mathscr{P}, \mathscr{Q})$, where, for any $(p, q) \in \mathscr{P} \times \mathscr{Q}$, $\Lambda_{\phi}(p, q)$ is the probability that ϕ is satisfied by the set of paths from the initial state $\mathscr{D}_{init}(q)$ of the instantiated CTMC $\mathscr{C}(p, q)$. The satisfaction function for reward CSL formulae is defined analogously.

Quality requirements. We assume that the quality requirements of a SUD with design space given by a *p*CTMC $\mathscr{C}(\mathscr{P}, \mathscr{Q})$ are defined in terms of:

1) A finite set of objective functions $\{f_i\}_{i\in I}$ corresponding to quality attributes of the system and defined in terms of a set of CSL path formulas $\{\phi_i\}_{i\in I}$, such that for any $i \in I$ and $(p, q) \in \mathscr{P} \times \mathscr{Q}$,

$$f_i(\mathscr{C}(p, q)) = \Lambda_{\phi_i}(p, q); \tag{8}$$

2) A finite set of Boolean constraints $\{c_j\}_{j\in J}$ corresponding to the set of CSL path formulas $\{\psi_j\}_{j\in J}$ and thresholds $\{\sim_j r_j\}_{j\in J}$, such that for any $j \in J$ and $(p, q) \in \mathscr{P} \times \mathscr{Q}$,

$$c_j(\mathscr{C}(p, q)) \Leftrightarrow \Lambda_{\psi_j}(p, q) \sim_j r_j. \tag{9}$$

Note that quality requirements (8) and (9) are defined over (non-parametric) CTMCs, but, in order to compare candidate designs with respect to some objective function, we need to interpret quality requirements over *p*CTMCs. Indeed, due to the continuous parameter space, a single candidate design induces an infinite number of objective function values, from which the designer must choose a representative value. For a candidate design $\mathscr{C}(\mathscr{P}', q)$ and objective f_i, this is typically identified as one of the minimum, maximum and mid-range value of $f_i(\mathscr{C}(p, q))$ over all $p \in \mathscr{P}'$, as illustrated in Table 1.

On the other hand, constraints have a unique interpretation because they must be met for any parameter value of a candidate design. Formally, for candidate design $\mathscr{C}(\mathscr{P}', q)$ and constraint c_j, we define

$$c_j(\mathscr{C}(\mathscr{P}', q)) \Leftrightarrow \forall\, p \in \mathscr{P}'.\ c_j(\mathscr{C}(p, q)).$$

Without loss of generality, we will assume that all objective functions $\{f_i\}_{i\in I}$ in Sections 3 and 4 should be minimised and that all thresholds $\{\sim_j r_j\}_{j\in J}$ are upper bounds of the form of $\leq r_j$.

Example 3 (*Quality requirements*). Below we define quality requirements for the producer-consumer model of Example 1. We consider two maximisation objectives and one constraint:

f_1: $R\{\text{``consume''}\}_{=?}[C^{<=25}]$, a cumulative transition reward describing the number of requests transferred to the consumer within 25 time units (line 21 in Fig. 3);
f_2: $P_{=?}[G[20, 25]((\text{buffers} \geq \text{slow\_max}/2)\&(\text{bufferf} \geq \text{fast\_max}/2))]$, which calculates the probability that the utilisation of both buffers is at least 50% of their respective capacities;
c_1: $R\{\text{``lost''}\}_{\leq 10}[C^{<=25}]$, a cumulative transition reward that limits the number of packets lost within 25 time units (line 20 in Fig. 3).With these quality requirements, we seek to maximise the system throughput (objective f_1), expressed as the number of requests transferred to the consumer, and also to maximize the probability that both buffers are sufficiently utilised after an initial period (objective f_2). Finally, constraint c_1 imposes a reliability requirement by restricting the number of packets lost to be less than 10 within 25 time units of operation.

[1] In other words, the tolerance of parameter k, γ_k, measures the extent to which k can be perturbed from its reference (midpoint) value.

[2] For simplicity, we use $\sim r$ to denote the threshold for both probability and reward quality attributes.

[3] $P_{\sim r}[F^{I}\Phi] = P_{\sim r}[\text{true}\ U^{I}\Phi]$ and $P_{\sim r}[G^{I}\Phi] = P_{\not\sim 1-r}[F^{I}\neg\Phi]$.

Table 1
Alternative definitions for objective functions $\{f_i\}_{i\in I}$ over candidate designs.

Type	Notation	Definition
Lower bound	$f_i^{\perp}(\mathscr{C}(\mathscr{P}', q))$	$\inf_{p\in\mathscr{P}'}\Lambda_{\phi_i}(p, q)$
Upper bound	$f_i^{\top}(\mathscr{C}(\mathscr{P}', q))$	$\sup_{p\in\mathscr{P}'}\Lambda_{\phi_i}(p, q)$
Mid-range	$f_i^{*}(\mathscr{C}(\mathscr{P}', q))$	$(f_i^{\perp}(\mathscr{C}(\mathscr{P}', q)) + f_i^{\top}(\mathscr{C}(\mathscr{P}', q)))/2$

2.3. Sensitivity of candidate designs

Quantifying the sensitivity of candidate designs is a crucial step in our robust synthesis method. Intuitively, the sensitivity of a design $\mathscr{C}(\mathscr{P}', q)$ captures how the objective functions $\{f_i\}_{i\in I}$ change in response to variations in the continuous parameters $k\in K$. The variation of each objective f_i is measured by the length of the interval $[f_i^{\perp}(\mathscr{C}(\mathscr{P}', q)), f_i^{\top}(\mathscr{C}(\mathscr{P}', q)]$, describing the range of admissible values for f_i and $\mathscr{C}(\mathscr{P}', q)$ (cf. Table 1). The degree of variation for multiple objectives is given by the product of interval lengths, i.e., the volume of the corresponding quality-attribute region. The sensitivity takes also into account the size of the underlying parameter region, in order to account for designs with different tolerance values. For instance, a design with a large quality-attribute volume and high tolerance (large parameter region volume) must be considered *more robust* (less sensitive) than another design with comparable quality-attribute volume but lower tolerance.

Definition 3 (*Sensitivity*). For a set of objective functions $\{f_i\}_{i\in I}$ and tolerances $\{\gamma_k\}_{k\in K}$, the *sensitivity* of a feasible design $\mathscr{C}(\mathscr{P}', q)$ is defined as the volume of its quality-attribute region over the volume of $\mathscr{P}'$:

$$sens(\mathscr{C}(\mathscr{P}', q)) = \frac{\prod_{i\in I}(f_i^{\top}(\mathscr{C}(\mathscr{P}', q)) - f_i^{\perp}(\mathscr{C}(\mathscr{P}', q)))}{\prod_{k\in K} 2\gamma_k(k^{\top} - k^{\perp})}. \tag{10}$$

Example 4 (*Sensitivity*). Consider the candidate designs d_1, d_2, d_3 with tolerance $\gamma = 0.005$ from Example 2, and the objective functions f_1 (number of "consumed" packets) and f_2 (probability of buffers being sufficiently used) introduced in Example 3. Assume the following ranges for f_1 and f_2:

$$\begin{aligned}
[f_1^{\perp}(d_1), f_1^{\top}(d_1)] &= [416.94, 439.65]\\
[f_2^{\perp}(d_1), f_2^{\top}(d_1)] &= [0.8977, 0.9809]\\
[f_1^{\perp}(d_2), f_1^{\top}(d_2)] &= [407.11, 423.10]\\
[f_2^{\perp}(d_2), f_2^{\top}(d_2)] &= [0.891, 0.9621]\\
[f_1^{\perp}(d_3), f_1^{\top}(d_3)] &= [384.81, 413.09]\\
[f_2^{\perp}(d_3), f_2^{\top}(d_3)] &= [0.7501, 0.8225].
\end{aligned}$$

Recall that the three designs have the same tolerance, thus yielding the same parameter region volume

$$\prod_{k\in K} 2\gamma_k(k^{\top} - k^{\perp}) = 0.25{\cdot}0.3 = 0.075$$

The resulting sensitivities are:

$$\begin{aligned}
sens(d_1) &= (439.65 - 416.94)(0.9809 - 0.8977)/0.075 = 25.19\\
sens(d_2) &= (423.10 - 407.11)(0.9621 - 0.891)/0.075 = 15.16\\
sens(d_3) &= (413.09 - 384.81)(0.8225 - 0.7501)/0.075 = 27.3
\end{aligned}$$

indicating that d_2 is the most robust design (with the smallest sensitivity value). The three designs can be visualised in the *quality-attribute space* (i.e. the objective space), as shown in Fig. 4, providing a direct and intuitive way to assess robustness.

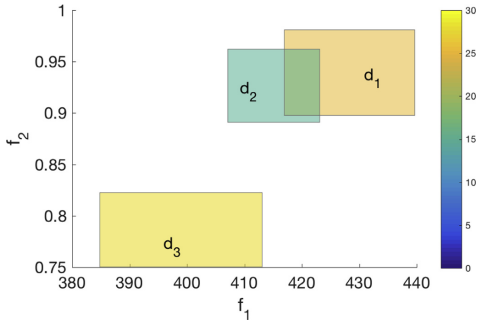


Fig. 4. Candidate designs of Example 4 represented in the quality-attribute space and coloured by sensitivity. Designs d_1 and d_2 were synthesised using RODES (full results are reported in Fig. 7 on a different scale).

3. Sensitivity-aware Pareto dominance relation

In this section, we introduce a novel dominance relation that adequately captures tradeoffs between the sensitivity and optimality of candidate designs with respect to given quality requirements, and that enables to formulate the robust design problem as an optimisation problem.

Consider a system with design space $\mathscr{C}(\mathscr{P}, \mathscr{Q})$, quality requirements given by objective functions $\{f_i\}_{i\in I}$ and constraints $\{c_j\}_{j\in J}$, and designer-specified tolerances $\{\gamma_k\}_{k\in K}$ for the continuous parameters of the system. Also, let $\mathscr{F}$ be the set of feasible designs for the system (i.e., of candidate designs that meet the tolerances $\{\gamma_k\}_{k\in K}$ and satisfy the constraints $\{c_j\}_{j\in J}$):

$$\begin{aligned}
\mathscr{F} = \{\mathscr{C}(\mathscr{P}', q) \mid \mathscr{P}' = \mathsf{X}_{k\in K}[k'^{\perp}, k'^{\top}] \subset \mathscr{P} \wedge q \in \mathscr{Q} \wedge\\
\forall\, k \in K.\ k'^{\top} - k'^{\perp} = 2\gamma_k(k^{\top} - k^{\perp}) \wedge \forall\, j \in J.\ c_j(\mathscr{C}(\mathscr{P}', q))\}.
\end{aligned} \tag{11}$$

Definition 4. A sensitivity-aware Pareto dominance relation over a feasible design set $\mathscr{F}$ and a set of minimisation objective functions $\{f_i\}_{i\in I}$ is a relation $\prec \subset \mathscr{F}\times\mathscr{F}$ such that for any feasible designs $d, d'\in\mathscr{F}$

$$\begin{aligned}
&d \prec d' \Leftrightarrow\\
&(\forall\, i\in I.\ f_i(d) \le f_i(d') \wedge \exists\, i\in I.\ (1+\epsilon_i) f_i(d) < f_i(d')) \vee\\
&(\forall\, i\in I.\ f_i(d) \le f_i(d') \wedge \exists\, i\in I.\ f_i(d) < f_i(d') \wedge\\
&\qquad sens(d) \le sens(d')).
\end{aligned} \tag{12}$$

where the objective functions $\{f_i\}_{i\in I}$ are calculated using one of the alternative definitions from Table 1 and $\epsilon_i \ge 0$ are sensitivity-awareness parameters.

The parametric Markov chain synthesis problem consists of finding the Pareto-optimal set *PS* of candidate designs (5) (i.e. *p*CTMCs) with tolerances $\{\gamma_k\}_{k\in K}$ that satisfy the constraints $\{c_j\}_{j\in J}$ and are non-dominated with respect to the objective functions $\{f_i\}_{i\in I}$ and the sensitivity-aware dominance relation '$\prec$':

$$PS = \{\mathscr{C}(\mathscr{P}', q)\in\mathscr{F} \mid \nexists\, \mathscr{C}(\mathscr{P}'', q')\in\mathscr{F}.\ \mathscr{C}(\mathscr{P}'', q')\prec\mathscr{C}(\mathscr{P}', q)\}, \tag{13}$$

Before discussing the rationale for this definition, we show that the sensitivity-aware Pareto dominance relation is a strict order like the classical Pareto dominance.

Theorem 1. *The sensitivity-aware Pareto dominance relation is a strict order.*

Proof. See Appendix A □

The classical Pareto dominance definition can be obtained by setting

$\epsilon_i = 0$ for all $i \in I$ in (12). When $\epsilon_i > 0$ for some $i \in I$, dominance with respect to quality attribute i holds in our generalised definition in two scenarios:

1) when the quality attribute has a much lower value for the dominating design, i.e. $(1+\epsilon_i)f_i(d) < f_i(d')$;
2) when in addition to a (slightly) lower quality attribute value, i.e. $f_i(d) < f_i(d')$, the sensitivity of the dominating design is no worse than that of the dominated design, i.e. $sens(d) \leq sens(d')$.

These scenarios are better aligned with the needs of designers than those obtained by using sensitivity as an additional optimisation criterion, which induces Pareto fronts comprising many designs with low sensitivity but unsuitably poor quality attributes. Similarly, each objective function definition from Table 1 captures specific needs of real-world systems. Thus, using the "upper bound" definition ($f_i^{\top}$) in (12) supports the synthesis of *conservative designs* by comparing competing designs based on the worst-case values of their quality attributes. This is suitable when the worst-case performance, reliability, etc. must be specified for a system, e.g. in its service-level agreement. In contrast, the "lower bound" definition from Table 1 ($f_i^{\perp}$) can be used when design selection must be based on the best expected quality values of a system. Finally, the "mid-range" definition ($f_i^{\star}$) may be useful—in conjunction with the actual sensitivity (10)—to compare and select designs based on their reference midpoint quality values.

Importantly, for $\epsilon_i > 0$ our generalised definition induces Pareto fronts comprising designs with non-optimal (in the classical sense) objective function values, but with low sensitivity. We call such designs *sub-optimal robust*. Thus, ϵ_i can be finely tuned to sacrifice objective function optimality (slightly) for better robustness. Below we formally characterize the set of robust sub-optimal designs and provide an example of the sensitivity-aware dominance relation.

Definition 5 (*Sub-optimal robust design*). Let *PS* be a Pareto-optimal set defined as per (13). A design $d' \in PS$ is called robust sub-optimal if $\exists d \in PS$ s.t.:

$$(\forall i \in I.\ f_i(d) \leq f_i(d')) \wedge \exists i \in I.\ f_i(d) < f_i(d'))$$

Example 5 (*Sensitivity-aware Pareto dominance relation*). Consider the quality-attribute regions of Fig. 4 induced by designs d_1, d_2, d_3 of the producer-consumer model introduced in Examples 1–4, and the objective functions defined as $f_i = f_i^{\perp}$ for $i \in 1, 2$. Visually, $f_i^{\perp}$ corresponds to the lower-left corners of the regions in Fig. 4. Since we maximize both objectives, for clarity, we report below the dominance relation for maximisation:

$$\begin{aligned} &d \succ d' \Leftrightarrow \\ &(\forall i \in I.\ f_i(d) \geq f_i(d') \wedge \exists i \in I.\ f_i(d) > (1+\epsilon_i) f_i(d')) \vee \\ &(\forall i \in I.\ f_i(d) \geq f_i(d') \wedge \exists i \in I.\ f_i(d) > f_i(d') \wedge \\ &\qquad\qquad sens(d) \leq sens(d')). \end{aligned}$$

The designs d_1, d_2, d_3 have identical parameter tolerances and thus, same parameter space volume V. We have that $d_1 \succ d_2 \succ d_3$ when $\epsilon_1 = \epsilon_2 = 0$ (classical dominance) because for $i = 1, 2$, $f_i^{\perp}(d_1) > f_i^{\perp}(d_2), f_i^{\perp}(d_3)$. Further, we have that $d_1 \neg \prec d_2$ when $\epsilon_1 = \epsilon_2 = 0.05$, implying that d_2 is robust sub-optimal, i.e., is retained in the sensitivity-aware Pareto-optimal set, because $f_1^{\perp}(d_1) \neg > 1.05 \cdot f_1^{\top}(d_2)$, $f_2^{\perp}(d_1) \neg > 1.05 \cdot f_2^{\perp}(d_2)$, and $sens(d_1) \neg \leq sens(d_2)$. Design d_3 is not included in the front (d_1, $d_2 \succ d_3$) because $f_i^{\perp}(d_1), f_i^{\perp}(d_2) > 1.05 \cdot f_i^{\perp}(d_3)$ for $i = 1, 2$.

4. Synthesis of sensitivity-aware Pareto sets

In this section, we describe our method for computing sensitivity-aware Pareto sets. The method employs genetic multi-objective optimisation algorithms for generating candidate designs and a precise parameter analysis of *p*CTMCs for evaluating the candidate designs. We start with a method overview, then we describe the two components the method builds on.

```
1:  function Synthesis(C(P,Q), {f_i}_{i∈I}, {c_j}_{j∈J}, {γ_k}_{k∈K})
2:      PS ← ∅
3:      while ¬Terminate(C(P,Q), PS) do
4:          CD ← CandidateDesigns(C(P,Q), {γ_k}_{k∈K}, PS)
5:          for all d ∈ CD do
6:              ({f_{i,d}^⊤}_{i∈I}, {f_{i,d}^⊥}_{i∈I}, {c_{j,d}}_{j∈J}) ←
                        AnalyseDesign(d, {f_i}_{i∈I}, {c_j}_{j∈J})
7:              if ⋀_{j∈J} c_{j,d} then
8:                  dominated = false
9:                  for all d' ∈ PS do
10:                     if d' ≺ d then dominated = true; break
11:                     if d ≺ d' then PS = PS \ {d'}
12:                 end for
13:                 if ¬dominated then PS = PS ∪ {d}
14:             end if
15:         end for
16:     end while
17:     return PS
18: end function
```

Algorithm 1. Parametric Markov chain synthesis.

4.1. Method overview

Computing the Pareto-optimal design set (13) using exhaustive analysis is very expensive and requires a significant amount of computational resources as the design space $\mathscr{C}(\mathscr{P}, \mathscr{Q})$ is extremely large due to its real-valued parameters. Also, every candidate design $\mathscr{C}(\mathscr{P}', q)$ consists of an infinite set of CTMCs that cannot all be analysed to establish its quality and sensitivity. To address these challenges, our *p*CTMC synthesis method combines search-based software engineering (SBSE) techniques (Harman et al., 2012a) with techniques for effective *p*CTMCs analysis (Češka et al., 2017; 2016), producing a close approximation of the Pareto-optimal design set.

Algorithm 1 presents the high-level steps of our *p*CTMC synthesis method. The approximate Pareto-optimal design set $\overline{PS}$ returned by this algorithm starts empty (line 2) and is assembled iteratively by the while loop in lines 5–15 until a termination criterion terminate($\mathscr{C}(\mathscr{P}, \mathscr{Q})$, $\overline{PS}$) is satisfied. Each iteration of this while loop uses an SBSE metaheuristic to get a new set of candidate designs (line 4) and then updates the approximate Pareto-optimal design set $\overline{PS}$ in the for loop from lines 5–15. This update involves analysing each candidate design $d = \mathscr{C}(\mathscr{P}', q)$ to establish its associated objective function and constraint values in line 6, where we use the shorthand notation $f_{i,d}^{\top} \equiv f_i^{\top}(\mathscr{C}(\mathscr{P}', q))$, $f_{i,d}^{\perp} \equiv f_i^{\perp}(\mathscr{C}(\mathscr{P}', q))$ and $c_{j,d} \equiv \forall\, p \in \mathscr{P}'.\ c_j(\mathscr{C}(p, q))$ for all $i \in I$, $j \in J$. If the design satisfies all constraints (line 7), the for loop in lines 9–12 finds out if the new design d is dominated by, or dominates, any designs already in $\overline{PS}$. Existing designs dominated by d are removed from $\overline{PS}$ (line 11), and d is added to the Pareto-optimal design set if it is not dominated by any existing designs (line 13).

The elements below must be concretised in the synthesis algorithm, and are described in the next two sections:

1) The analysedesign function for establishing the quality attributes and constraint compliance of a candidate design;
2) The candidatedesigns SBSE metaheuristic and the associated terminate criterion.

The time complexity of Algorithm 1 is *linear* with respect to the overall number of optimisation objectives and constraints and the time required to analyse one quality attribute of a candidate design. The

complexity is further affected by the SBSE metaheuristic setting, namely by the number of generations k (i.e. the number of iterations of the while loop) and the size of the candidate design population $N = |CD|$. Increasing the total number of design evaluations (i.e. $k \cdot N$) typically improves the Pareto optimality of the generated design set, but also slows down the synthesis process. We provide a detailed complexity analysis of the synthesis process in Appendix B.

4.2. Computing safe property bounds for pCTMCs

To establish the quality attributes and sensitivity of candidate designs, ANALYSEDESIGN uses precise parameter synthesis techniques (Češka et al., 2017) to compute safe enclosures of the satisfaction probability of CSL formulae over *p*CTMCs. Given a *p*CTMC $\mathscr{C}(\mathscr{P}', q)$ and a CSL path formula ϕ, these techniques provide a safe under-approximation $\Lambda^q_{\min}$ and a safe over-approximation $\Lambda^q_{\max}$ of the minimal and maximal probability that $\mathscr{C}(\mathscr{P}', q)$ satisfies ϕ:

$$\Lambda^q_{\min} \leq \inf_{p \in \mathscr{P}'} \Lambda_\phi(p, q) \quad \text{and} \quad \Lambda^q_{\max} \geq \sup_{p \in \mathscr{P}'} \Lambda_\phi(p, q).$$

This supports the safe approximation of the bounds $\{f_i{}^\perp, f_i{}^\top\}_{i \in I}$ of the objective functions and of the constraints $\{c_j\}_{j \in J}$. As shown in Češka et al. (2017), the over-approximation quality improves as the size of $\mathscr{P}'$ decreases. Therefore, the precision of the approximation can be effectively controlled via parameter space decomposition, where $\mathscr{P}'$ is decomposed into subspaces $\mathscr{P}'_1, \mathscr{P}'_2 \ldots \mathscr{P}'_n$ and $\Lambda^q_{\min}$ ($\Lambda^q_{\max}$) is taken as the minimum (maximum) of the bounds computed for these n subspaces. Although this refinement step improves the precision of bounds, it also increases the complexity of ANALYSEDESIGN n-fold (Češka et al., 2017).

The satisfaction function Λ_ϕ is typically non-monotonic (and, for nested properties, non-continuous), so safe bounds cannot be obtained by simply evaluating Λ_ϕ at the extrema of parameter region $\mathscr{P}'$. Accordingly, our technique builds on a parametric backward transient analysis that computes safe bounds for the parametric transient probabilities in the discrete-time process derived from the *p*CTMC. This discretisation is obtained through standard uniformisation, and through using the Fox and Glynn algorithm (Kwiatkowska et al., 2007) to derive the required number of discrete steps for a given time bound. Once the parametric discrete-time process is obtained, the computation of the bounds reduces to a local and stepwise minimisation/maximisation of state probabilities in a time non-homogenous Markov process. Presenting the technique in detail as well as the analysis of the approximation error is outside the scope of our paper, but the interested reader can find a complete description in Češka et al. (2017).

Our approach can be easily extended to also support time-unbounded properties by using the method of Quatmann et al. (2016) for parameter synthesis of discrete-time Markov models and properties expressed by time-unbounded formulae of probabilistic computation tree logic.

4.3. Metaheuristic for parametric CTMC synthesis

To ensure that CANDIDATEDESIGNS selects suitable candidate designs, Algorithm 1 is implemented as a *multiobjective optimisation genetic algorithm* (MOGA) such as NSGA-II (Deb et al., 2002) or MOCell (Nebro et al., 2009). MOGAs are genetic algorithms specifically tailored for the synthesis of close Pareto-optimal set approximations that are spread uniformly across the search space. As with any genetic algorithm (Koza, 1992), possible solutions—candidate designs in our case—are encoded as tuples of *genes*, i.e. values for the problem variables. In particular, any candidate design $\mathscr{C}(\mathscr{P}', q)$ that satisfies a fixed set of tolerances $\{\gamma_k\}_{k \in \mathcal{K}}$ is uniquely encoded by the gene tuple (p, q), where $p \in \mathscr{P}$ is the centre point of the continuous parameter region $\mathscr{P}'$. The structure of the gene tuple (p, q) for any *p*CTMC $\mathscr{C}(\mathscr{P}, \mathscr{Q})$ is automatically extracted through parsing the evolvable constructs (4). This feature enables to conveniently encode the *p*CTMC parameters into a representation suitable for the MOGAs.

Example 6 (*Candidate design encoding*). Consider the candidate designs d_1, d_2, d_3 with tolerance value $\gamma = 0.005$ from Example 2. The gene tuple (p, q) of a candidate design $\mathscr{C}(\mathscr{P}', q)$ has the structure (rslowrate, deltarate, moduleidx), where moduleidx $\in \{1, 2\}$ is the index of the Buffer module used by the candidate design. Thus, the designs d_1, d_2, d_3 have gene tuples given by (15.145, 2.08, 2), (13.325, 3.66, 2) and (17.365, 2.93, 1), respectively.

The first execution of CANDIDATEDESIGNS from Algorithm 1 returns a randomly generated *population* (i.e. set) of feasible designs (11). This population is then iteratively evolved by subsequent CANDIDATEDESIGNS executions into populations of "fitter" designs through MOGA *selection, crossover* and *mutation*. Selection chooses the population for the next iteration and a *mating pool* of designs for the current iteration by using the objective functions $\{f_i\}_{i \in I}$, the sensitivity-aware dominance relation (12) and the distance in the parameter space $\mathscr{P}$ between designs to evaluate each design. Crossover randomly selects two designs from the mating pool, and generates a new design by combining their genes, and mutation yields a new design by randomly modifying some of the genes of a design from the pool.

The evolution of the design population terminates (i.e. the predicate $Terminate(\mathscr{C}(\mathscr{P}, \mathscr{Q}), \overline{PS})$ returns true) after a fixed number of design evaluations or when a predetermined number of successive iterations generate populations with no significantly fitter designs.

The implementation of the selection, crossover and mutation operations is specific to each MOGA. For instance, Deb et al. (2002) presents these features for the NSGA-II MOGA used in our experimental evaluation from Section 6.

5. RODES: a robust-design synthesis tool

Our GPU-accelerated RODES tool synthesises sensitivity-aware Pareto sets by implementing the process described in Algorithm 1. In this section, we first present the architecture of RODES, and then describe how we achieved significant performance and scalability improvements through the use of a two-level parallelisation for the synthesis process.

5.1. RODES architecture

As shown in Fig. 5, the operation of RODES is managed by a *Robust-design synthesis engine*. First, a *Model parser* (built using the Antlr parser generator, www.antlr.org) preprocesses the design-space *p*CTMC model. Next, a *Sensitivity-aware synthesiser* uses the jMetal Java framework for multi-objective optimisation with metaheuristics (jmetal.github.io/jMetal) to evolve an initially random population of *candidate designs*, generating a close approximation of the sensitivity-aware Pareto front. This involves using a *Candidate design analyser*, which invokes the probabilistic model checker PRISM-PSY (Češka et al., 2016) to obtain the ranges of values for the relevant quality attributes of candidate designs through precise parameter synthesis. The Pareto front and corresponding Pareto-optimal set of designs are then plotted using MATLAB/Octave scripts, as shown in Fig. 7.

A key feature of RODES is its modular architecture. The Sensitivity-aware synthesiser supports several metaheuristics algorithms, including variants of genetic algorithms and swarm optimisers. Furthermore, the sensitivity-aware Pareto dominance relation can be adapted to match better the needs of the system under development (e.g., by comparing designs based on the worst, best or average quality attribute values). Finally, different solvers could be used for the probabilistic model checker component, including the parameter synthesis solvers for discrete-time Markov chains and time unbounded properties (Quatmann et al., 2016) implemented in the tools PROPhESY (Dehnert et al., 2015) and STORM (Dehnert et al., 2017).

The open-source code of RODES is available on our project website

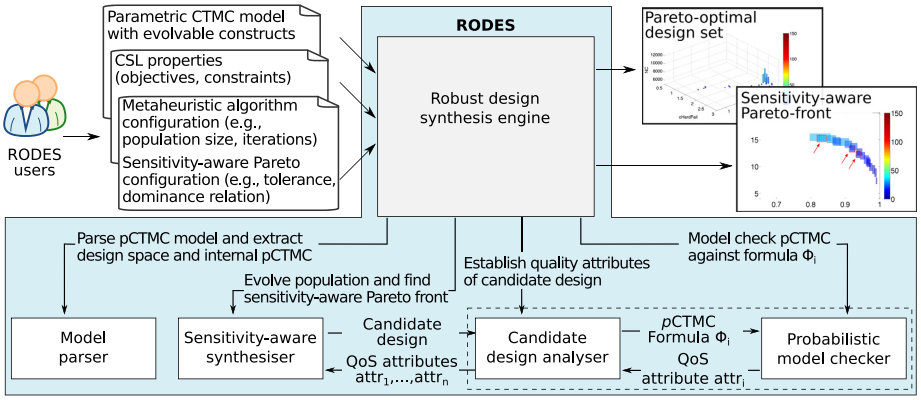


Fig. 5. High-level RODES architecture.

https://www.github.com/gerasimou/RODES.

5.2. Two-level parallelisation

Synthesising sensitivity-aware Pareto sets is a computationally expensive process. To mitigate the performance issues that could arise due to the increased total number of evaluations ($k \cdot N$) or the complexity of evaluating candidate designs (t), we employ a two-level parallelisation.

At the first level, we exploit the fact that the evaluations of particular candidates within a single population are independent and thus they can run in parallel (line 6 in Algorithm 1). A synchronisation is required only after all candidates are evaluated to update the approximate Pareto-optimal set $\overline{PS}$ and to generate new candidates. This granularity of parallelism allows us to efficiently utilise both multi-core and multi-processor architectures. In particular, we can span in parallel a number of tasks that is equal to the population size N and thus significantly alleviate the complexity corresponding to the total number of design evaluations per MOGA generation. We can further increase the parallelisation at this level given that the evaluation of quality attributes for each design is independent. Thus, we can span up to $N{\cdot}(|I| + |J|)$ tasks to evaluate these attributes in parallel and reduce the computation time. The current RODES implementation supports parallelisation at the population level but not at the level of quality attributes, which we plan to add in future tool releases.

The second level of parallelisation aims at accelerating the evaluation of a single candidate over a single quality attribute. The key factor affecting the time t required to analyse a quality attribute of a candidate design is the size of the candidate, namely, the number of non-zero elements M in the rate matrix of the underlying *p*CTMC. This number is proportional to the number of states in the *p*CTMC, and reflects the complexity of the candidate designs. To ensure that RODES supports robust design synthesis for complex systems comprising up to tens thousands of states, our second-level parallelisation improves scalability with respect to the number of states. In particular, we build on our previous work (Češka et al., 2016) to integrate a GPU acceleration of the *p*CTMC analysis into RODES.

This parallelisation is much more involved, since the computation for individual states is not independent. As such, the *p*CTMC analysis is formulated in terms of matrix-vector operations, making it suitable for effective data-parallel processing. Accordingly, RODES implements a state space parallelisation, where a single row of the parametric rate matrix (corresponding to the processing of a single state) is mapped to a single computational element. As the underlying *p*CTMCs typically have a balanced distribution of the state successors, this mapping yields a balanced distribution of non-zero elements in the rows of the matrix. The outcome is a good load balancing within the computation elements, leading to significant acceleration. In contrast to the parallelisation proposed in Češka et al. (2016), RODES is designed to leverage the computational power of modern GPUs, which provide hundreds of computational elements and can schedule thousands of active threads in a different way. In particular, RODES can evaluate on a single GPU several candidate designs (that can differ both in their discrete and in their continuous parameters) in parallel, provided that the underlying *p*CTMCs can fit in the GPU memory. This enables an efficient and flexible utilisation of the available computation power for complex robust design synthesis problems (see performance evaluation results in Section 6.4).

6. Evaluation

We evaluate the effectiveness of RODES using three systems from different application domains. Also, we assess the performance and scalability of RODES including the impact of the two-level parallelisation. We conclude our evaluation with a discussion of threats to validity.

6.1. Research questions

The aim of our experimental evaluation was to answer the following research questions.

RQ1 (Decision support): Can RODES support decision making by identifying effective tradeoffs between the QoS optimality and the sensitivity of alternative designs? To support decision making, RODES must provide useful insights into the robustness of alternative system designs. Therefore, we assessed the optimality-sensitivity tradeoffs suggested by RODES for the software systems used in our evaluation.

RQ2 (Performance): Does the two-level paralellisation improve the efficiency of RODES? Since the synthesis of robust models is a computationally expensive process, we examined the change in performance thanks to the two-level parallelisation architecture described in Section 5.2.

RQ3 (Metaheuristic effectiveness): How does our RODES approach perform compared to random search? Following the standard practice in search-based software engineering Harman et al. (2012b), we assessed if the stochastic models synthesised

by RODES "comfortably outperform" those synthesised by a random search approach.

6.2. Analysed software systems

We performed a wide range of experiments to evaluate our RODES approach and tool using three software systems from different application domains:

- a producer-consumer (PC) software system described in Examples 1–5;
- a replicated file system used by Google's search engine Baier et al. (2013);
- a cluster availability management system Haverkort et al. (2000).

We have already presented the PC system in Examples 1–5. In the following paragraphs, we introduce the other systems, provide a description of their stochastic models and present the objectives and constraints used to synthesise robust Pareto optimal designs. Further information about these systems are available on our project website at https://www.github.com/gerasimou/RODES/wiki.

Google file system (GFS). GFS partitions files into chunks of equal size, and stores copies of each chunk on multiple *chunk servers*. A master server monitors the locations of these copies and the chunk servers, replicating the chunks as needed. During normal operation, GFS stores CMAX copies of each chunk. However, as servers fail and are repaired, the number c of copies for a chunk may vary from 0 to CMAX.

Previous work modelled GFS as a CTMC with fixed parameters and focused on the analysis of its ability to recover from disturbances (e.g. $c <$ CMAX) or disasters (e.g. master server down) (Baier et al., 2013). In our work, we adapt the CTMC of the lifecycle of a GFS chunk from Baier et al. (2013) by considering several continuous and discrete parameters that a designer of the system has to decide. Fig. 6 shows the resulting model, encoded in the PRISM modelling language extended with the evolve constructs from (4). As in Baier et al. (2013), we model separately the software and hardware failures and repairs, for both the master server (lines 22–25) and the chunk servers (lines 26–31), and assume that loss of chunk copies due to chunk server failures leads to further chunk replications, which is an order of magnitude slower if $c = 0$ and a backup of the chunk must be used (line 32).

To evaluate RODES, we assume that GFS designers must select the hardware failure and repair rates cHardFail and cHardRepair of the chunk servers, and the maximum number of chunks NC stored on a chunk server within the ranges indicated in Fig. 6. These parameters reflect the fact that designers can choose from a range of physical servers, can select different levels of service offered by a hardware repair workshop, and can decide a maximum workload for chunk servers. We consider an initial system state modelling a severe hardware disaster with all servers down due to hardware failures and all chunk copies lost, and we formulate a *p*CTMC synthesis problem for quality requirements given by two maximising objective functions and one constraint:

f_1: $P_{=?}$ [$\neg$ SL1 $U^{[10,60]}$ SL1], where SL1 = Mup $\wedge$ c > 0 holds in states where *service level 1* (master up and at least one chunk copy available) is provided;
f_2: $R\{\text{``}active\text{''}\}_{=?}[C^{<=60}]$, where a reward of 1 is assigned to the states with a number of running chunk servers of at least 0.5M (i.e., half of the total number of chunk servers);
c_1: $R\{\text{``}replicates\text{''}\}_{\leq 5}[C^{<=60}]$, where a transition reward of 1 is assigned to each chunk replication transition.

Objective f_1 maximises the probability that the system recovers service level 1 in the time interval [10,60] hours. Objective f_2 maximises the expected time the system stays in (optimal) states with at least $0.5M$ chunk servers up in the first 60 hours of operation. Finally, constraint c_1 restricts the number of expected chunk replications over 60 h of operations.

```
1  ctmc
   // Failure rates
2  const double mSoftFail = 0.000475;     // master software
3  const double mHardFail = 0.000025;     // master hardware
6  const double cSoftFail = 0.475;        // chunk server software
7  evolve double cHardFail [0.25..4.0];   // chunk server hardware
   // Repair rates
4  const double mSoftRepair = 12;         // master software
5  const double mHardRepair = 6;          // master hardware
8  const double cSoftRepair = 12;         // chunk server software
9  evolve double cHardRepair [0.5..4.0];  // chunk server hardware
10 const int N=100000;                    // total number of GFS chunks
11 const int M=20;                        // number of chunk servers
12 evolve int NC [5000..20000];           // max chunks per chunk server
13 const int CMAX=3;                      // optimal number of chunk copies

14 module GFS_Chunk
15  M_up : bool init false;      // master is up
16  M_sdown : bool init false;   // master is down with SW problem
17  M_hdown : bool init true;    // master is down with HW problem
18  Cup : [0..M] init 0;         // number of chunk servers up
19  Csdown : [0..M] init 0;      // number of chunk servers down (SW problem)
20  Chdown : [0..M] init 20;     // number of chunk servers down (HW problem)
21  c : [0..CMAX] init 0;        // number of chunk copies available
   // Master server failure and repair
22  [] M_up → mSoftFail : (M_up'=false)&(M_sdown'=true);
23  [] M_up → mHardFail : (M_up'=false)&(M_hdown'=true);
24  [] M_sdown → mSoftRepair : (M_up'=true)&(M_sdown'=false);
25  [] M_hdown → mHardRepair : (M_up'=true)&(M_hdown'=false);
   // Chunk servers failure and repair
26  [] Cup>0 & c>0 & Csdown<M → (c/Cup)*cSoftFail :
                    (Cup'=Cup-1)&(Csdown'=Csdown+1)&(c'=c-1);
27  [] Cup>0 & Cup>c & Csdown<M → (1-(c/Cup))*cSoftFail :
                    (Cup'=Cup-1)&(Csdown'=Csdown+1);
28  [] Cup>0 & c>0 & Chdown<M → (c/Cup)*cHardFail :
                    (Cup'=Cup-1)&(Chdown'=Chdown+1)&(c'=c-1);
29  [] Cup>0 & Cup>c & Chdown<M → (1-(c/Cup))*cHardFail :
                    (Cup'=Cup-1)&(Chdown'=Chdown+1);
30  [] Cup<M & Csdown>0 → Csdown*cSoftRepair :
                    (Csdown'=Csdown-1)&(Cup'=Cup+1);
31  [] Cup<M & Chdown>0 → cHardRepair :
                    (Chdown'=Chdown-1)&(Cup'=Cup+1);
32  [] M_up & c<CMAX & Cup>c & Cup*NC>=(c+1)*N →
                    ((c>0)?20:2):(c'=c+1);
33 endmodule
```

Fig. 6. *p*CTMC model of the Google file system.

Workstation cluster (WC). We extend the CTMC of a cluster availability management system from Haverkort et al. (2000). This CTMC models a system comprising two sub-clusters, each with N workstations and a switch that connects the workstations to a central backbone. For each component, we consider failure, inspection and repair rates (where repairs are initiated only after an inspection detects failures), and we assume that designers must decide these rates for workstations—i.e., the real-valued parameters wsFail, wsCheck and wsRepair for our *p*CTMC, respectively. Additionally, we assume that designers must select the sub-cluster size N, and must choose between an expensive repair implementation (i.e., *p*CTMC module) with a 100% success probability and a cheaper repair module with 50% success probability—i.e., two discrete parameters for the *p*CTMC. We made this model available on our repository of case studies.

For an initial system state with 5 workstations active in each sub-cluster and switches and backbone working, we formulate a *p*CTMC synthesis problem for quality requirements given by two maximising objective functions and one constraint:

f_1: $P_{=?}$ [$\neg$ premium U [20, 100] premium], where premium denotes a system service where at least $1.25N$ workstations are connected and operating;
f_2: $R\{\text{``}operational\text{''}\}_{=?}[C^{\leq 100}]$, where a reward of 1 is assigned to

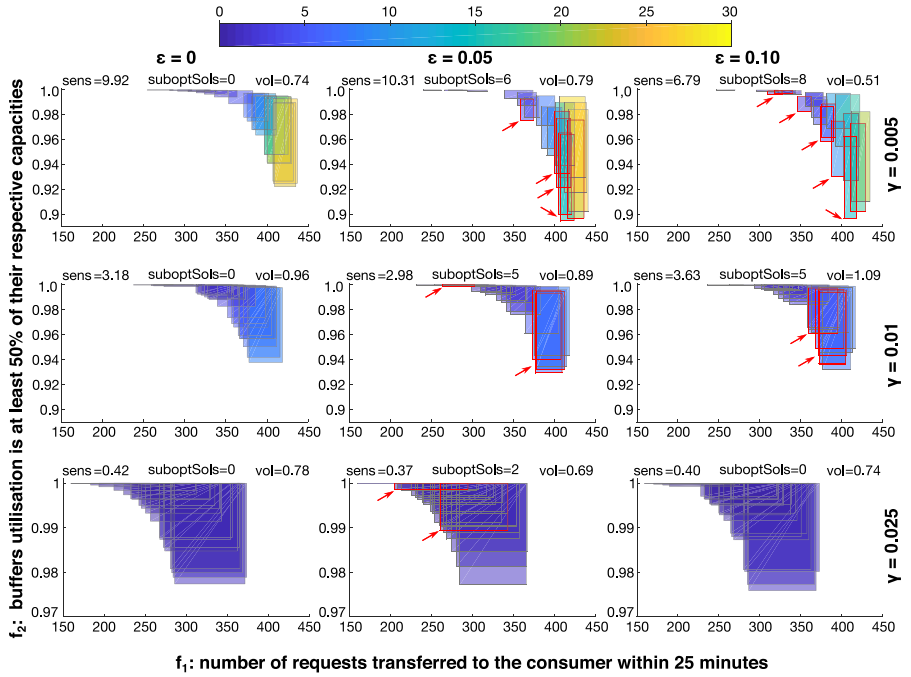


Fig. 7. Sensitivity-aware Pareto fronts for the producer-consumer model. Boxes represent quality-attribute regions, coloured by sensitivity (yellow: sensitive, blue: robust). Red-bordered boxes indicate sub-optimal robust designs. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box). Statistics are: sens, average sensitivity of the front; suboptSols, number of suboptimal solutions; vol, average volume of the front. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

states with a number of operating clusters between $1.2N$ and $1.6N$;
c_1: $R\{\text{“}repair\text{”}\}_{\leq 80}[C^{\leq 100}]$, where transition rewards are associated with repair actions of the workstations, backbone and switches.

Objective f_1 maximises the probability that the system recovers the premium service in the time interval [20,100] hours. Objective f_2 maximises the expected time the system spends in cost-optimal states during the first 100 hours of operation. Constraint c_1 restricts the cost of repair actions during this time (the definition of the cost is provided on our project website).

6.3. Evaluation methodology

We used the following configuration to evaluate RODES: NSGA-II MOGA, 10,000 evaluations, initial population of 20 individuals, and default values for single-point crossover probability $p_c = 0.9$ and single-point mutation probability $p_m = 1/(|K| + |D|)$, with $|K| + |D|$ the number of (continuous and discrete) design-space parameters. We examine the behaviour of the sensitivity-aware Pareto dominance relation using different combinations of tolerance values $\gamma \in \{0.005, 0.01, 0.025\}$ and sensitivity-awareness coefficients $\epsilon_i \in \{0.00, 0.05, 0.10\}$.

For each experiment, we report the sensitivity-aware Pareto fronts (Figs. 7, 9, 12 and 14). The Pareto-optimal designs are depicted as boxes in the quality-attribute space and coloured by sensitivity, using the same representation as in Figs. 1 and 4. We also show the synthesised designs in the design space, given by the continuous and discrete parameters of the system. In this case, designs are represented as boxes in the continuous parameter space, representing the extent of the parameter variation under the given tolerance. The third dimension (vertical axis) in Figs. 10 and 13 gives the value of the discrete parameter.

6.4. Results and discussion

RQ1 (Decision support). We analysed the designs synthesised by RODES in order to identify actionable insights regarding the tradeoffs between the QoS attributes and sensitivity of alternative architecture designs. For each system, we present our findings independently.

Producer-consumer system (PC). First, we present the results for the producer consumer system introduced in Examples 1–5, obtained by running our RODES tool with tolerances $\gamma \in \{0.005, 0.01, 0.025\}$ for both continuous parameters (r_slow_rate and delta_rate). The resulting Pareto fronts are shown in Fig. 7, for objectives f_1 (number of requests transferred to the consumer within 25 minutes) and f_2 (probability of adequate buffer utilization) and sensitivity-awareness parameters $\epsilon_1 = \epsilon_2 = \epsilon \in \{0, 0.05, 0.1\}$. The corresponding synthesised designs are presented in Fig. 8.

These Pareto fronts provide a wealth of information supporting the evaluation of the optimality and robustness of alternative designs. In particular, the Pareto front for $\epsilon = 0$ and $\gamma = 0.005$ contains several large (yellow) boxes that correspond to highly sensitive designs.

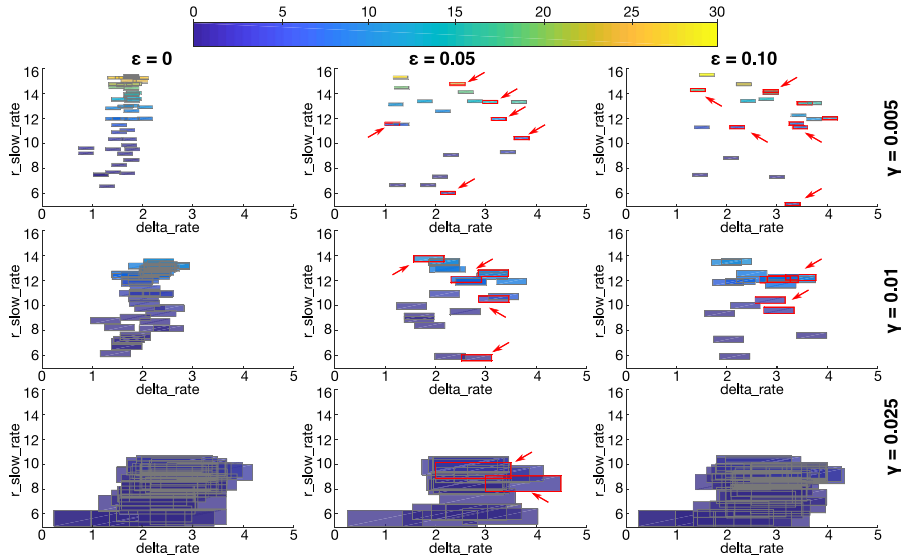


Fig. 8. Synthesised Pareto-optimal designs for the producer-consumer model and experiments from Fig. 7. Rectangles in x-y plane correspond to the continuous parameter regions. The discrete parameter (module - 'mod') is omitted since RODES synthesised solutions using only the redirection module ('mod2'). Boxes are coloured by sensitivity.

Increasing ϵ produces a number of robust sub-optimal designs (red-bordered) with slightly sub-optimal quality attributes but improved robustness. Such designs represent valuable alternatives to the highly sensitive solutions obtained using the classical, sensitivity-agnostic, dominance relation. This ability to identify poor (i.e. highly sensitive) designs and then alternative robust designs with similar quality attributes is a key and unique benefit of our design synthesis method. Consider for instance the results for $\epsilon = 0.05$ and $\gamma = 0.005$. There are several sensitive designs at high f_1 values (see Fig. 7), which correspond to designs with rslowrate above 15 and low values deltarate (below 2.5), see Fig. 8. Through our method, we found that there exist alternative sub-optimal designs with improved robustness (highlighted green boxes), corresponding to higher deltarate and lower rslowrate values, i.e, to designs with a slower slow buffer and a faster fast buffer.

Furthermore, we observe that the overall sensitivity improves as the tolerance γ increases, meaning that the uncertainty (volume) of the quality attribute regions grows proportionally smaller than the uncertainty of the corresponding parameter regions, see (10). This explains why we observe fewer sub-optimal robust designs for higher tolerances ($\gamma = 0.01, 0.025$). Increasing parameter tolerances also affects the quality attribute profiles as it leads to larger ranges for objective f_1 (i.e., more sensitive) and to smaller ranges for f_2 (i.e., more robust). As a consequence, RODES tends to favour Pareto-optimal solutions with better f_2 and worse f_1 values as the tolerance increases. In particular, for $\gamma = 0.025$ all designs with $f_1^{\perp} \geq 300$ are excluded (corresponding to the most sensitive designs for $\gamma = 0.005, 0.01$), which yields regions with average volume comparable to those for $\gamma = 0.025$.

The synthesised parameter regions (Fig. 8) indicate that redirection (second module – 'mod2') is always preferred to non-redirection. Also, the generated designs select values for the continuous parameters from the lower-end of their respective range, with rslowrate $\in$ [5.00, 15.650] and deltarate $\in$ [0.242, 4.489]. In other words, our algorithm found Pareto-optimal designs where both buffers have slow transmission rates (with the fast buffer being slightly faster), while solutions where the fast buffer has a sensibly higher transmission rate, but a proportional packet loss rate, are excluded. In particular, configurations with slow transmission rates have associated good robustness, with very little ranges for objective f_2.

We also observe an interesting relationship between the Pareto-optimal fronts and the Pareto-optimal designs for different values of the sensitivity-awareness parameter $\epsilon \in \{0, 0.05, 0.1\}$. The average values for both objectives f_1 and f_2 experience only little variation as ϵ increases for a fixed tolerance value. For instance, when $\gamma = 0.01$, on average $f_1 \in [369.37, 372.40]$ and $f_2 \in [0.974, 0.98]$, and when $\gamma = 0.05$, $f_1 \in [284.94, 285.48]$ and $f_2 \in [0.995, 0.996]$. Conversely, the average values for the continuous parameters rslowrate and deltarate experience more significant variation and present an interesting negative relationship. More specifically, for any γ value and as the ϵ parameter becomes larger, rslowrate shows a decreasing trend while deltarate shows an increasing trend. We used the Pearson correlation test to analyse this observation and received a strong negative correlation with the coefficient $R \in [-0.992, -0.988]$[4]. This result indicates that as ϵ increases, the sensitivity-aware Pareto-optimal set includes designs in which the transmission rate difference between the slow and fast buffers grows. Although unexpected, this observation is very useful.

Producer-consumer variant. We further analyze a variant of the producer-consumer model, illustrated in Fig. 11. In this version, we assume a different redirection strategy (lines 10 and 11) that yields a 100% probability of redirection when the slow buffer is full, while in the original variant the maximum redirection probability is limited to 0.1. We also consider different continuous parameters: the request

[4] This result should not be confused with the correlation between the continuous parameters rslowrate and deltarate for fixed γ and ϵ values which ranges from zero to weak, i.e., $R \in [0, 0.3]$.

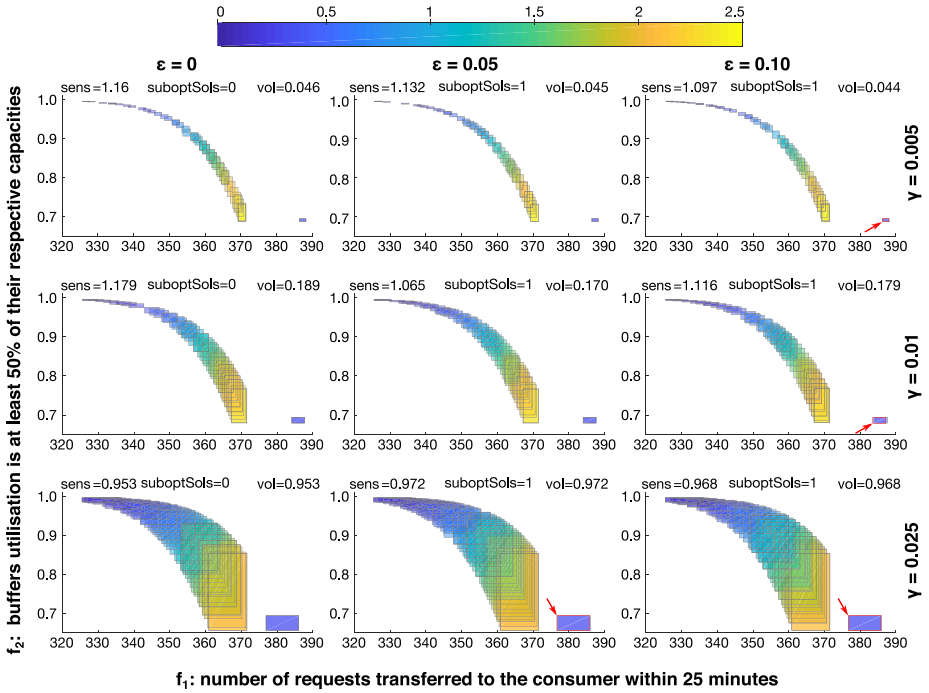


Fig. 9. Sensitivity-aware Pareto fronts for the second variant of the producer-consumer model. Legend and colour code are as in Fig. 7. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box).

production rate (p_rate) and the packet transmission rate for the fast buffer (r_fast_rate). The synthesized Pareto fronts and designs are reported in Figs. 9 and 10, respectively.

We observe that the obtained Pareto-optimal set is substantially different from the one obtained in the first variant of the model (Fig. 7). Solutions in this variant are generally more robust, demonstrated by the fact that at most one suboptimal solution is synthesised for each configuration. A common trait is that favouring objective f_2 leads to robust designs, while robustness is penalized for high f_1 values. Comparing the two PC variants, whose *p*CTMC models are shown Figs. 3 and 11, we observe that most of the solutions of the second variant are dominated by the Pareto front of the first variant for $\gamma \in \{0.005, 0.01\}$ and all ϵ values, which therefore provides the best performance.

The synthesized parameter regions (Fig. 10) confirm the results of the first variant: redirection is always preferred (for all but one design), and the fast buffer rate is not too far from that of the slow buffer (rfastrate = 13.03). Similarly, all synthesized values for parameter p_rate are very close to the fixed value (40) used for the same parameter in the first variant of the model. In the Pareto front, we can observe an outlier yielding the highest system throughput (f_1). This design is obtained when redirection is disabled (see Fig. 10). Notably, no other designs with no redirection are present in the Pareto front which provides evidence that redirection is essential to achieve a well-balanced utilisation of the buffers.

Google file system (GFS). Given the *p*CTMC model, the two maximisation objectives and one constraint of the GFS system, we used RODES to generate Pareto-optimal design sets with tolerances $\gamma \in \{0.005, 0.01, 0.025\}$ for both continuous parameters (cHardFail and cHardRepair) of our *p*CTMC. Fig. 12 shows the Pareto fronts obtained using the "lower bound" definition from Table 1 for the objective functions f_1 and f_2 over candidate designs, and parameters $\epsilon_1 = \epsilon_2 = \epsilon \in \{0, 0.05, 0.1\}$ for the sensitivity-aware Pareto dominance relation (12). The design-space representation is given in Fig. 13. We observe that the Pareto front for $\epsilon = 0$ and $\gamma = 0.005$ contains several large (yellow) boxes that correspond to highly sensitive designs. For $\epsilon \in \{0.05, 0.1\}$ and $\gamma = 0.005$, these poor designs are "replaced" by robust designs – surrounded by red borders – with very similar quality attributes but slightly sub-optimal. The same pattern occurs for $\gamma = 0.01$ and (to a lesser extent because of the overall lower sensitivity) for $\gamma = 0.025$. For instance, consider the sensitive design obtained for $\epsilon = 0.1$ and $\gamma = 0.005$ characterized by low hardware fail and repair rates and high number of chunks (yellow bar on Fig. 13). Our method found that a more robust solution is possible (highlighted green region), with lower NC and higher cHardFail and cHardRepair.

We also observe that favouring objective f_1 over f_2 generally yields more robust designs (i.e., smaller quality-attribute regions towards the right end of the Pareto fronts) for all combinations of ϵ and γ.

The design-space view of Fig. 13 evidences a trade-off between cHardFail and cHardRepair, i.e., optimal designs tend to have either high failure rates and high repair rates, or low failure and repair rates. Results for $\gamma = 0.025$ reveal that there is actually an ideal ratio between the two parameters as the corresponding optimal design appear to keep a relatively constant proportion between cHardFail and cHardRepair. This result was unexpected, yet very useful, since it indicates that designs not satisfying this trade-off yield excessively fast or slow recovery

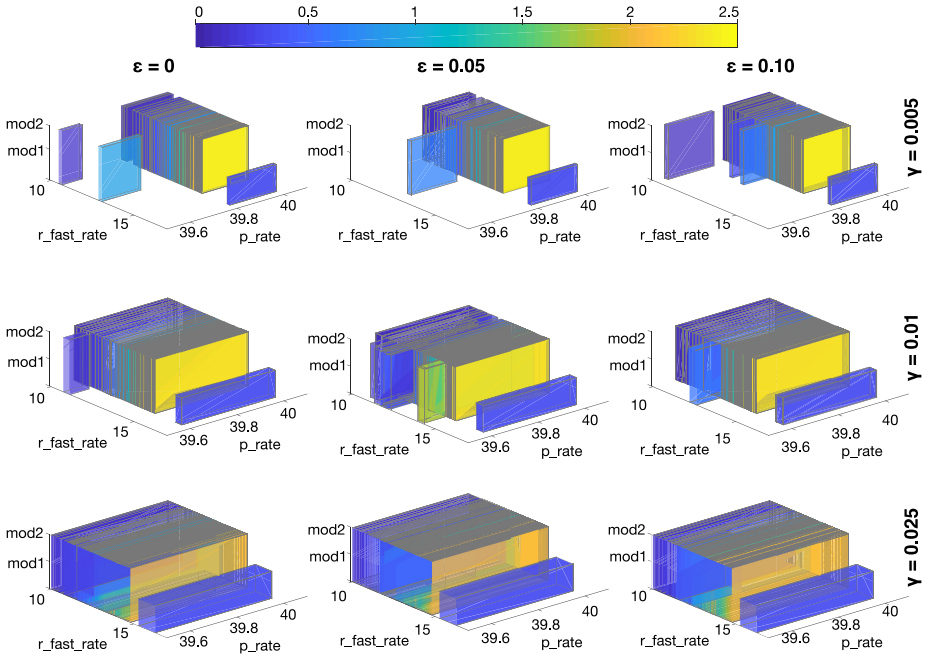


Fig. 10. Synthesised Pareto-optimal designs for the second variant of the producer-consumer model and experiments from Fig. 9. Rectangles in x-y plane correspond to the continuous parameter regions Boxes are coloured by sensitivity.

```
1  ctmc
   // buffer capacities
2  const int slow_max = 31;
3  const int fast_max = 21;
4  evolve double p_rate [20..40];      //request production rate
5  const double s_rate = 30;           //request transmission rate
   // packet transmission rates
6  const double r_slow_rate = 10;      //slow buffer
7  evolve double r_fast_rate [10..30]; //fast buffer
8  const double c_rate = 40;           //packet consumption rate
⋮
// redirection
9  evolve module Buffer
     ⋮
     //send
10   [] (slow=1) & (buffer_s<slow_max) → s_rate *(1-buffer_s/slow_max) :
                                          (slow'=0) & (buffer_s' = buffer_s +1);
11   [] (slow=1) & (buffer_s>0) & (buffer_f<fast_max) → s_rate *buffer_s/slow_max :
                                          (slow'=0) & (buffer_f' = buffer_f +1);
12   [] (fast=1) & (buffer_f<fast_max) → s_rate :
                                          (fast'=0) & (buffer_f' = buffer_f +1);
     ⋮
13 endmodule
```

Fig. 11. Variant of the producer-consumer model introduced in Section 2.

times, and thus are far from the optimal f_1 values.

Further, we observe that the maximum number of chunks per server, NC, has a major influence on the design robustness, with high NC values leading to highly sensitive designs. These designs should be avoided in favour of the alternative designs with low NC values depicted in Fig. 13 (for $\epsilon > 0$).

Workstation cluster (WC). Fig. 14 depicts the Pareto fronts obtained for all γ, ϵ combinations of the WC *p*CTMC model. These Pareto fronts show again how the large quality-attribute regions (corresponding to high-sensitivity designs) obtained for $\epsilon = 0$ are "replaced" by much smaller quality-attribute regions on the Pareto fronts obtained for both $\epsilon > 0$ values. For instance, the fronts produced for $\gamma = 0.005$ and $\epsilon \in \{0.05, 0.10\}$, include sub-optimal robust designs in the objective space $[0.6, 0.8] \times [40, 50]$ that do not exist for $\epsilon = 0$. Further, the Pareto front for $\gamma = 0.005$, $\epsilon = 0.10$ includes a sub-optimal robust design in the objective space $[0.3, 0.5] \times [45, 70]$ to support the Pareto-optimal but volatile (i.e., highly sensitive) designs within that space. Similar observations can be made for other γ values.

With respect to the system dynamics, our sensitivity-aware synthesis method reveals that the most robust solutions correspond to the objective-function "extrema" from the Pareto front, i.e., to quality-attribute regions in which either f_1 is very high and f_2 is very low, or vice versa. In particular, solutions in the middle of quality-attribute regions are highly sensitive as indicated by the yellow-green boxes for $\gamma = 0.005$ and $\epsilon \in \{0.00, 0.05, 0.10\}$. The equivalent solutions are absent from the Pareto fronts for $\gamma = 0.01$ indicating that they are replaced by more robust solutions whose quality attributes are close to the low- and high-end of their respective ranges. Thus, if designers seek robust solutions they need to select designs that favour one of the quality attributes, since solutions with balanced trade-off between the quality attributes lead to either sensitive or sub-optimal robust designs.

We also identified an interesting property of the synthesized designs. Although they cover the entire design space for the real-valued parameters wsFail, wsCheck and wsRepair, the synthesized designs select very few values for the sub-cluster size *N*. In particular, in more

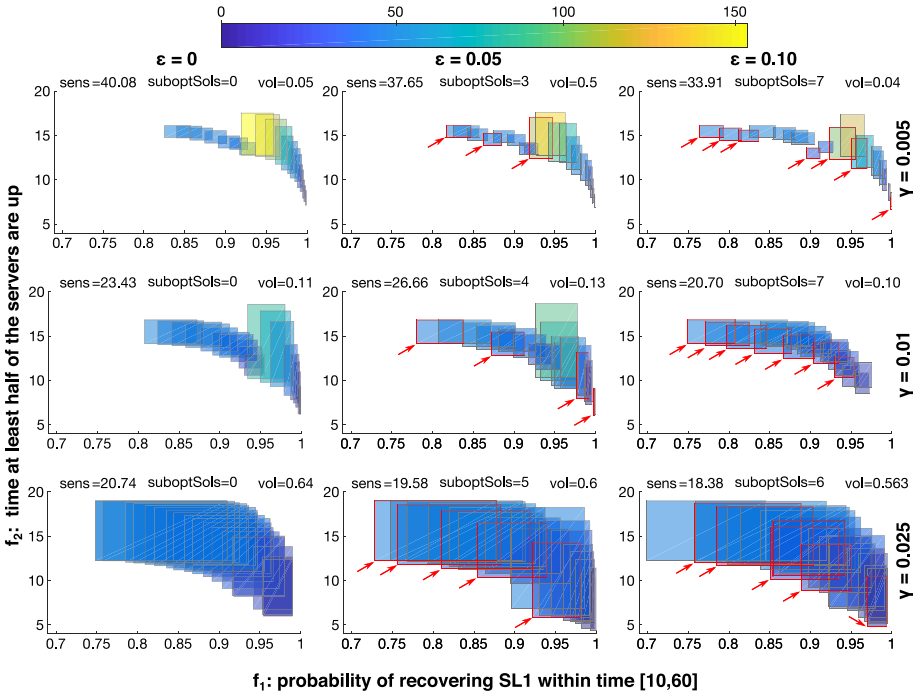


Fig. 12. Sensitivity-aware Pareto fronts for the GFS model. Legend and colour code are as in Fig. 7. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box).

than 95% of the experiments $N \in \{10, 15\}$ and in the remaining $N \in \{9, 12\}$. We analysed further this observation and ran another experiment by setting the possible range for sub-cluster size $N \in \{11, .., 14\}$. Table 2 compares the average sensitivity between these two experiments for all γ, ϵ combinations. Our results validate that the 'ideal' values of the parameter N for the synthesised robust designs are 10 or 15. This finding demonstrates an unexpected and interesting relationship between the size of the cluster and robustness, impossible to derive through existing analysis methods.

RQ2 (Performance). Since the synthesis process is computationally demanding (see Appendix B), we evaluated the performance of RODES to analyse multiple candidate designs in parallel using the two-level parallelisation architecture described in Section 5.2. By employing the two-level parallelisation, we are able to partially alleviate the CPU overheads incurred not only due to the complexity of evaluating a candidate design but also due to the high number of evaluations. All experiments were run on a CentOS Linux 6.5 64bit server with two 2.6GHz Intel Xeon E5-2670 processors and 64GB memory. For the experiments involving GPU parallelisation, we used two nodes using either an nVidia K40 GPGPU card or an nVidia K80 GPGPU card.

The key results of our performance evaluation are described in Tables 3 and 4. The tables show the design synthesis run-times for $k = 500$ and $N = 20$ (i.e. for $kN = 10,000$ design evaluations), for our three case studies. Run-time statistics are computed over more than 30 independent runs, obtained using all combinations of $\epsilon \in \{0, 0.05, 0.1\}$ and $\gamma \in \{0.005, 0.01, 0.025\}$. Note that 10,000 evaluations, for which we obtained high quality sensitivity-aware Pareto fronts, are still negligible with respect to the size of the design space that an exhaustive search would need to explore (theoretically the design space is uncountable). To demonstrate this difference, we list the number of candidate designs required to "cover" the design space for a given tolerance value γ (this number is indeed much smaller than the total number of candidate designs). For PC model ($\gamma = 0.005$) it is around 20,000 designs, but for WC and GFS ($\gamma = 0.01$) it is more than 3 millions designs.

Results in Table 3 confirm that performance of the synthesis process is affected mainly by the size of the underlying pCTMC and by the average number of the discretisation steps required to evaluate particular quantitative attributes (around 4000 steps are required for WC and PC, 160,000 for GFS v1, and 46,000 for GFS v2). Note that this number depends on the highest time bound appearing in the properties and on the highest rate appearing in the transition matrix. This observation also explains the significant slowdown of the synthesis process when switching from v1 to v2 of GFS.

First, we evaluate the performance of CPU-only paralellisation at different numbers of cores. The results clearly confirm the scalability with respect to the number of cores. We can also observe that a better scalability is obtained for more complicated synthesis problems (i.e. 5.5-times speed for 10 cores on GFS v1 versus 7.9-times speed up for 10 core on GFS v2).

Second, we evaluate the performance of the two-level parallelisation. Table 4 compares the run-times for different number of CPU cores and GPU devices. In this configuration, we obtain a significant reduction of runtimes, e.g. for GFS v2 we obtain 8.4-times speedup with one GPU and one CPU core, and 7.6-times speedup with two GPUs and two

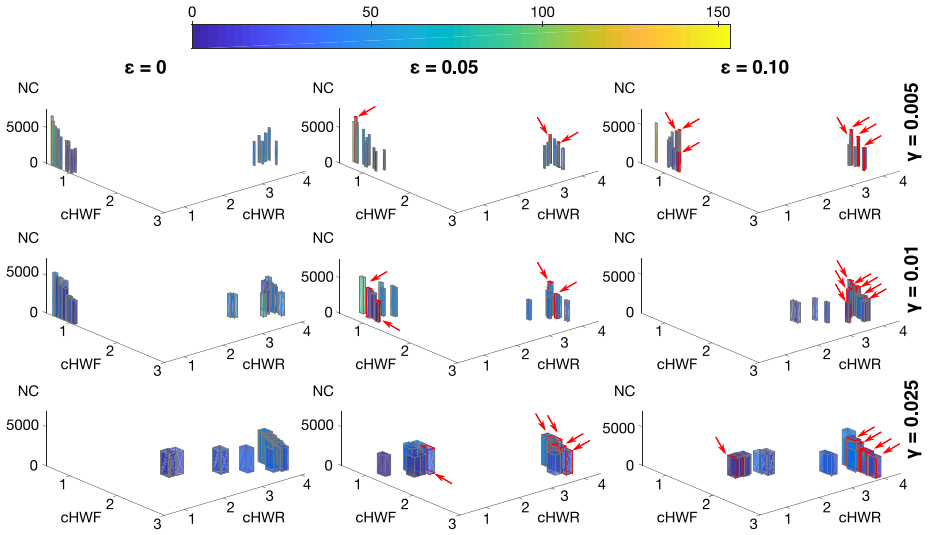


Fig. 13. Synthesised Pareto-optimal designs for the GFS model and experiments from Fig. 12. Rectangles in x-y plane correspond to the continuous parameter regions.

CPU cores. The slightly worse speedup observed in the latter case is due to the increased CPU-GPU communication overhead when more devices are employed.

Finally, we see that evaluating more that one candidate solutions (generated using several CPU cores) on a single GPU further improves the performance until the GPU is fully utilised (i.e. the maximal number of active threads that can be dispatched is reached and thus some parallel evaluations has to be serialised). The performance is also affected by the memory access pattern that depends on the concrete candidate solutions evaluated in parallel. In particular, the performance degrades when the memory access locality is decreased. Note that the maximal number of candidate solutions that can be evaluated in parallel on a single GPU is also limited by the GPU memory that has to accommodate the underlying *p*CTMC.

RQ3 (Metaheuristic effectiveness). To answer this research question, we analysed the goodness of the Pareto-optimal designs of the GFS model obtained with our NSGA-II-based RODES against a variant that uses random search (RS). For each variant and combination of $\epsilon \in \{0, 0.05, 0.10\}$ and $\gamma \in \{0.005, 0.01\}$ we carried out 30 independent runs, in line with standard SBSE practice (Harman et al., 2012b). As building the actual Pareto front for large design spaces is challenging and computationally expensive (GFS has $|\mathscr{P} \times \mathscr{A}| > 24E10$ assuming a three-decimal precision for continuous parameters), we again followed the standard practice and combined the sensitivity-aware Pareto fronts from all 60 RODES and RS runs for each ϵ, γ combination into a *reference Pareto front* (Zitzler et al., 2003). We then compared the Pareto fronts achieved by each variant against this reference front by using the metrics

$$M_1 = wI_{\epsilon_{norm}} + (1-w)\overline{sens}_{norm}$$

and

$$M_2 = wI_{IGD_{norm}} + (1-w)\overline{sens}_{norm}$$

which use a weight $w \in [0, 1]$ to combine normalised versions of the established (but sensitivity-agnostic) Pareto-front quality metrics I_ϵ and I_{IGD} (Zitzler et al., 2003) with the normalised design sensitivity. The unary additive epsilon (I_ϵ) gives the minimum additive term by which the objectives of a particular design from a Pareto front must be altered to dominate the respective objectives from the reference front. The inverted generational distance (I_{IGD}) measures the shortest Euclidean distance from each design in the Pareto front to the closest design in the reference front. These indicators show convergence and diversity to the reference front (smaller is better).

Fig. 15 compares RODES and RS across our ϵ, γ combinations using metrics M_1 and M_2 with $w = 0.5$. The RODES median is consistently lower than that of RS for all ϵ, γ combinations with the exception of $\epsilon = 0$, $\gamma = 0.01$ (which ignores design sensitivity) for M_2. For a given γ, RODES results improve as ϵ increases, unlike the corresponding RS results. Thus, the difference between RODES and RS increases with larger ϵ for both metrics. This shows that RODES drives the search using sensitivity (10), and thus it can identify more robust designs. We confirmed these visual inspection findings using the non-parametric Mann–Whitney test with 95% confidence level ($\alpha = 0.05$). We obtained statistical significance (p-value < 0.05) for all ϵ, γ combinations except for $\epsilon = 0$, $\gamma = 0.005$, with p-value in the range [1.71E-06, 0.0026] and [1.086E-10, 0.00061] for M_1 and M_2, respectively.

Considering these results, we have sufficient empirical evidence that RODES synthesises significantly more robust designs than RS. These results are also in line with our previous work which demonstrated through extensive evaluation that probabilistic model synthesis using MOGAs achieves significantly better results that RS (Gerasimou et al., 2015). Hence, the problem of synthesising sensitivity-aware Pareto optimal sets (13) is challenging, as expected for any well-defined SBSE problem.

6.5. Threats to validity

Construct validity threats may arise due to assumptions made when modelling the three systems. To mitigate these threats, we used models and quality requirements based on established case studies from the literature (Ghemawat et al., 2003; Haverkort et al., 2000).

Internal validity threats may correspond to bias in establishing cause-

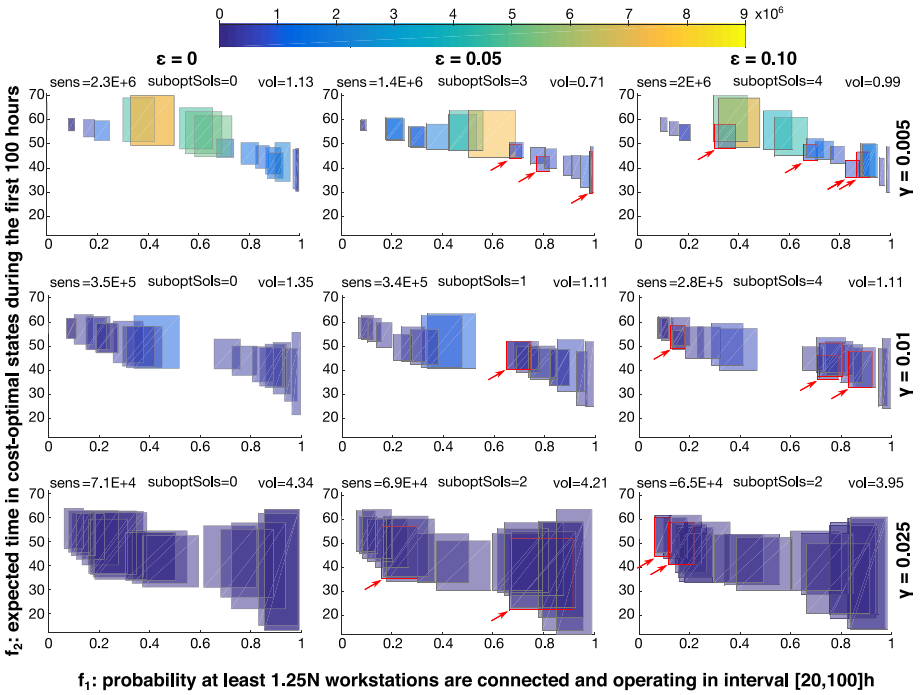


Fig. 14. Sensitivity-aware Pareto fronts for the workstation cluster model. Legend and colour code are as in Fig. 7. Designs are compared based on the worst-case quality attribute value (i.e. lower-left corner of each box).

Table 2
Average design sensitivity for two variants of the workstation cluster synthesis problem, given by different ranges for parameter N. Sensitivity-aware designs (i.e. where $\epsilon > 0$) for $N \in \{10..15\}$ have lower sensitivity than for $N \in \{11..14\}$.

	Average sensitivity								
N	$\gamma = 0.005$, $\epsilon = 0.00$	$\gamma = 0.005$, $\epsilon = 0.05$	$\gamma = 0.005$, $\epsilon = 0.10$	$\gamma = 0.01$, $\epsilon = 0.00$	$\gamma = 0.01$, $\epsilon = 0.05$	$\gamma = 0.01$, $\epsilon = 0.10$	$\gamma = 0.025$, $\epsilon = 0.00$	$\gamma = 0.025$, $\epsilon = 0.05$	$s\gamma = 0.025$, $\epsilon = 0.10$
{10.15}	1.6E6	7.86E5	6.58E5	2.1E5	2.49E5	2.19E5	6.45E4	6.68E4	7.56E4
{11.14}	1.33E6	1.3E6	1.22E6	5.2E5	5.28E5	4.77E5	2E5	1.93E5	1.87E5

effect relationships in our experiments. We limit them by examining instantiations of the sensitivity-aware Pareto dominance relation (12) for multiple values of the sensitivity-awareness ϵ_i and tolerance level γ_k. To alleviate further the risk of biased results due to the MOGAs being stuck at local optimum and not synthesising a global optimum Pareto front, we performed multiple independent runs. Although this scenario never occurred in our experiments, when detected, it can be solved by re-initialising the sub-population outside the Pareto front. Also, Algorithm 1 ensures that the Pareto front monotonically improves at each iteration. Finally, we enable replication by making all

Table 3
Time (mean ± SD) in minutes for the synthesis using 10,000 evaluations for one-level CPU parallelisation. **#states** (**#trans.**): number of states (transitions) of the underlying pCTMC. $|K|$: number of continuous parameters.

Model	#states	#trans.	CPU (#cores)			
			1	2	5	10
WC ($\|K\| = 3$)	3440–8960	18,656–49424	394 ± 25	217 ± 29	118 ± 14	68 ± 8
PC ($\|K\| = 2$)	5632	21,968–24572	251 ± 46	131 ± 33	50 ± 2	31 ± 5
GFS v1 ($\|K\| = 2$)	1323–2406	7825–15545	390 ± 27	267 ± 49	125 ± 19	71 ± 10
GFS v2 ($\|K\| = 2$)	21,606	145,335–148245	19,011 ± 400	8207 ± 361	4562 ± 36	2399 ± 9

Table 4
Time (mean ± SD) in minutes for the synthesis using 10,000 evaluations for two-level CPU + GPU parallelisation.

	CPU (#cores)			CPU (#cores)/GPU (#devices)				
Model	1	2	5	1/1	2/1	5/1	2/2	5/2
GFS v2	19,011 ± 400	8207 ± 361	4562 ± 36	2264 ± 33	1736 ± 8	1625 ± 16	1082 ± 3	1043 ± 22

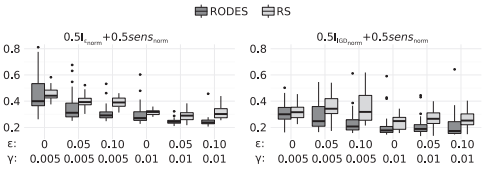


Fig. 15. RODES vs. random search (RS) comparison for combinations of $\gamma \in \{0.005, 0.01\}$ and $\epsilon \in \{0, 0.05, 0.10\}$, over 30 independent GFS runs. For both metrics – I_ϵ indicator and sensitivity (left) and I_{IGD} indicator and sensitivity (right) – smaller is better.

experimental results publicly available on the project webpage.

External validity threats might exist if the search for robust designs for other systems cannot be expressed as a *p*CTMC synthesis problem using objective functions (8) and constraints (9). We limit these threats by specifying *p*CTMCs in an extended variant of the widely used modelling language of PRISM (Kwiatkowska et al., 2011), with objective functions and constraints specified in the established temporal logic CSL. PRISM parametric Markov models are increasingly used to model software architectures, e.g. in the emerging field of self-adaptive software (Calinescu and Kwiatkowska, 2009; Calinescu et al., 2015; Moreno et al., 2015; Gerasimou et al., 2014). Another threat might occur if our method generated a Pareto front that approached the actual Pareto front insufficiently, producing only low quality designs or designs that did not satisfy the required quality constraints. We mitigated this threat by using established Pareto-front performance indices to confirm the quality of the Pareto fronts from our case studies. Nevertheless, additional experiments are needed to establish the applicability and feasibility of the method in domains with characteristics different from those used in our evaluation.

7. Related work

RODES builds on the significant body of *software performance and reliability engineering* research that employs formal models to analyse the quality attributes of alternative software designs (e.g. Balsamo et al., 2004; Bondy, 2014; Becker et al., 2009; Fiondella and Puliafito, 2016; Stewart, 2009; Woodside et al., 2014). Approaches based on formal models such as queueing networks (Balsamo et al., 2003), Petri nets (Lindemann, 1998), stochastic models (Calinescu et al., 2016; Sharma and Trivedi, 2007) and timed automata (Hessel et al., 2008; Larsen, 2014), and tools for their simulation (e.g. Palladio (Becker et al., 2009)) and verification (e.g. PRISM (Kwiatkowska et al., 2011) and UPPAAL (Hessel et al., 2008)) have long been used for this analysis. However, unlike RODES, these approaches can only analyse alternative models through a tedious iterative process carried out manually by experts.

Performance antipatterns can be used to speed up this process by avoiding the analysis of poor designs (Arcelli et al., 2012; Smith and Williams, 2000; Cortellessa et al., 2010), but approaches that automate the search for correct or optimal designs have only been proposed recently. Three types of such approaches are related to RODES. Given a Markov model that violates a quality requirement, the first approach—called *probabilistic model repair* (Bartocci et al., 2011; Chen et al., 2013)—automatically adjusts its transition probabilities to produce a "repaired" model that meets the requirement. The second approach is called *precise parameter synthesis* (Češka et al., 2017), and works by identifying transition rates that enable continuous-time Markov models to satisfy a quality requirement or to optimise a quality attribute of the system under development. Finally, our previous work on *probabilistic model synthesis* (Gerasimou et al., 2015) applies multi-objective optimisation and genetic algorithms to a design template that captures alternative system designs, and generates the Pareto-optimal set of Markov models associated with the quality optimisation criteria of the system. While these approaches represent a significant advance over the previously manual methods of alternative design analysis, they do not take into account the robustness of their repaired or synthesised models. Likewise, the approach from Martens et al. (2010) employs evolutionary algorithms to search the configuration space of Palladio Component Models, but the synthesis process does not reflect the sensitivity of the candidate models.

Syntax-guided synthesis has been used to find probabilistic programs that best match the available data (Nori et al., 2015), including synthesis from "sketches", i.e. partial programs with incomplete details (Solar-Lezama et al., 2005). In Solar-Lezama et al. (2006), counter-example guided inductive synthesis (CEGIS) has been introduced as an SMT-based synthesiser for sketches and, due to the enormous improvement of SMT solvers in the last decade, CEGIS is currently able to find deterministic programs for a variety of challenging problems (Solar-Lezama et al., 2005; 2008). Very recently, the concept of meta-sketches introducing the "optimal synthesis problem" has been proposed (Bornholt et al., 2016) and adapted for synthesis of stochastic reaction networks (Cardelli et al., 2017). These solutions are complementary to RODES, as they explore other aspects of design alternatives, and do not take robustness into account.

Methods that rigorously evaluate how the transition probabilities affect the satisfiability of temporal properties (expressed as probabilistic temporal logic formulae) have been studied in the context of parameter synthesis. The methods either construct symbolic expressions describing the satisfaction probability as a function of the model parameters (Dehnert et al., 2015; Hahn et al., 2011), or compute—for given intervals of parameter values—safe bounds on the satisfaction probability (Quatmann et al., 2016). In contrast to this work, our robust design synthesis directly integrates sensitivity analysis into the automated design process.

Another research area related to RODES is *sensitivity analysis*, which analyses the impact of parameter changes on the performance, reliability, cost and other quality attributes of the system under development (e.g. Gokhale and Trivedi, 2002; Lo et al., 2005; Huang and Lyu, 2005). However, sensitivity analysis typically operates by sampling the parameter space and evaluating the system quality attributes for the sampled values. As such, the result is not guaranteed to reflect the whole range of quality-attribute values for the parameter region of interest. RODES does not have this drawback, as it operates with close over-approximations of the quality-attribute regions for the synthesised robust designs. The perturbation theory for Markov processes has been applied to analysing the sensitivity of software operational profiles (Kamavaram and Goseva-Popstojanova, 2003). However, this approach quantifies the effect of variations in model transition probabilities without synthesising the analysed solutions. Furthermore, RODES supports a wide range of continuous and discrete parameters that cannot be used with the approach from Kamavaram and Goseva-Popstojanova (2003). Stochastic analysis of architectural models was

used for early predictions of system component reliability and sensitivity with respect to different operational profiles (Cheung et al., 2008). Unlike RODES, the research from Cheung et al. (2008) focuses on exploiting different architectural models and associated analysis techniques, and is therefore complementary to the work presented in our paper.

The *smoothed model checking* technique from Bortolussi et al. (2016a) computes an analytical approximation of the satisfaction probability of a formula over a parametric CTMC. While not providing the same guarantees as the safe over-approximation method from RODES, the technique was experimentally shown to be highly accurate, so it can be used to estimate the sensitivity of a probabilistic temporal logic property to variations in the CTMC parameters.

Finally, the problem of parameter synthesis of stochastic reaction networks with respect to multi-objective specification has been recently considered in Bortolussi et al. (2016b). The authors employ statistical methods to estimate how kinetic parameters affect the satisfaction probability and average robustness of Signal Temporal Logic properties. In contrast to our approach, a candidate solution from Bortolussi et al. (2016b) has all parameters fixed and the robustness captures how far the candidate is from violating the particular properties.

8. Conclusion

Robustness is a key and yet insufficiently explored characteristic of software designs, as it can mitigate the unavoidable discrepancies between real systems and their models. We presented RODES, a tool-supported method for the automated synthesis of Pareto-optimal probabilistic models corresponding to robust software designs.

RODES integrates for the first time search-based synthesis and parameter analysis for parametric Markov chains. Our RODES tool automates the application of the method, and provides multi-core as well as GPU-based parallelisation that significantly speeds up the design synthesis process. We performed an extensive experimental evaluation of RODES on three case studies from different application domains. These experiments showed that the sensitivity-aware Pareto-optimal design sets synthesised by RODES enable the selection of robust designs with a wide range of quality-attribute values and provide insights into the system dynamics. The experiments also demonstrate that the parallelisation ensures scalability with respect to the complexity of the systems under development.

In our future work, we will assess the effectiveness of Pareto-dominance relations defined over intervals, and we will augment RODES with alternative multiobjective optimisation techniques such as particle swarm optimisation Reyes-Sierra and Coello (2006). In addition, we are planning to extend the RODES modelling language (and the underpinning search method) with support for syntax-based synthesis Alur et al. (2013) of robust designs from partial *p*CTMC specifications, including sketches of chemical reaction networks Cardelli et al. (2017).

Appendix A. Proof of theorem 1

We show that the sensitivity-aware Pareto dominance relation defined in Definition 4 is a strict order.

Proof. We need to show that relation $\prec$ from (12) is irreflexive and transitive. For any $d \in \mathscr{F}$, $d \prec d$ would require that $f_i(d) < (1+\epsilon_i)f_i(d)$ or $f_i(d) < f_i(d)$ for some $i \in I$, which is impossible. Thus, $\prec$ is irreflexive. To show that $\prec$ is transitive, consider three designs d, d', $d'' \in \mathscr{F}$ such that $d \prec d'$ and $d' \prec d''$. According to (12), we have $\forall i \in I. f_i(d) \le f_i(d')$ and $\forall i \in I. f_i(d') \le f_i(d'')$, so $\forall i \in I. f_i(d) \le f_i(d'')$ due to the transitivity of $\le$. Furthermore, at least one half of the disjunction from definition (12) must hold for each of $d' \prec d''$ and $d' \prec d''$. We have three cases. Assume first that the left half holds for $d \prec d'$, i.e. that $(1+\epsilon_{i_1})f_{i_1}(d) < f_{i_1}(d')$ for some $i_1 \in I$; as $f_{i_1}(d') \le f_{i_1}(d'')$, we also have $(1+\epsilon_{i_1})f_{i_1}(d) < f_{i_1}(d'')$, so $d \prec d''$ in this case. Assume now that left half of disjunction (12) holds for $d' \prec d''$, i.e., that $(1+\epsilon_{i_1})f_{i_1}(d') < f_{i_1}(d'')$ for some $i_1 \in I$; as $f_{i_1}(d) \le f_{i_1}(d')$, we again have $(1+\epsilon_{i_1})f_{i_1}(d) < f_{i_1}(d'')$ and $d \prec d''$. Finally, consider that only the right half of disjunction (12) holds for both $d \prec d'$ and $d \prec d'$. In this last case, $sens(d) \le sens(d') \le sens(d'')$ and there is an $i_1 \in I$ such that $f_{i_1}(d) < f_{i_1}(d') \le f_{i_1}(d'')$, so also $d \prec d''$, and therefore $\prec$ is transitive. □

Appendix B. Complexity analysis

The time complexity of Algorithm 1 representing the synthesis process is

$$\mathscr{O}(k{\cdot}N{\cdot}(|I|+|J|){\cdot}t + k{\cdot}|I|{\cdot}N^2),$$

where k is the number of iterations of the (MOGA) while loop (i.e. the number of generations); $N = |CD|$ is the size of the candidate design population; $|I| + |J|$ is the overall number of objective functions and constraints; and t is the time required to analyse a quality attribute of a candidate design. The term $k{\cdot}N{\cdot}(|I|+|J|){\cdot}t$ quantifies the overall complexity of evaluating candidate designs, while $k \cdot |I| \cdot N^2$ corresponds to comparing designs and building the front in lines 7–14 of Algorithm 1.

The factor t depends on the size of the underlying state space and on the number of discrete-time steps required to evaluate the particular quality attributes. As shown in Češka et al. (2017), $t = \mathscr{O}(t_{CSL}{\cdot}t_{pCSL})$. The factor $t_{CSL} = |\phi|{\cdot}M{\cdot}q{\cdot}t_{\max}$ is the worst-case time complexity of time-bounded CSL model checking Kwiatkowska et al. (2007), where $|\phi|$ is the length of the input CSL formula ϕ, $t_{\max}$ is the highest time bound occurring in it, M is the number of non-zero elements in the rate matrix and q is the highest rate in the matrix. The factor t_{pCSL} is due to the parametric analysis of the design and depends on the form of polynomials appearing in the parametric rate matrix $\mathscr{D}'_{\mathbf{R}}$. Models of software systems typically include only linear polynomials, for which $t_{pCSL} = \mathscr{O}(n)$, where n is the number of continuous parameters.

References

Alur, R., Bodik, R., Juniwal, G., et al., 2013. Syntax-guided synthesis. FMCAD'13. pp. 1–8.
Arcelli, D., Cortellessa, V., Trubiani, C., 2012. Antipattern-based model refactoring for software performance improvement. QoSA'12. pp. 33–42.
Baier, C., Hahn, E.M., Haverkort, B., et al., 2013. Model checking for performability. Math. Struct. Comp. Sci. 23 (4), 751–795.
Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M., 2004. Model-based performance prediction in software development: a survey. IEEE Trans. Softw. Eng. 30 (5), 295–310.
Balsamo, S., Personè, V.D.N., Inverardi, P., 2003. A review on queueing network models with finite capacity queues for software architectures performance prediction. Perform. Eval. 51 (2), 269–288.
Bartocci, E., Grosu, R., Katsaros, P., et al., 2011. Model repair for probabilistic systems. TACAS'11. pp. 326–340.
Becker, S., Koziolek, H., Reussner, R., 2009. The palladio component model for model-driven performance prediction. J. Syst. Softw. 82 (1).
Bondy, A.B., 2014. Foundations of Software and System Performance Engineering. Addison Wesley.
Bornholt, J., Torlak, E., Grossman, D., Ceze, L., 2016. Optimizing synthesis with metasketches. Proc. of POPL'16. ACM, pp. 775–788.

Bortolussi, L., Milios, D., Sanguinetti, G., 2016. Smoothed model checking for uncertain continuous-time markov chains. Inf. Comput. 247, 235–253.
Bortolussi, L., Policriti, A., Silvetti, S., 2016. Logic-based multi-objective design of chemical reaction networks. HSB'16. Springer, pp. 164–178.
Calinescu, R., Gerasimou, S., Banks, A., 2015. Self-adaptive software with decentralised control loops. FASE. pp. 235–251.
Calinescu, R., Ghezzi, C., Johnson, K., et al., 2016. Formal verification with confidence intervals to establish quality of service properties of software systems. IEEE Trans. Rel. 65 (1), 107–125.
Calinescu, R., Kwiatkowska, M., 2009. CADS*: Computer-aided development of self-* systems. FASE. pp. 421–424.
Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N., 2017. Designing robust software systems through parametric Markov chain synthesis. ICSA. IEEE, pp. 131–140.
Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N., 2017. RODES: A robust-design synthesis tool for probabilistic systems. QEST. pp. 304–308.
Cardelli, L., Češka, M., Fränzle, M., Kwiatkowska, M., Laurenti, L., Paoletti, N., Whitby, M., 2017. Syntax-guided optimal synthesis for chemical reaction networks. CAV'17. Springer, pp. 375–395.
Češka, M., Dannenberg, F., Paoletti, N., et al., 2017. Precise parameter synthesis for stochastic biochemical systems. Acta Informatica 54, 589–623.
Češka, M., Pilař, P., Paoletti, N., Brim, L., Kwiatkowska, M., 2016. PRISM-PSY: Precise GPU-accelerated Parameter Synthesis for Stochastic Systems. TACAS'16. Springer, pp. 367–384.
Chen, T., Hahn, E.M., Han, T., et al., 2013. Model repair for Markov decision processes. TASE'13. pp. 85–92.
Cheung, L., Roshandel, R., Medvidovic, N., Golubchik, L., 2008. Early prediction of software component reliability. ICSE. ACM, pp. 111–120.
Cortellessa, V., Martens, A., Reussner, R., Trubiani, C., 2010. A process to effectively identify 'guilty' performance antipatterns. FASE'10. pp. 368–382.
Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comp. 6 (2), 182–197.
Dehnert, C., Junges, S., Jansen, N., et al., 2015. PROPhESY: a probabilistic parameter synthesis tool. CAV'15. Springer, pp. 214–231.
Dehnert, C., Junges, S., Katoen, J.-P., Volk, M., 2017. A STORM is coming: a modern probabilistic model checker. CAV'17. Springer, pp. 592–600.
Filieri, A., Tamburrelli, G., Ghezzi, C., 2016. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Trans. Softw. Eng. 42 (1), 75–99.
Fiondella, L., Puliafito, A., 2016. Principles of Performance and Reliability Modeling and Evaluation. Springer Series in Reliability Engineering.
Gerasimou, S., Calinescu, R., Banks, A., 2014. Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration. SEAMS. pp. 115–124.
Gerasimou, S., Tamburrelli, G., Calinescu, R., 2015. Search-based synthesis of probabilistic models for QoS software engineering. ASE'15. pp. 319–330.
Ghemawat, S., Gobioff, H., Leung, S.-T., 2003. The google file system. SOSP'03. pp. 29–43.
Gokhale, S.S., Trivedi, K.S., 2002. Reliability prediction and sensitivity analysis based on software architecture. ISSRE'03. pp. 64–75.
Hahn, E.M., Hermanns, H., Zhang, L., 2011. Probabilistic reachability for parametric markov models. STTT 13 (1), 3–19.
Han, T., Katoen, J., Mereacre, A., 2008. Approximate parameter synthesis for probabilistic time-bounded reachability. RTSS. pp. 173–182.
Harman, M., Mansouri, S.A., Zhang, Y., 2012. Search-based software engineering: trends, techniques and applications. ACM Comp. Surv. 45 (1), 11:1–11:61.
Harman, M., McMinn, P., de Souza, J., Yoo, S., 2012. Search based software engineering: techniques, taxonomy, tutorial. Emp. Softw. Eng. and Verif. pp. 1–59.
Haverkort, B.R., Hermanns, H., Katoen, J.-P., 2000. On the use of model checking techniques for dependability evaluation. SRDS'00. IEEE, pp. 228–237.
Hessel, A., Larsen, K.G., Mikucionis, M., et al., 2008. Testing real-time systems using UPPAAL. Formal Methods and Testing. Springer, pp. 77–117.
Huang, C.-Y., Lyu, M.R., 2005. Optimal testing resource allocation, and sensitivity analysis in software development. Trans. Reliab. 54 (4), 592–603.
International Organization for Standardization, 2010. ISO 286–1:2010, Geometrical product specifications (GPS) – ISO code system for tolerances on linear sizes.
International Organization for Standardization, 2013. ISO general purpose metric screw threads – Tolerances.
Kamavaram, S., Goseva-Popstojanova, K., 2003. Sensitivity of software usage to changes in the operational profile. NASA Soft. Eng. Workshop.
Kitano, H., 2004. Biological robustness. Nat. Rev. Genet. 5, 826–837.
Koza, J.R., 1992. Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press.
Kwiatkowska, M., Norman, G., Parker, D., 2007. Stochastic model checking. SFM'07. pp. 220–270.
Kwiatkowska, M., Norman, G., Parker, D., 2011. PRISM 4.0: verification of probabilistic real-time systems. CAV'11. pp. 585–591.
Larsen, K.G., 2014. Verification and performance analysis of embedded and cyber-physical systems using uppaal. MODELSWARD'14.
Lindemann, C., 1998. Performance modelling with deterministic and stochastic petri nets. Perform. Eval. Rev. 26 (2), 3.
Lo, J.-H., Huang, C.-Y., Chen, I.-Y., et al., 2005. Reliability assessment and sensitivity analysis of software reliability growth modeling based on software module structure. J. Syst. Softw. 76 (1), 3–13.
Martens, A., Koziolek, H., Becker, S., Reussner, R., 2010. Automatically improve software architecture models for performance, reliability, and cost using evolutionary algorithms. WOSP/SIPEW. pp. 105–116.
Moreno, J.C., Lopes, A., Garlan, D., Schmerl, B., 2015. Impact models for architecture-based self-adaptive systems. FACS. pp. 89–107.
Nebro, A.J., Durillo, J.J., Luna, F., et al., 2009. MOCell: A cellular genetic algorithm for multiobjective optimization. J. Intell. Syst. 24 (7), 726–746.
Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D., 2015. Efficient synthesis of probabilistic programs. PLDI'14. pp. 208–217.
Phadke, M.S., 1995. Quality Engineering using Robust Design. Prentice Hall PTR.
Quatmann, T., Dehnert, C., Jansen, N., et al., 2016. Parameter synthesis for Markov models: faster than ever. ATVA'16. pp. 50–67.
Reyes-Sierra, M., Coello, C.C., 2006. Multi-objective particle swarm optimizers: a survey of the state-of-the-art. Int. J. Comput. Intell. Res. 2 (3), 287–308.
Sharma, V.S., Trivedi, K.S., 2007. Quantifying software performance, reliability and security: an architecture-based approach. J. Syst. Softw. 80 (4), 493–509.
Smith, C.U., Williams, L.G., 2000. Software performance antipatterns. Workshop on Software and Performance. 17. pp. 127–136.
Solar-Lezama, A., Jones, C.G., Bodik, R., 2008. Sketching concurrent data structures. Proc. of PLDI'08. ACM, pp. 136–148.
Solar-Lezama, A., Rabbah, R., Bodík, R., Ebcioğlu, K., 2005. Programming by sketching for bit-streaming programs. Proc. of PLDI'05. ACM, pp. 281–294.
Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V., 2006. Combinatorial sketching for finite programs. Proc. of ASPLOS'06. ACM, pp. 404–415.
Stewart, W.J., 2009. Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. Princeton University Press.
Woodside, M., Petriu, D., Merseguer, J., Petriu, D., Alhaj, M., 2014. Transformation challenges: from software models to performance models. J. Softw. Syst. Model. 13 (4), 1529–1552.
Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C., da Fonseca, V., 2003. Performance assessment of multiobjective optimizers: an analysis and review. IEEE Trans. Evol. Comp. 7 (2), 117–132.

Radu Calinescu is a senior lecturer within the Department of Computer Science at the University of York, UK. His main research interests are in formal methods for self-adaptive, autonomous, secure and dependable software systems, and in performance and reliability software engineering. He is an active promoter of formal methods at runtime as a way to improve the integrity and predictability of selfadaptive and autonomous software systems and processes.

Milan Češka has completed his Ph.D. thesis in the area of data-parallel algorithms for model checking at the Faculty of Informatics, Masaryk University, Brno, Czech Republic in Jun 2012. Afterwards, he was a research assistant in the Systems Biology Laboratory at the same faculty under supervision of prof. Lubos Brim. From February 2014, he was a postdoctoral researcher at Department of Computer Science at Oxford University in the group led by prof. Marta Kwiatkowska. In May 2016, he returned to Czech Republic as an assistant professor at Faculty of Information Technology, Brno University of Technology. His current research interests include formal methods for automated design of probabilistic and approximate systems, and formal analysis of biochemical systems.

Simos Gerasimou is a research associate within the Department of Computer Science at the University of York. He received his BSc in Computer Science from the University of Cyprus, Cyprus, in 2010, and his MSc in Software Engineering and PhD in Computer Science from the University of York, UK, in 2011 and 2017, respectively. His research interests lie in the area of self-adaptive and autonomous systems with a focus on methods that enable dependable system adaptation. Other research interests include runtime quantitative verification, search-based software engineering, model-driven engineering, and artificial intelligence.

Marta Kwiatkowska is Professor of Computing Systems and Fellow of Trinity College, University of Oxford. Prior to this she was Professor in the School of Computer Science at the University of Birmingham, Lecturer at the University of Leicester and Assistant Professor at the Jagiellonian University in Cracow, Poland. Kwiatkowska has made fundamental contributions to the theory and practice of model checking for probabilistic systems, focusing on automated techniques for verification and synthesis from quantitative specifications. She led the development of the PRISM model checker (www.prismmodelchecker.org), the leading software tool in the area and winner of the HVC Award 2016. Probabilistic model checking has been adopted in many diverse fields, including distributed computing, wireless networks, security, robotics, game theory, systems biology, DNA computing and nanotechnology, with genuine flaws found and corrected in real-world protocols. Kwiatkowska awarded an honorary doctorate from KTH Royal Institute of technology in Stockholm in 2014 and the Royal Society Milner Medal in 2018. He recent work was supported by the ERC Advanced Grant VERIWARE "From software verification to 'everyware' verification" and the EPSRC Programme Grant on Mobile Autonomy. She is a Fellow of ACM and Member of Academia Europea.

Nicola Paoletti is a postdoctoral associate at the Department of Computer Science at Stony Brook University. He obtained an MSc in Computer Science in 2010 and a Ph.D in Information Sciences and Complex Systems in 2014 at the University of Camerino, Italy. From 2014 to 2016, he was a post-doctoral researcher at the Department of Computer Science, University of Oxford, UK. His research interests are in the verification, control, and synthesis of stochastic and hybrid systems, with application to biological and biomedical systems.

International Journal on Software Tools for Technology Transfer
https://doi.org/10.1007/s10009-019-00520-8

REGULAR PAPER

Approximate reduction of finite automata for high-speed network intrusion detection

Milan Češka[1] · Vojtěch Havlena[1] · Lukáš Holík[1] · Ondřej Lengál[1] · Tomáš Vojnar[1]

Abstract
We consider the problem of *approximate reduction of non-deterministic automata* that appear in hardware-accelerated network intrusion detection systems (NIDSes). We define an error *distance* of a reduced automaton from the original one as the probability of packets being incorrectly classified by the reduced automaton (wrt the probabilistic distribution of packets in the network traffic). We use this notion to design an *approximate reduction procedure* that achieves a great size reduction (much beyond the state-of-the-art language-preserving techniques) with a controlled and small error. We have implemented our approach and evaluated it on use cases from SNORT, a popular NIDS. Our results provide experimental evidence that the method can be highly efficient in practice, allowing NIDSes to follow the rapid growth in the speed of networks.

Keywords Reduction · Nondeterministic finite automata · Deep packet inspection · High-speed network monitoring

1 Introduction

The recent years have seen a boom in the number of security incidents in computer networks. In order to alleviate the impact of network attacks and intrusions, Internet service providers want to detect malicious traffic at their network's entry points and on the backbones between sub-networks. Software-based network intrusion detection systems (NIDSes), such as the popular open-source system SNORT [50], are capable of detecting suspicious network traffic by testing (among others) whether a packet payload matches a regular expression (regex) describing known patterns of malicious traffic. NIDSes collect and maintain vast databases of such regexes that are typically divided into groups according to types of the attacks and target protocols.

Regex matching is the most computationally demanding task of a NIDS as its cost grows with the speed of the network traffic as well as with the number and complexity of the regexes being matched. The current software-based NIDSes cannot perform the regex matching on networks beyond 1 Gbps [5,28], so they cannot handle the current speed of backbone networks ranging between tens and hundreds of Gbps. A promising approach to speed up NIDSes is to (partially) offload regex matching into hardware [27,28,36]. The hardware then serves as a pre-filter of the network traffic, discarding the majority of the packets from further processing. Such pre-filtering can easily reduce the traffic the NIDS needs to handle by two or three orders of magnitude [28].

Field-programmable gate arrays (FPGAs) are the leading technology in high-throughput regex matching. Due to their inherent parallelism, FPGAs provide an efficient way of implementing *non-deterministic finite automata* (NFAs), which naturally arise from the input regexes. Although the amount of available resources in FPGAs is continually increasing, the speed of networks grows even faster. Working with multi-gigabit networks requires the hardware to use many parallel packet processing branches in a single

Ondřej Lengál
lengal@fit.vutbr.cz
http://www.fit.vutbr.cz/~lengal

Milan Češka
ceskam@fit.vutbr.cz
http://www.fit.vutbr.cz/~ceskam

Vojtěch Havlena
ihavlena@fit.vutbr.cz
http://www.fit.vutbr.cz/~ihavlena

Lukáš Holík
holik@fit.vutbr.cz
http://www.fit.vutbr.cz/~holik

Tomáš Vojnar
vojnar@fit.vutbr.cz
http://www.fit.vutbr.cz/~vojnar

[1] IT4Innovations Centre of Excellence, FIT, Brno University of Technology, Brno, Czech Republic

Published online: 24 May 2019

FPGA [36]; each of them implementing a separate copy of the concerned NFA, and so reducing the size of the NFAs is of the utmost importance. Various language-preserving automata reduction approaches exist, mainly based on computing (bi)simulation relations on automata states (cf. the related work). The reductions they offer, however, do not satisfy the needs of high-speed hardware-accelerated NIDSes.

Our answer to the problem is *approximate reduction* of NFAs, allowing for a trade-off between the achieved reduction and the precision of the regex matching. To formalize the intuitive notion of precision, we propose a novel *probabilistic distance* of automata. It captures the probability that a packet of the input network traffic is incorrectly accepted or rejected by the approximated NFA. The distance assumes a *probabilistic model* of the network traffic. (We show later how such a model can be obtained.)

Having formalized the notion of precision, we specify the target of our reductions as two variants of an optimization problem: (1) minimizing the NFA size given the maximum allowed error (distance from the original), or (2) minimizing the error given the maximum allowed NFA size. Finding such optimal approximations is, however, computationally hard (**PSPACE**-complete, the same as precise NFA minimization).

Consequently, we sacrifice the optimality and, motivated by the typical structure of NFAs that emerge from a set of regexes used by NIDSes (a union of many long "tentacles" with occasional small strongly connected components), we limit the space of possible reductions by restricting the set of operations they can apply to the original automaton. Namely, we consider two reduction operations: (i) collapsing the future of a state into a *self-loop* (this reduction over-approximates the language), or (ii) *removing states* (such a reduction is under-approximating).

The problem of identifying the optimal sets of states on which these operations should be applied is still **PSPACE**-complete. The restricted problem is, however, more amenable to an approximation by a *greedy algorithm*. The algorithm applies the reductions state-by-state in an order determined by a pre-computed *error labelling* of the states. The process is stopped once the given optimization goal in terms of the size or error is reached. The labelling is based on the probability of packets that may be accepted through a given state and hence over-approximates the error that may be caused by applying the reduction at a given state. As our experiments show, this approach can give us high-quality reductions while ensuring formal error bounds.

Finally, it turns out that even the pre-computation of the error labelling of the states is costly (again **PSPACE**-complete). Therefore, we propose several ways to cheaply over-approximate it such that the strong error bound guarantees are still preserved. In particular, we are able to exploit the typical structure of the "union of tentacles" of the hardware NFA in an algorithm that is exponential in the size of the largest "tentacle" only, which gives us a method that is indeed much faster in practice.

We have implemented our approach and evaluated it on regexes used to classify malicious traffic in SNORT. We obtain quite encouraging experimental results demonstrating that our approach provides a much better reduction than language-preserving techniques with an almost negligible error. In particular, our experiments, going down to the level of an actual implementation of NFAs in FPGAs, confirm that we can squeeze into an up-to-date FPGA chip real-life regexes encoding malicious traffic, allowing them to be used with a negligible error for filtering at speeds of 100 Gbps (and even 400 Gbps). This is far beyond what one can achieve with current exact reduction approaches.

This paper is an extended version of the paper that appeared in the proceedings of TACAS'18 [12], containing complete proofs of the presented lemmas and theorems.

Related Work Hardware acceleration for regex matching at the line rate is an intensively studied technology that uses general-purpose hardware [3,4,29–33,49,53] as well as FPGAs [8,14,25,27,28,36,39,45,47]. Most of the works focus on DFA implementation and optimization techniques. NFAs can be exponentially smaller than DFAs but need, in the worst case, $\mathcal{O}(n)$ memory accesses to process each byte of the payload where n is the number of states. In most cases, this incurs an unacceptable slowdown. Several works alleviate this disadvantage of NFAs by exploiting reconfigurability and fine-grained parallelism of FPGAs, allowing one to process one character per clock cycle (e.g. [8,27,28,36,39,45,47]).

In [33], which is probably the closest work to ours, the authors consider a set of regexes describing network attacks. They replace a potentially prohibitively large DFA by a tree of smaller DFAs, an alternative to using NFAs that minimizes the latency occurring in a non-FPGA-based implementation. The language of every DFA-node in the tree over-approximates the languages of its children. Packets are filtered through the tree from the root downwards until they belong to the language of the encountered nodes, and may be finally accepted at the leaves, or are rejected otherwise. The over-approximating DFAs are constructed using a similar notion of probability of an occurrence of a state as in our approach. The main differences from our work are that (1) the approach targets approximation of DFAs (not NFAs), (2) the over-approximation is based on a given traffic sample only (it cannot benefit from a probabilistic model), and (3) no probabilistic guarantees on the approximation error are provided.

Approximation of DFAs was considered in various other contexts. Hyper-minimization is an approach that is allowed to alter language membership of a finite set of words [21,35]. A DFA with a given maximum number of states is constructed in [20], minimizing the error defined either by (i) counting

prefixes of misjudged words up to some length, or (ii) the sum of the probabilities of the misjudged words wrt the Poisson distribution over Σ^*. Neither of these approaches considers reduction of NFAs nor allows to control the expected error with respect to the real traffic.

In addition to the metrics mentioned above when discussing the works [20,21,35], the following metrics should also be mentioned. The Cesaro–Jaccard distance studied in [44] is, in spirit, similar to [20] and does also not reflect the probability of individual words. The edit distance of weighted automata from [41] depends on the minimum edit distance between pairs of words from the two compared languages, again regardless of their statistical significance. One might also consider using the error metric on a pair of automata introduced by Angluin in the setting of PAC (probably approximately correct) learning of DFAs [1], where n words are sampled from a given distribution and their (non-)acceptance tested in the two automata. If the outputs of both automata agree on all n words, one can say that with confidence δ the distance between the two automata is at most ϵ, where δ and ϵ can be determined from n. None of these notions is suitable for our needs.

Language-preserving minimization of a given NFA is a **PSPACE**-complete problem [26,34]. More feasible (polynomial time) is language-preserving size reduction of NFAs based on (bi)simulations [9,13,24,42], which does not aim for a truly minimal NFA. A number of advanced variants exist, based on multi-pebble or look-ahead simulations, or on combinations of forward and backward simulations [15,18,37]. The practical efficiency of these techniques is, however, often insufficient to allow them to handle the large NFAs that occur in practice and/or they do not manage to reduce the NFAs enough. Finally, even a minimal NFA for the given set of regexes is often too big to be implemented in the given FPGA operating on the required speed (as shown even in our experiments). Our approach is capable of a much better reduction for the price of a small change of the accepted language.

2 Preliminaries

We use $\langle a, b\rangle$ to denote the set $\{x \in \mathbb{R} \mid a \leq x \leq b\}$ and $\mathbb{N}$ to denote the set $\{0, 1, 2, \dots\}$. Given a pair of sets X_1 and X_2, we use $X_1 \triangle X_2$ to denote their *symmetric difference*, i.e. the set $\{x \mid \exists! i \in \{1, 2\} : x \in X_i\}$. We use the notation $[v_1, \dots, v_n]$ to denote a vector of n elements, $\mathbf{1}$ to denote the all 1's vector $[1, \dots, 1]$ (the dimension of $\mathbf{1}$ is always clear from the context), $\boldsymbol{A}$ to denote a matrix, and $\boldsymbol{A}^\top$ for its transpose, and $\boldsymbol{I}$ for the identity matrix.

In the following, we fix a finite non-empty alphabet Σ. A *non-deterministic finite automaton* (NFA) is a quadruple $\mathcal{A} = (Q, \delta, I, F)$ where Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $I \subseteq Q$ is a set of initial states, and $F \subseteq Q$ is a set of accepting states. We use $Q[\mathcal{A}]$, $\delta[\mathcal{A}]$, $I[\mathcal{A}]$, and $F[\mathcal{A}]$ to denote Q, δ, I, and F, respectively, and $q \xrightarrow{a} q'$ to denote that $q' \in \delta(q, a)$. Often, we abuse notation and treat δ as a subset of $Q \times \Sigma \times 2^Q$. A sequence of states $\rho = q_0 \cdots q_n$ is a *run* of A over a word $w = a_1 \cdots a_n \in \Sigma^*$ from a state q to a state q', denoted as $q \overset{w,\rho}{\rightsquigarrow} q'$, if $\forall 1 \leq i \leq n : q_{i-1} \xrightarrow{a_i} q_i$, $q_0 = q$, and $q_n = q'$. Sometimes, we use ρ in set operations where it behaves as the set of states it contains. We also use $q \overset{w}{\rightsquigarrow} q'$ to denote that $\exists \rho \in Q^* : q \overset{w,\rho}{\rightsquigarrow} q'$ and $q \rightsquigarrow q'$ to denote that $\exists w : q \overset{w}{\rightsquigarrow} q'$. The *language* of a state q is defined as $L_{\mathcal{A}}(q) = \{w \mid \exists q_F \in F : q \overset{w}{\rightsquigarrow} q_F\}$ and its *banguage* (back-language) is defined as $L^{\flat}_{\mathcal{A}}(q) = \{w \mid \exists q_I \in I : q_I \overset{w}{\rightsquigarrow} q\}$. Both notions can be naturally extended to a set $S \subseteq Q$: $L_{\mathcal{A}}(S) = \bigcup_{q \in S} L_{\mathcal{A}}(q)$ and $L^{\flat}_{\mathcal{A}}(S) = \bigcup_{q \in S} L^{\flat}_{\mathcal{A}}(q)$. We drop the subscript A when the context is obvious. $\mathcal{A}$ *accepts* the language $L(\mathcal{A})$ defined as $L(\mathcal{A}) = L_{\mathcal{A}}(I)$. $\mathcal{A}$ is called *deterministic* (DFA) if $|I| = 1$ and $\forall q \in Q$ and $\forall a \in \Sigma : |\delta(q, a)| \leq 1$, and *unambiguous* (UFA) if $\forall w \in L(\mathcal{A}) : \exists! q_I \in I, \rho \in Q^*, q_F \in F : q_I \overset{w,\rho}{\rightsquigarrow} q_F$.

The *restriction* of $\mathcal{A}$ to $S \subseteq Q$ is an NFA $\mathcal{A}_{|S}$ given as $\mathcal{A}_{|S} = (S, \delta \cap (S \times \Sigma \times 2^S), I \cap S, F \cap S)$. We define the *trim* operation as $trim(\mathcal{A}) = \mathcal{A}_{|C}$ where $C = \{q \mid \exists q_I \in I, q_F \in F : q_I \rightsquigarrow q \rightsquigarrow q_F\}$. For a set of states $R \subseteq Q$, we use $reach(R)$ to denote the set of states reachable from R, $reach(R) = \{r' \mid \exists r \in R : r \rightsquigarrow r'\}$. We use the number of states of $\mathcal{A}$ as a measure of its size, i.e. $|\mathcal{A}| = |Q|$.

A (discrete probability) *distribution* over a countable set X is a mapping $\Pr : X \to \langle 0, 1\rangle$ such that $\sum_{x \in X} \Pr(x) = 1$. An n-state *probabilistic automaton* (PA) over Σ is a triple $\mathcal{P} = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$ where $\boldsymbol{\alpha} \in \langle 0, 1\rangle^n$ is a vector of *initial weights*, $\boldsymbol{\gamma} \in \langle 0, 1\rangle^n$ is a vector of *final weights*, and for every $a \in \Sigma$, $\boldsymbol{\Delta}_a \in \langle 0, 1\rangle^{n \times n}$ is a *transition matrix* for symbol a. We abuse notation and use $Q[\mathcal{P}]$ to denote the set of states $Q[\mathcal{P}] = \{1, \dots, n\}$. Moreover, the following two properties need to hold: (i) $\sum\{\boldsymbol{\alpha}[i] \mid i \in Q[\mathcal{P}]\} = 1$ (the initial probability is 1) and (ii) for every state $i \in Q[\mathcal{P}]$ it holds that $\sum\{\boldsymbol{\Delta}_a[i, j] \mid j \in Q[\mathcal{P}], a \in \Sigma\} + \boldsymbol{\gamma}[i] = 1$. (The probability of accepting or leaving a state is 1.) We define the *support* of $\mathcal{P}$ as the NFA $supp(\mathcal{P}) = (Q[\mathcal{P}], \delta[\mathcal{P}], I[\mathcal{P}], F[\mathcal{P}])$ s.t.

$$\delta[\mathcal{P}] = \{(i, a, j) \mid \boldsymbol{\Delta}_a[i, j] > 0\},$$
$$I[\mathcal{P}] = \{i \mid \boldsymbol{\alpha}[i] > 0\},$$
$$F[\mathcal{P}] = \{i \mid \boldsymbol{\gamma}[i] > 0\}.$$

Let us assume that every PA $\mathcal{P}$ is such that $supp(\mathcal{P}) = trim(supp(\mathcal{P}))$. For a word $w = a_1 \dots a_k \in \Sigma^*$, we use $\boldsymbol{\Delta}_w$ to denote the matrix $\boldsymbol{\Delta}_{a_1} \cdots \boldsymbol{\Delta}_{a_k}$. For the empty word ε, we define $\boldsymbol{\Delta}_\varepsilon = \boldsymbol{I}$. It can be easily shown that $\mathcal{P}$ represents a distribution over words $w \in \Sigma^*$ defined as $\Pr_{\mathcal{P}}(w) = \boldsymbol{\alpha}^\top \cdot \boldsymbol{\Delta}_w \cdot \boldsymbol{\gamma}$. We call $\Pr_{\mathcal{P}}(w)$ the *probability* of w in $\mathcal{P}$. Given a language $L \subseteq \Sigma^*$, we define the probability of L in P as $\Pr_{\mathcal{P}}(L) = \sum_{w \in L} \Pr_{\mathcal{P}}(w)$.

In some of the proofs later, we use the PA $\mathcal{P}_{Exp}$ defined as $\mathcal{P}_{Exp} = (\mathbf{1}, [\mu], \{[\mu]_a\}_{a\in\Sigma})$ where $\mu = \frac{1}{|\Sigma|+1}$. $\mathcal{P}_{Exp}$ models a distribution over the words from Σ^* using a combination of an exponential distribution (for selecting the length l of a word) and the uniform distribution (for selecting symbols in a word of the length l). In particular, the purpose of $\mathcal{P}_{Exp}$ is to assign every word $w \in \Sigma^*$ the (nonzero) probability $\Pr_{\mathcal{P}_{Exp}}(w) = \mu^{|w|+1}$; any other PA assigning nonzero probabilities to all words would work as well.

If Conditions (i) and (ii) from the definition of PAs are dropped, we speak about a *pseudo-probabilistic automaton (PPA)*, which may assign a word from its support a quantity that is not necessarily in the range $\langle 0, 1\rangle$, denoted as the *significance* of the word below. PPAs may arise during some of our operations performed on PAs. Note that PPAs can be seen as instantiations of multiplicity or weighted automata [46].

3 Approximate reduction of NFAs

In this section, we first introduce the key notion of our approach: a *probabilistic distance* of a pair of finite automata wrt a given probabilistic automaton that, intuitively, represents the significance of particular words. We discuss the complexity of computing the probabilistic distance. Finally, we formulate two problems of *approximate automata reduction via probabilistic distance*.

3.1 Probabilistic distance

We start by defining our notion of a probabilistic distance of two NFAs. Assume NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ and a probabilistic automaton $\mathcal{P}$ specifying the distribution $\Pr_{\mathcal{P}} : \Sigma^* \to \langle 0, 1\rangle$. The *probabilistic distance* $d_{\mathcal{P}}(\mathcal{A}_1, \mathcal{A}_2)$ between $\mathcal{A}_1$ and $\mathcal{A}_2$ wrt $\Pr_{\mathcal{P}}$ is defined as

$$d_{\mathcal{P}}(\mathcal{A}_1, \mathcal{A}_2) = \Pr_{\mathcal{P}}(L(\mathcal{A}_1) \triangle L(\mathcal{A}_2)).$$

Intuitively, the distance captures the significance of the words accepted by one of the automata only. We use the distance to drive the reduction process towards automata with small errors and to assess the quality of the result. (The distance is sometimes called the *symmetric difference semi-metric* [17].)

The value of $\Pr_{\mathcal{P}}(L(\mathcal{A}_1) \triangle L(\mathcal{A}_2))$ can be computed as follows. Using the fact that (1) $L_1 \triangle L_2 = (L_1 \backslash L_2) \uplus (L_2 \backslash L_1)$ and (2) $L_1 \backslash L_2 = L_1 \backslash (L_1 \cap L_2)$, we get

$$\begin{aligned} &d_{\mathcal{P}}(\mathcal{A}_1, \mathcal{A}_2) \\ &= \Pr_{\mathcal{P}}(L(\mathcal{A}_1) \backslash L(\mathcal{A}_2)) + \Pr_{\mathcal{P}}(L(\mathcal{A}_2) \backslash L(\mathcal{A}_1)) \\ &= \Pr_{\mathcal{P}}(L(\mathcal{A}_1) \backslash (L(\mathcal{A}_1) \cap L(\mathcal{A}_2))) \\ &\quad + \Pr_{\mathcal{P}}(L(\mathcal{A}_2) \backslash (L(\mathcal{A}_2) \cap L(\mathcal{A}_1))) \\ &= \Pr_{\mathcal{P}}(L(\mathcal{A}_1)) + \Pr_{\mathcal{P}}(L(\mathcal{A}_2)) - 2 \cdot \Pr_{\mathcal{P}}(L(\mathcal{A}_1) \cap L(\mathcal{A}_2)). \end{aligned}$$

Hence, the key step is to compute $\Pr_{\mathcal{P}}(L(\mathcal{A}))$ for an NFA $\mathcal{A}$ and a PA $\mathcal{P}$. Problems similar to computing such a probability have been extensively studied in several contexts including verification of probabilistic systems [2,6,52].

In our approach, we apply the method of [6] and compute $\Pr_{\mathcal{P}}(L(\mathcal{A}))$ in the following way. We first check whether the NFA $\mathcal{A}$ is unambiguous. This can be done by using the standard product construction (denoted as $\cap$) for computing the intersection of the NFA $\mathcal{A}$ with itself and trimming the result, formally $\mathcal{B} = \mathit{trim}(\mathcal{A} \cap \mathcal{A})$, followed by a check whether there is some state $(p, q) \in Q[\mathcal{B}]$ s.t. $p \neq q$ [40]. If $\mathcal{A}$ is ambiguous, we either determinize it or disambiguate it [40], leading to a DFA/UFA $\mathcal{A}'$, respectively.[1] Then, we construct the trimmed product of $\mathcal{A}'$ and $\mathcal{P}$ (this can be seen as computing $\mathcal{A}' \cap \mathit{supp}(\mathcal{P})$ while keeping the probabilities from $\mathcal{P}$ on the edges of the result), yielding a PPA $\mathcal{R} = (\boldsymbol{\alpha}_{\mathcal{R}}, \boldsymbol{\gamma}_{\mathcal{R}}, \{\boldsymbol{\Delta}^{\mathcal{R}}_a\}_{a\in\Sigma})$.[2] Intuitively, $\mathcal{R}$ represents not only the words of $L(\mathcal{A})$ but also their probability in $\mathcal{P}$ (we give the formal definition of $\mathcal{R}$ inside the proof of Lemma 2). Now, let $\boldsymbol{\Delta} = \sum_{a\in\Sigma} \boldsymbol{\Delta}_a$ be the matrix that expresses, for any $p, q \in Q[\mathcal{R}]$, the significance of getting from p to q via any $a \in \Sigma$. Further, it can be shown (cf. the proof of Lemma 1) that the matrix $\boldsymbol{\Delta}^*$, representing the significance of going from p to q via any $w \in \Sigma^*$, can be computed as $(\boldsymbol{I} - \boldsymbol{\Delta})^{-1}$. Then, to get $\Pr_{\mathcal{P}}(L(\mathcal{A}))$, it suffices to take $\boldsymbol{\alpha}^\top \cdot \boldsymbol{\Delta}^* \cdot \boldsymbol{\gamma}$. Note that, due to the determinization/disambiguation step, the obtained value indeed is $\Pr_{\mathcal{P}}(L(\mathcal{A}))$ despite $\mathcal{R}$ being a PPA. The two lemmas below summarize the complexity of this step for NFAs and UFAs, respectively.

Lemma 1 *Let $\mathcal{P}$ be a PA and $\mathcal{A}$ an NFA. The problem of computing $\Pr_{\mathcal{P}}(L(\mathcal{A}))$ is **PSPACE**-complete.*

Proof The membership in **PSPACE** can be shown as follows. The computation described above corresponds to solving a linear equation system. The system has an exponential size because of the blowup caused by the determinization/disambiguation of $\mathcal{A}$ required by the product construction. The equation system can, however, be constructed by a **PSPACE** transducer $\mathcal{M}_{eq}$. Moreover, as solving linear equation systems can be done using a polylogarithmic-space transducer $\mathcal{M}_{SysLin}$, one can combine these two transducers to obtain a **PSPACE** algorithm. Details of the construction follow:

First, we construct a transducer $\mathcal{M}_{eq}$ that, given an NFA $\mathcal{A} = (Q_{\mathcal{A}}, \delta_{\mathcal{A}}, I_{\mathcal{A}}, F_{\mathcal{A}})$ and a PA $\mathcal{P} = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a\in\Sigma})$ on its input, constructs a system of $m = 2^{|Q_{\mathcal{A}}|} \cdot |Q[\mathcal{P}]|$ linear equations $\mathcal{S}(\mathcal{A}, \mathcal{P})$ of m unknowns $\xi_{[R,p]}$ for $R \subseteq Q_{\mathcal{A}}$ and $p \in Q[\mathcal{P}]$ representing the product of $\mathcal{A}'$ and $\mathcal{P}$, where $\mathcal{A}'$

[1] In theory, disambiguation can produce smaller automata, but, in our experiments, determinization proved to work better.

[2] $\mathcal{R}$ is not necessarily a PA since there might be transitions in $\mathcal{P}$ that are either removed or copied several times in the product construction.

is a deterministic automaton obtained from $\mathcal{A}$ using the standard subset construction. The system of equations $\mathcal{S}(\mathcal{A}, \mathcal{P})$ is defined as follows (cf. [6]):

$$\xi_{[R,p]} = \begin{cases} 0 & \text{if } L_{\mathcal{A}}(R) \cap L_{\mathcal{P}'}(p) = \emptyset, \\ \sum_{a \in \Sigma} \sum_{p' \in Q[\mathcal{P}]} \left(\boldsymbol{\Delta}_a[p, p'] \cdot \xi_{[\delta_{\mathcal{A}}(R,a),p']} \right) + \boldsymbol{\gamma}[p] & \\ & \text{if } R \cap F_{\mathcal{A}} \neq \emptyset, \\ \sum_{a \in \Sigma} \sum_{p' \in Q[\mathcal{P}]} \boldsymbol{\Delta}_a[p, p'] \cdot \xi_{[\delta_{\mathcal{A}}(R,a),p']} & \\ & \text{otherwise,} \end{cases}$$

such that $\mathcal{P}' = supp(\mathcal{P})$ and $\delta_{\mathcal{A}}(R, a) = \bigcup_{r \in R} \delta(r, a)$. The test $L_{\mathcal{A}}(R) \cap L_{\mathcal{P}'}(p) = \emptyset$ can be performed by checking $\exists r \in R : L_{\mathcal{A}}(r) \cap L_{\mathcal{P}'}(p) = \emptyset$, which can be done in polynomial time.

It holds that $\Pr_{\mathcal{P}}(L(\mathcal{A})) = \sum_{p \in Q[\mathcal{P}]} \boldsymbol{\alpha}[p] \cdot \xi_{[I_{\mathcal{A}},p]}$. Although the size of $\mathcal{S}(\mathcal{A}, \mathcal{P})$ (which is the output of $\mathcal{M}_{eq}$) is exponential in the size of the input of $\mathcal{M}_{eq}$, the internal configuration of $\mathcal{M}_{eq}$ only needs to be of polynomial size, i.e. $\mathcal{M}_{eq}$ works in **PSPACE**. Note that the size of each equation is at most polynomial.

Given a system $\mathcal{S}$ of m linear equations with m unknowns, solving $\mathcal{S}$ can be done in the time $\mathcal{O}(\log^2 m)$ using $\mathcal{O}(m^k)$ processors for a fixed k [16, Corollary 2] (i.e. it is in the class **NC**).[3] According to [19, Lemma 1b], an $\mathcal{O}(\log^2 m)$ time-bounded parallel machine can be simulated by an $\mathcal{O}(\log^4 m)$ space-bounded Turing machine. Therefore, there exists an $\mathcal{O}(\log^4 m)$ space-bounded Turing machine $\mathcal{M}_{SysLin}$ that solves a system of m linear equations with m unknowns. As a consequence, $\mathcal{M}_{SysLin}$ can solve $\mathcal{S}(\mathcal{A}, \mathcal{P})$ using the space

$$\begin{aligned} \mathcal{O}(\log^4(2^{|Q_{\mathcal{A}}|} \cdot |Q[\mathcal{P}]|)) &= \mathcal{O}(\log^4 2^{|Q_{\mathcal{A}}|} + \log^4 |Q[\mathcal{P}]|) \\ &= \mathcal{O}(|Q_{\mathcal{A}}|^4 + \log^4 |Q[\mathcal{P}]|). \end{aligned}$$

The missing part is how to combine $\mathcal{M}_{eq}$ and $\mathcal{M}_{SysLin}$ to avoid using the exponential-size output tape of $\mathcal{M}_{eq}$. For this, we use the following standard technique for combining reductions [43, Proposition 8.2].

We take turns in simulating $\mathcal{M}_{SysLin}$ and $\mathcal{M}_{eq}$. We start with simulating $\mathcal{M}_{SysLin}$. When $\mathcal{M}_{SysLin}$ moves its head right, we pause it and simulate $\mathcal{M}_{eq}$ until it outputs the corresponding bit, which is fed into the input of $\mathcal{M}_{SysLin}$. Then we pause $\mathcal{M}_{eq}$ and resume the run of $\mathcal{M}_{SysLin}$. On the other hand, when $\mathcal{M}_{SysLin}$ moves its head left (from the k-th position on the tape), we pause it, restart $\mathcal{M}_{eq}$ from its initial state, and simulate it until it outputs the $(k-1)$-st bit of its output tape, and then pause $\mathcal{M}_{eq}$ and return the control to $\mathcal{M}_{SysLin}$. In order to keep track of the position k of the head of $\mathcal{M}_{SysLin}$ on its tape, we use a binary counter.

[3] We use $\log k$ to denote the base-2 logarithm of k.

The internal configuration of both $\mathcal{M}_{eq}$ and $\mathcal{M}_{SysLin}$ is of a polynomial size and the overhead of keeping track of the position of the head of $\mathcal{M}_{SysLin}$ also requires only polynomial space. Therefore, the whole transducer runs in a polynomially bounded space.

The **PSPACE**-hardness is obtained by a reduction from the (**PSPACE**-complete) universality of NFAs: using the PA $\mathcal{P}_{Exp}$ defined in Sect. 2, which assigns every word a nonzero probability. it holds that

$$L(\mathcal{A}) = \Sigma^* \quad \text{iff} \quad \Pr_{\mathcal{P}_{Exp}}(L(\mathcal{A})) = 1.$$

□

Lemma 2 *Let $\mathcal{P}$ be a PA and $\mathcal{A}$ a UFA. The problem of computing $\Pr_{\mathcal{P}}(L(\mathcal{A}))$ is in **PTIME**.*

Proof We modify the proof from [6] into our setting. First, we give a formal definition of the *product* of a PA $\mathcal{P} = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$ and an NFA $\mathcal{A} = (Q, \delta, I, F)$ as the $(|Q[\mathcal{P}]| \cdot |Q|)$-state PPA $\mathcal{R} = (\boldsymbol{\alpha}_{\mathcal{R}}, \boldsymbol{\gamma}_{\mathcal{R}}, \{\boldsymbol{\Delta}_a^{\mathcal{R}}\}_{a \in \Sigma})$ where[4]

$$\begin{aligned} \boldsymbol{\alpha}_{\mathcal{R}}[(q_{\mathcal{P}}, q_{\mathcal{A}})] &= \boldsymbol{\alpha}_{\mathcal{R}}[q_{\mathcal{P}}] \cdot |\{q_{\mathcal{A}}\} \cap I|, \\ \boldsymbol{\gamma}_{\mathcal{R}}[(q_{\mathcal{P}}, q_{\mathcal{A}})] &= \boldsymbol{\gamma}_{\mathcal{R}}[q_{\mathcal{P}}] \cdot |\{q_{\mathcal{A}}\} \cap F|, \\ \boldsymbol{\Delta}_a^{\mathcal{R}}[(q_{\mathcal{P}}, q_{\mathcal{A}}), (q'_{\mathcal{P}}, q'_{\mathcal{A}})] &= \boldsymbol{\Delta}_a[q_{\mathcal{P}}, q'_{\mathcal{P}}] \cdot |\{q'_{\mathcal{A}}\} \cap \delta(q_{\mathcal{A}}, a)|. \end{aligned}$$

Note that $\mathcal{R}$ is not necessarily a PA any more because for $w \in \Sigma^*$ such that $\Pr_{\mathcal{P}}(w) > 0$, (i) if $w \notin L(A)$, then $\Pr_{\mathcal{R}}(w) = 0$ and (ii) if $w \in L(\mathcal{A})$ and $\mathcal{A}$ can accept w using n different runs, then $\Pr_{\mathcal{R}}(w) = n \cdot \Pr_{\mathcal{P}}(w)$. As a consequence, the probabilities of all words from Σ^* are no longer guaranteed to add up to 1. If $\mathcal{A}$ is unambiguous, the second issue is avoided and $\mathcal{R}$ preserves the probabilities of words from $L(\mathcal{A})$, i.e. $\Pr_{\mathcal{R}}(w) = \Pr_{\mathcal{P}}(w)$ for all $w \in L(\mathcal{A})$, so $\mathcal{R}$ can be seen as the restriction of $\Pr_{\mathcal{P}}$ to $L(\mathcal{A})$. In the following, we assume $\mathcal{R}$ is trimmed.

In order to compute $\Pr_{\mathcal{P}}(L(\mathcal{A}))$, we construct a matrix $\boldsymbol{E}$ defined as $\boldsymbol{E} = \sum_{a \in \Sigma} \boldsymbol{\Delta}_a^{\mathcal{R}}$. Because $\mathcal{R}$ is trimmed, the *spectral radius* of $\boldsymbol{E}$, denoted as $\rho(\boldsymbol{E})$, is less than one, i.e. $\rho(\boldsymbol{E}) < 1$. (The proof of this fact can be found, for example, in [6].) Intuitively, $\rho(\boldsymbol{E}) < 1$ holds because we trimmed the redundant states from the product of $\mathcal{P}$ and $\mathcal{A}$. We further use the following standard result in linear algebra: if $\rho(\boldsymbol{E}) < 1$, then (i) the matrix $\boldsymbol{I} - \boldsymbol{E}$ is invertible and (ii) the sum of powers of $\boldsymbol{E}$, denoted as $\boldsymbol{E}^*$, can be computed as $\boldsymbol{E}^* = \sum_{i=0}^{\infty} \boldsymbol{E}^i = (\boldsymbol{I} - \boldsymbol{E})^{-1}$ [23]. Moreover, note that matrix inversion can be done in polynomial time [48].

$\boldsymbol{E}^*$ represents the reachability between nodes of $\mathcal{R}$, i.e. $\boldsymbol{E}^*[r, r']$ is the sum of significances of all (possibly infinitely

[4] We assume an implicit bijection between states of the product $\mathcal{R}$ and $\{1, \ldots, |Q[\mathcal{R}]|\}$.

many) paths from r to r' in $\mathcal{R}$. When related to $\mathcal{P}$ and $\mathcal{A}$, the matrix $\boldsymbol{E}^*$ represents the reachability in $\mathcal{P}$ wrt $L(\mathcal{A})$, i.e.

$$\boldsymbol{E}^*[(q_\mathcal{P}, q_\mathcal{A}), (q'_\mathcal{P}, q'_\mathcal{A})] = \sum \left\{ \boldsymbol{\Delta}_w[q_\mathcal{P}, q'_\mathcal{P}] \;\middle|\; q_\mathcal{A} \overset{w}{\rightsquigarrow} q'_\mathcal{A}, w \in \Sigma^* \right\}. \tag{1}$$

We prove Equation (1) using the following reasoning. First, we show that

$$\boldsymbol{E}^n[(q_\mathcal{P}, q_\mathcal{A}), (q'_\mathcal{P}, q'_\mathcal{A})] = \sum \left\{ \boldsymbol{\Delta}_w[q_\mathcal{P}, q'_\mathcal{P}] \;\middle|\; q_\mathcal{A} \overset{w}{\rightsquigarrow} q'_\mathcal{A}, w \in \Sigma^n \right\}, \tag{2}$$

i.e. $\boldsymbol{E}^n$ represents the reachability in $\mathcal{P}$ wrt $L(\mathcal{A})$ for words of length n. We prove Equation (2) by induction on n: For $n = 0$, the equation follows from the fact that $\boldsymbol{E}^0 = \boldsymbol{I}$. For $n = 1$, the equation follows directly from the definition of $\mathcal{R}$ and $\boldsymbol{\Delta}$. Next, suppose that Equation (2) holds for $n > 1$; we show that it holds also for $n + 1$. We start with the following reasoning:

$$\begin{aligned} &b{E}^{n+1}[(q_\mathcal{P}, q_\mathcal{A}), (q'_\mathcal{P}, q'_\mathcal{A})] \\ &= \boldsymbol{E}^n \boldsymbol{E})[(q_\mathcal{P}, q_\mathcal{A}), (q'_\mathcal{P}, q'_\mathcal{A})] \\ &= sum\Big\{ \boldsymbol{E}^n[(q_\mathcal{P}, q_\mathcal{A}), (q''_\mathcal{P}, q''_\mathcal{A})] \cdot \boldsymbol{E}[(q''_\mathcal{P}, q''_\mathcal{A}), \\ &\quad (q'_\mathcal{P}, q'_\mathcal{A})] \;\Big|\; (q''_\mathcal{P}, q''_\mathcal{A}) \in Q[\mathcal{R}] \Big\}. \end{aligned}$$

The last line is obtained via the definition of matrix multiplication. Further, using the induction hypothesis, we get

$$\begin{aligned} &b{E}^{n+1}[(q_\mathcal{P}, q_\mathcal{A}), (q'_\mathcal{P}, q'_\mathcal{A})] \\ &= sum\Big\{ \sum \left\{ \boldsymbol{\Delta}_w[q_\mathcal{P}, q''_\mathcal{P}] \;\middle|\; q_\mathcal{A} \overset{w}{\rightsquigarrow} q''_\mathcal{A}, w \in \Sigma^n \right\} \cdot \\ &\quad \sum \left\{ \boldsymbol{\Delta}_a[q''_\mathcal{P}, q'_\mathcal{P}] \;\middle|\; q''_\mathcal{A} \overset{a}{\rightarrow} q'_\mathcal{A}, a \in \Sigma \right\} \;\Big| \\ &\quad (q''_\mathcal{P}, q''_\mathcal{A}) \in Q[\mathcal{R}] \Big\} \\ &= \sum \Big\{ \sum \left\{ \boldsymbol{\Delta}_w[q_\mathcal{P}, q''_\mathcal{P}] \cdot \boldsymbol{\Delta}_a[q''_\mathcal{P}, q'_\mathcal{P}] \;\middle|\; q_\mathcal{A} \overset{w}{\rightsquigarrow} q''_\mathcal{A}, \right. \\ &\quad \left. q''_\mathcal{A} \overset{a}{\rightarrow} q'_\mathcal{A}, a \in \Sigma, w \in \Sigma^n \right\} \;\Big|\; (q''_\mathcal{P}, q''_\mathcal{A}) \in Q[\mathcal{R}] \Big\} \\ &= \sum \left\{ \boldsymbol{\Delta}_{w'}[q_\mathcal{P}, q'_\mathcal{P}] \;\middle|\; q_\mathcal{A} \overset{w'}{\rightsquigarrow} q'_\mathcal{A}, w' \in \Sigma^{n+1} \right\}. \end{aligned}$$

Since $\boldsymbol{E}^* = \sum_{i=0}^{\infty} \boldsymbol{E}^i$, Equation (1) follows. Using the matrix $\boldsymbol{E}^*$, it remains to compute $\Pr_\mathcal{P}(L(\mathcal{A}))$ as

$$\Pr_\mathcal{P}(L(\mathcal{A})) = \boldsymbol{\alpha}_\mathcal{R}^\top \cdot \boldsymbol{E}^* \cdot \boldsymbol{\gamma}_\mathcal{R}.$$

□

3.2 Automata reduction using probabilistic distance

We now exploit the probabilistic distance introduced above to formulate the task of approximate reduction of NFAs as two optimization problems. Given an NFA $\mathcal{A}$ and a PA $\mathcal{P}$ specifying the distribution $\Pr_\mathcal{P} : \Sigma^* \to \langle 0, 1 \rangle$, we define

- **size-driven reduction**: for $n \in \mathbb{N}$, find an NFA $\mathcal{A}'$ such that $|\mathcal{A}'| \leq n$ and the distance $d_\mathcal{P}(\mathcal{A}, \mathcal{A}')$ is minimal,
- **error-driven reduction**: for $\epsilon \in \langle 0, 1 \rangle$, find an NFA $\mathcal{A}'$ such that $d_\mathcal{P}(\mathcal{A}, \mathcal{A}') \leq \epsilon$ and the size $|\mathcal{A}'|$ is minimal.

The following lemma shows that the natural decision problem underlying both of the above optimization problems is **PSPACE**-complete, which matches the complexity of computing the probabilistic distance as well as that of the *exact* reduction of NFAs [26].

Lemma 3 *Consider an NFA $\mathcal{A}$, a PA $\mathcal{P}$, a bound on the number of states $n \in \mathbb{N}$, and an error bound $\epsilon \in \langle 0, 1 \rangle$. It is **PSPACE**-complete to determine whether there exists an NFA $\mathcal{A}'$ with n states s.t. $d_\mathcal{P}(\mathcal{A}, \mathcal{A}') \leq \epsilon$.*

Proof Membership in **PSPACE**: We non-deterministically generate an automaton $\mathcal{A}'$ with n states and test (in **PSPACE**, as shown in Lemma 1) that $d_\mathcal{P}(\mathcal{A}, \mathcal{A}') \leq \epsilon$. This shows the problem is in **NPSPACE** = **PSPACE**.

PSPACE-hardness: We use a reduction from the problem of checking universality of an NFA $\mathcal{A} = (Q, \delta, I, F)$ over Σ, i.e. from checking whether $L(\mathcal{A}) = \Sigma^*$, which is **PSPACE**-complete. First, for a reason that will become clear later, we test if $\mathcal{A}$ accepts all words over Σ of length 0 and 1, which can be done in polynomial time. It holds that $L(\mathcal{A}) = \Sigma^*$ iff there is a 1-state NFA $\mathcal{A}'$ s.t. $d_{\mathcal{P}_{Exp}}(\mathcal{A}, \mathcal{A}') \leq 0$. ($\mathcal{P}_{Exp}$ is defined in Sect. 2.) The implication from left to right is clear: $\mathcal{A}'$ can be constructed as $\mathcal{A}' = (\{q\}, \{q \xrightarrow{a} q \mid a \in \Sigma\}, \{q\}, \{q\})$. To show the reverse implication, we note that we have tested that $\{\epsilon\} \cup \Sigma \subseteq L(\mathcal{A})$. Since the probability of any word from $\{\epsilon\} \cup \Sigma \subseteq L(\mathcal{A})$ in $\mathcal{P}_{Exp}$ is nonzero, the only 1-state NFA that processes those words with zero error is the NFA $\mathcal{A}'$ defined above. Because the language of $\mathcal{A}'$ is $L(\mathcal{A}') = \Sigma^*$, it holds that $d_{\mathcal{P}_{Exp}}(\mathcal{A}, \mathcal{A}') \leq 0$ iff $L(\mathcal{A}) = \Sigma^*$. □

The notions defined above do not distinguish between introducing a *false positive* ($\mathcal{A}'$ accepts a word $w \notin L(\mathcal{A})$) or a *false negative* ($\mathcal{A}'$ rejects a word $w \in L(\mathcal{A})$) answers. To this end, we define *over-approximating* and *under-approximating* reductions as reductions for which the conditions $L(\mathcal{A}) \subseteq L(\mathcal{A}')$ and $L(\mathcal{A}) \supseteq L(\mathcal{A}')$ hold.

A naïve solution to the reductions would enumerate all NFAs $\mathcal{A}'$ of sizes from 0 up to k (resp. $|\mathcal{A}|$), for each of them compute $d_\mathcal{P}(\mathcal{A}, \mathcal{A}')$, and take an automaton with the smallest probabilistic distance (resp. a smallest one satisfying

Algorithm 1: A greedy size-driven reduction

Input : NFA $\mathcal{A} = (Q, \delta, I, F)$, PA $\mathcal{P}$, $n \geq 1$
Output: NFA $\mathcal{A}'$, $\epsilon \in \mathbb{R}$ s.t. $|\mathcal{A}'| \leq n$ and $d_{\mathcal{P}}(\mathcal{A}, \mathcal{A}') \leq \epsilon$

```
V ← ∅;
for q ∈ Q in the order ⪯_{A,label(A,P)} do
    V ← V ∪ {q}; A' ← reduce(A, V);
    if |A'| ≤ n then break
return A', ε = error(A, V, label(A, P));
```

the restriction on $d_{\mathcal{P}}(\mathcal{A}, \mathcal{A}')$). Obviously, this approach is computationally infeasible.

4 A heuristic approach to approximate reduction

In this section, we introduce two techniques for approximate reduction of NFAs that avoid the need to iterate over all automata of a certain size. The first approach is based on under-approximating the automata by removing states—we call it the *pruning reduction*—while the second approach is based on over-approximating the automata by adding self-loops to states and removing redundant states—we call it the *self-loop reduction*. Finding an optimal automaton using these reductions is also **PSPACE**-complete, but more amenable to heuristics like greedy algorithms. We start with introducing two high-level greedy algorithms, one for the size- and one for the error-driven reduction, and follow by showing their instantiations for the pruning and the self-loop reduction. A crucial role in the algorithms is played by a function that labels states of the automata by an estimate of the error that will be caused when some of the reductions is applied at a given state.

4.1 A general algorithm for size-driven reduction

Algorithm 1 shows a general greedy method for performing the size-driven reduction. In order to use the same high-level algorithm in both directions of reduction (over-/under-approximating), it is parameterized with the functions: *label*, *reduce*, and *error*. The real intricacy of the procedure is hidden inside these three functions. Intuitively, $label(\mathcal{A}, \mathcal{P})$ assigns every state of an NFA $\mathcal{A}$ an approximation of the error that will be caused wrt the PA $\mathcal{P}$ when a reduction is applied at this state, while the purpose of $reduce(\mathcal{A}, V)$ is to create a new NFA $\mathcal{A}'$ obtained from $\mathcal{A}$ by introducing some error at states from V.[5] Further, $error(\mathcal{A}, V, label(\mathcal{A}, \mathcal{P}))$ estimates the error introduced by the application of $reduce(\mathcal{A}, V)$, possibly in a more precise (and costly) way than by just summing the concerned error labels: Such a computation is possible outside of the main computation loop. We show instantiations of these functions later, when discussing the reductions used. Moreover, the algorithm is also parameterized with a total order $\preceq_{\mathcal{A},label(\mathcal{A},\mathcal{P})}$ that defines which states of $\mathcal{A}$ are processed first and which are processed later. The ordering may take into account the pre-computed labelling. The algorithm accepts an NFA $\mathcal{A}$, a PA $\mathcal{P}$, and $n \in \mathbb{N}$ and outputs a pair consisting of an NFA $\mathcal{A}'$ of the size $|\mathcal{A}'| \leq n$ and an error bound ϵ such that $d_{\mathcal{P}}(\mathcal{A}, \mathcal{A}') \leq \epsilon$.

The main idea of the algorithm is that it creates a set V of states where an error is to be introduced. V is constructed by starting from an empty set and adding states to it in the order given by $\preceq_{\mathcal{A},label(\mathcal{A},\mathcal{P})}$, until the size of the result of $reduce(\mathcal{A}, V)$ has reached the desired bound n (in our setting, *reduce* is always antitone, i.e. for $V \subseteq V'$, it holds that $|reduce(\mathcal{A}, V)| \geq |reduce(\mathcal{A}, V')|$). We now define the necessary condition for *label*, *reduce*, and *error* that makes Algorithm 1 correct.

Condition C1 *holds if for every NFA $\mathcal{A}$, PA $\mathcal{P}$, and a set $V \subseteq Q[\mathcal{A}]$, we have that*

(a) $error(\mathcal{A}, V, label(\mathcal{A}, \mathcal{P})) \geq d_{\mathcal{P}}(\mathcal{A}, reduce(\mathcal{A}, V))$,
(b) $|reduce(\mathcal{A}, Q[\mathcal{A}])| \leq 1$, *and*
(c) $reduce(\mathcal{A}, \emptyset) = \mathcal{A}$.

C1(a) ensures that the error computed by the reduction algorithm indeed over-approximates the exact probabilistic distance, **C1**(b) is a boundary condition for the case when the reduction is applied at every state of $\mathcal{A}$, and **C1**(c) ensures that when no error is to be introduced at any state, we obtain the original automaton.

Lemma 4 *Algorithm 1 is correct if **C1** holds.*

Proof Follows straightforwardly from Condition **C1**. □

4.2 A general algorithm for error-driven reduction

In Algorithm 2, we provide a high-level method of computing the error-driven reduction. The algorithm is in many ways similar to Algorithm 1; it also computes a set of states V where an error is to be introduced, but an important difference is that we compute an approximation of the error in each step and only add q to V if it does not raise the error over the threshold ϵ. Note that the *error* does not need to be monotone, so it may be advantageous to traverse all states from Q and not terminate as soon as the threshold is reached. The correctness of Algorithm 2 also depends on **C1**.

Lemma 5 *Algorithm 2 is correct if **C1** holds.*

Proof Follows straightforwardly from Condition **C1**. □

[5] We emphasize that this does not mean that states from V will be simply removed from $\mathcal{A}$—the performed operation depends on the particular reduction.

Algorithm 2: A greedy error-driven reduction.

Input : NFA $\mathcal{A} = (Q, \delta, I, F)$, PA $\mathcal{P}$, $\epsilon \in \langle 0, 1\rangle$
Output: NFA $\mathcal{A}'$ s.t. $d_{\mathcal{P}}(\mathcal{A}, \mathcal{A}') \leq \epsilon$

$\ell \leftarrow label(\mathcal{A}, P)$;
$V \leftarrow \emptyset$;
for $q \in Q$ *in the order* $\preceq_{\mathcal{A},label(\mathcal{A},\mathcal{P})}$ **do**
  $e \leftarrow error(\mathcal{A}, V \cup \{q\}, \ell)$;
  if $e \leq \epsilon$ **then** $V \leftarrow V \cup \{q\}$
return $\mathcal{A}' = reduce(\mathcal{A}, V)$;

4.3 Pruning reduction

The pruning reduction is based on identifying a set of states to be removed from an NFA $\mathcal{A}$, under-approximating the language of $\mathcal{A}$. In particular, for $\mathcal{A} = (Q, \delta, I, F)$, the pruning reduction finds a set $R \subseteq Q$ and restricts $\mathcal{A}$ to $Q \backslash R$, followed by removing useless states, to construct a reduced automaton $\mathcal{A}' = trim(\mathcal{A}_{|Q\backslash R})$. Note that the natural decision problem corresponding to this reduction is also **PSPACE**-complete.

Lemma 6 *Consider an NFA $\mathcal{A}$, a PA $\mathcal{P}$, a bound on the number of states $n \in \mathbb{N}$, and an error bound $\epsilon \in \langle 0, 1\rangle$. It is **PSPACE**-complete to determine whether there exists a subset of states $R \subseteq Q[\mathcal{A}]$ of size $|R| = n$ such that $d_{\mathcal{P}}(\mathcal{A}, \mathcal{A}_{|R}) \leq \epsilon$.*

Proof Membership in **PSPACE**: We non-deterministically generate a subset R of $Q[\mathcal{A}]$ having n states and test (in **PSPACE**, as shown in Lemma 1) that $d_{\mathcal{P}}(\mathcal{A}, \mathcal{A}_{|R}) \leq \epsilon$. This shows the problem is in **NPSPACE** = **PSPACE**.

PSPACE-hardness: We use a reduction from the **PSPACE**-complete problem of checking universality of an NFA $\mathcal{A} = (Q, \delta, I, F)$ over Σ. Consider a symbol $x \notin \Sigma$. Let us construct an NFA $\mathcal{A}'$ over $\Sigma \cup \{x\}$ s.t. $L(\mathcal{A}') = x^*.L(\mathcal{A})$. $\mathcal{A}'$ is constructed by adding a fresh state q_{new} to $\mathcal{A}$ that can loop over x and make a transition to any initial state of $\mathcal{A}$ over x: $\mathcal{A}' = (Q \uplus \{q_{new}\}, \delta \cup \{q_{new} \xrightarrow{x} q \mid q \in I \cup \{q_{new}\}\}, I \cup \{q_{new}\}, F)$. We set $n = |\mathcal{A}'| + 1$. Further, we also construct an $(n+1)$-state NFA $\mathcal{B}$ accepting the language $x^n.\Sigma^*$ defined as $\mathcal{B} = (Q_{\mathcal{B}}, \delta_{\mathcal{B}}, \{q_1\}, \{q_{n+1}\})$ where $Q_{\mathcal{B}} = \{q_1, \ldots, q_{n+1}\}$ and $\delta_{\mathcal{B}} = \{q_i \xrightarrow{x} q_{i+1} \mid 1 \leq i \leq n\} \cup \{q_{n+1} \xrightarrow{a} q_{n+1} \mid a \in \Sigma\}$. Moreover, let $\mathcal{P}$ be a PA representing a distribution $\Pr_{\mathcal{P}}$ that is defined for each $w \in (\Sigma \cup \{x\})^*$ as

$$\Pr_{\mathcal{P}}(w) = \begin{cases} \mu^{|w'|+1} & \text{for } w = x^n.w',\ w' \in \Sigma^*, \\ & \text{and } \mu = \frac{1}{|\Sigma|+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

Note that $\Pr_{\mathcal{P}}(x^n.w) = \Pr_{\mathcal{P}_{Exp}}(w)$ for $w \in \Sigma^*$, and $\Pr_{\mathcal{P}}(u) = 0$ for $u \notin x^n.\Sigma^*$ ($\mathcal{P}$ can be easily constructed from $\mathcal{P}_{Exp}$.) Also note that $\mathcal{B}$ accepts exactly those words w such that $\Pr_{\mathcal{P}}(w) \neq 0$ and that $\Pr_{\mathcal{P}}(L(\mathcal{B})) = 1$. Using the automata defined above, we construct an NFA $\mathcal{C} = \mathcal{A}' \cup \mathcal{B}$ where the union of two NFAs is defined as $\mathcal{A}_1 \cup \mathcal{A}_2 = (Q[\mathcal{A}_1] \uplus Q[\mathcal{A}_2], \delta[\mathcal{A}_1] \uplus \delta[\mathcal{A}_2], I[\mathcal{A}_1] \uplus I[\mathcal{A}_2], F[\mathcal{A}_1] \uplus F[\mathcal{A}_2])$. NFA $\mathcal{C}$ has $2n$ states, the language of $\mathcal{C}$ is $L(\mathcal{C}) = x^*.L(\mathcal{A}) \cup x^n.\Sigma^*$ and its probability is $\Pr_{\mathcal{P}}(L(\mathcal{C})) = 1$.

The important property of $\mathcal{C}$ is that if there exists a set $R \subseteq Q[\mathcal{C}]$ of the size $|R| = n$ s.t. $d_{\mathcal{P}}(\mathcal{C}, \mathcal{C}_{|R}) \leq 0$, then $L(\mathcal{A}) = \Sigma^*$. The property holds because since $|Q[\mathcal{A}']| = n-1$, when we remove n states from $\mathcal{C}$, at least one state from $Q[\mathcal{B}]$ is removed, making the whole subautomaton of $\mathcal{C}$ corresponding to $\mathcal{B}$ useless, and, therefore, $L(\mathcal{C}_{|R}) \subseteq x^*.L(\mathcal{A})$. Because $d_{\mathcal{P}}(\mathcal{C}, \mathcal{C}_{|R}) \leq 0$, we know that $\Pr_{\mathcal{P}}(L(\mathcal{C}_{|R})) = 1$, so $x^n.\Sigma^* \subseteq x^*.L(\mathcal{A}) = L(\mathcal{C}_{|R})$ and, therefore, $L(\mathcal{A}) = \Sigma^*$. For the other direction, if $L(\mathcal{A}) = \Sigma^*$, then there exists a set $R \subseteq Q[\mathcal{A}'] \cup Q[\mathcal{B}]$ of the size $|R| = n$ s.t. $d_{\mathcal{P}}(\mathcal{C}, \mathcal{C}_{|R}) \leq 0$. (In particular, R can be such that $R \subseteq Q[\mathcal{B}]$.) □

Although Lemma 6 shows that the pruning reduction is as hard as a general reduction (cf. Lemma 3), the pruning reduction is more amenable to using heuristics like the greedy algorithms from Sects. 4.1 and 4.2. We instantiate *reduce*, *error*, and *label* in these high-level algorithms in the following way (the subscript p stands for *pruning*):

$$reduce_p(\mathcal{A}, V) = trim(\mathcal{A}_{|Q\backslash V}),$$
$$error_p(\mathcal{A}, V, \ell) = \min_{V' \in \lfloor V \rfloor_p} \sum \{\ell(q) \mid q \in V'\},$$

where $\lfloor V \rfloor_p$ is defined in the rest of this paragraph: Because of the use of *trim* in $reduce_p$, for a pair of sets V, V' s.t. $V \subset V'$, it holds that $reduce_p(\mathcal{A}, V)$ may, in general, yield the same automaton as $reduce_p(\mathcal{A}, V')$. Therefore, in order to obtain a tight approximation, we wish to compute the least error that is obtained when removing the states in V. We define a partial order $\sqsubseteq_p$ on 2^Q as $V_1 \sqsubseteq_p V_2$ iff $reduce_p(\mathcal{A}, V_1) = reduce_p(A, V_2)$ and $V_1 \subseteq V_2$, and use $\lfloor V \rfloor_p$ to denote the set of minimal elements of the set of elements that are smaller than V (wrt $\sqsubseteq_p$). The value of the approximation $error_p(A, V, \ell)$ is therefore the minimum of the sum of errors over all sets from $\lfloor V \rfloor_p$.

Note that the size of $\lfloor V \rfloor_p$ can again be exponential, and thus we employ a greedy approach for guessing an optimal V'. Clearly, this cannot affect the soundness of the algorithm, but only decreases the precision of the bound on the distance. Our experiments indicate that for automata appearing in NIDSes, this simplification has typically only a negligible impact on the precision of the bounds.

For computing the state labelling, we provide the following three functions, which differ in the precision they provide

and the difficulty of their computation (naturally, more precise labellings are harder to compute): $label^1_p$, $label^2_p$, and $label^3_p$. Given an NFA $\mathcal{A}$ and a PA $\mathcal{P}$, they generate the labellings ℓ^1_p, ℓ^2_p, and ℓ^3_p, respectively, defined as

$$\ell^1_p(q) = \sum \left\{ \Pr_{\mathcal{P}}(L^{\flat}_{\mathcal{A}}(q')) \mid q' \in \mathit{reach}(\{q\}) \cap F \right\},$$
$$\ell^2_p(q) = \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(F \cap \mathit{reach}(q)) \right),$$
$$\ell^3_p(q) = \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q) \right).$$

A state label $\ell(q)$ approximates the error of the words removed from $L(\mathcal{A})$ when q is removed. More concretely, $\ell^1_p(q)$ is a rough estimate saying that the error can be bounded by the sum of probabilities of the banguages of all final states reachable from q. (In the worst case, all those final states might become unreachable.) Note that $\ell^1_p(q)$ (1) counts the error of a word accepted in two different final states of $\mathit{reach}(q)$ twice and (2) it also considers words that are accepted in some final state in $\mathit{reach}(q)$ without going through q. The labelling ℓ^2_p deals with (1) by computing the total probability of the banguage of the set of all final states reachable from q, and the labelling ℓ^3_p in addition also deals with (2) by only considering words that traverse through q. (They can, however, be accepted in some final state not in $\mathit{reach}(q)$ by a run completely disjoint from q and $\mathit{reach}(q) \cap F$, so even ℓ^3_p can still be imprecise.) Note that if $\mathcal{A}$ is unambiguous, then $\ell^1_p = \ell^2_p$.

Each state labelling is given as the probability (or the sum of probabilities in the case of ℓ^1_p) of the language related to q. Therefore, when computing the particular label of q, we first modify $\mathcal{A}$ to obtain $\mathcal{A}'$ accepting the language related to the labelling. Then, we compute the value of $\Pr_{\mathcal{P}}(L(\mathcal{A}'))$ using the algorithm from Sect. 3.1. Recall that this step is in general costly, due to the determinization/disambiguation of $\mathcal{A}'$. The key property of the labelling computation resides in the fact that if $\mathcal{A}$ is composed of several disjoint sub-automata, the automaton $\mathcal{A}'$ is typically much smaller than $\mathcal{A}$ and thus the computation of the label is considerably less demanding. Since the automata appearing in regex matching for NIDS are composed of the union of "tentacles", the particular $\mathcal{A}'$s are very small, which enables an efficient component-wise computation of the labels.

The following lemma states the correctness of using the pruning reduction as an instantiation of Algorithms 1 and 2 and also the relation among ℓ^1_p, ℓ^2_p, and ℓ^3_p.

Lemma 7 *For every $x \in \{1, 2, 3\}$, the functions reduce_p, error_p, and label^x_p satisfy **C1**. Moreover, consider an NFA $\mathcal{A}$, a PA $\mathcal{P}$, and let $\ell^x_p = \mathit{label}^x_p(\mathcal{A}, \mathcal{P})$ for $x \in \{1, 2, 3\}$. Then, for each $q \in Q[\mathcal{A}]$, we have $\ell^1_p(q) \geq \ell^2_p(q) \geq \ell^3_p(q)$.*

Proof We start by proving the inequalities $\ell^1_p(q) \geq \ell^2_p(q) \geq \ell^3_p(q)$ for each $q \in Q[\mathcal{A}]$, which will then help us prove the first part of the lemma. The first inequality follows from the fact that if the banguages of reachable final states are not disjoint, in the case of ℓ^1_p, we may sum probabilities of the same words multiple times. The second inequality follows from the inclusion $L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q) \subseteq L^{\flat}_{\mathcal{A}}(F \cap \mathit{reach}(q))$.

Second, we prove that the functions reduce_p, error_p, and label^x_p satisfy the properties of **C1**:

– **C1**(a): In order to show the inequality

$$\mathit{error}_p(\mathcal{A}, V, \mathit{label}^x_p(\mathcal{A}, \mathcal{P})) \geq d_{\mathcal{P}}(\mathcal{A}, \mathit{reduce}_p(\mathcal{A}, V)),$$

we prove it for $\ell^3_p = \mathit{label}^3_p(\mathcal{A}, \mathcal{P})$; the rest follows from $\ell^1_p(q) \geq \ell^2_p(q) \geq \ell^3_p(q)$, which is proved above. Consider some set of states $V \subseteq Q[\mathcal{A}]$ and a set $V' \in \lfloor V \rfloor_p$ s.t. for any $V'' \in \lfloor V \rfloor_p$, it holds that $\sum\{\ell^3_p(q) \mid q \in V'\} \leq \sum\{\ell^3_p(q) \mid q \in V''\}$. We have

$$\begin{aligned}
&L(\mathcal{A}) \triangle L(\mathit{reduce}_p(\mathcal{A}, V)) \\
&\quad = L(\mathcal{A}) \triangle L(\mathit{reduce}_p(\mathcal{A}, V')) && \text{⟅def. of } \sqsubseteq_p \text{⟆} \\
&\quad = \mathit{langof}\,\mathcal{A} \backslash L(\mathit{reduce}_p(\mathcal{A}, V')) \\
&\quad \text{⟅} L(\mathcal{A}) \supseteq L(\mathit{reduce}_p(\mathcal{A}, V')) \text{⟆} \\
&\quad \subseteq \bigcup_{q \in V'} L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q). && \text{⟅def. of } \mathit{reduce}_p \text{⟆}
\end{aligned} \tag{4}$$

Finally, using (4), we obtain

$$\begin{aligned}
&d_{\mathcal{P}}(\mathcal{A}, \mathit{reduce}_p(\mathcal{A}, V)) \\
&= \Pr_{\mathcal{P}}(L(\mathcal{A}) \triangle L(\mathit{reduce}_p(\mathcal{A}, V'))) \\
&\text{⟅def. of } d_{\mathcal{P}} \text{⟆} \\
&\leq \sum_{q \in V'} \Pr_{\mathcal{P}}(L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q)) && \text{⟅(4)⟆} \\
&= \sum\{\ell^3_p(q) \mid q \in V'\} && \text{⟅def. of } \ell^3_p \text{⟆} \\
&= \min_{V'' \in \lfloor V \rfloor_p} \sum\{\ell^3_p(q) \mid q \in V''\} && \text{⟅def. of } V' \text{⟆} \\
&= \mathit{error}_p(\mathcal{A}, V, \ell^3_p). && \text{⟅def. of } \mathit{error}_p \text{⟆}
\end{aligned}$$

– **C1**(b): $|\mathit{reduce}_p(\mathcal{A}, Q[\mathcal{A}])| \leq 1$ because

$$|\mathit{reduce}_p(\mathcal{A}, Q[\mathcal{A}])| = |\mathit{trim}(\mathcal{A}_{|\emptyset})| = 0.$$

– **C1**(c): $\mathit{reduce}_p(\mathcal{A}, \emptyset) = \mathcal{A}$ since

$$\mathit{reduce}_p(\mathcal{A}, \emptyset) = \mathit{trim}(\mathcal{A}_{|Q[\mathcal{A}]}) = \mathcal{A}.$$

(We assume that $\mathcal{A}$ is trimmed at the input.) □

4.4 Self-loop reduction

The main idea of the self-loop reduction is to over-approximate the language of $\mathcal{A}$ by adding self-loops over every symbol at selected states. This makes some states of $\mathcal{A}$ redundant, allowing them to be removed without introducing any more error. Given an NFA $\mathcal{A} = (Q, \delta, I, F)$, the self-loop reduction searches for a set of states $R \subseteq Q$, which will have self-loops added, and removes other transitions leading out of these states, making some states unreachable. The unreachable states are then removed.

Formally, let $sl(\mathcal{A}, R)$ be the NFA $(Q \cup \{s\}, \delta', I, F \cup \{s\})$ where $s \notin Q$ and the transition function δ' is defined such that $\delta'(s, a) = \{s\}$ and, for all states $p \in Q$ and symbols $a \in \Sigma$, $\delta'(p, a) = (\delta(p, a) \setminus R) \cup \{s\}$ if $\delta(p, a) \cap R \neq \emptyset$ and $\delta'(p, a) = \delta(p, a)$ otherwise. Similarly to the pruning reduction, the natural decision problem corresponding to the self-loop reduction is also **PSPACE**-complete.

Lemma 8 *Consider an NFA $\mathcal{A}$, a PA $\mathcal{P}$, a bound on the number of states $n \in \mathbb{N}$, and an error bound $\epsilon \in \langle 0, 1 \rangle$. It is **PSPACE**-complete to determine whether there exists a subset of states $R \subseteq Q[\mathcal{A}]$ of size $|R| = n$ such that $d_{\mathcal{P}}(\mathcal{A}, sl(\mathcal{A}, R)) \leq \epsilon$.*

Proof Membership in **PSPACE** can be proved in the same way as in the proof of Lemma 6.

PSPACE-hardness: We reduce from the **PSPACE**-complete problem of checking universality of an NFA $\mathcal{A} = (Q, \delta, I, F)$. First, we check whether $I[\mathcal{A}] \neq \emptyset$. We have that $L(\mathcal{A}) = \Sigma^*$ iff there exists a set of states $R \subseteq Q$ of the size $|R| = |Q|$ such that $d_{\mathcal{P}_{Exp}}(\mathcal{A}, sl(\mathcal{A}, R)) \leq 0$. (Note that this means that a self-loop is added to every state of $\mathcal{A}$.) □

The required functions in the error- and size-driven reduction algorithms are instantiated in the following way (the subscript *sl* stands for *self-loop*):

$$reduce_{sl}(\mathcal{A}, V) = trim(sl(\mathcal{A}, V)),$$
$$error_{sl}(\mathcal{A}, V, \ell) = \sum \{\ell(q) \mid q \in \min(\lfloor V \rfloor_{sl})\},$$

where $\lfloor V \rfloor_{sl}$ is defined in a similar manner as $\lfloor V \rfloor_p$ in the previous section (using a partial order $\sqsubseteq_{sl}$ defined similarly to $\sqsubseteq_p$; the difference is that in this case, the order $\sqsubseteq_{sl}$ has a single minimal element, though).

The functions $label^1_{sl}$, $label^2_{sl}$, and $label^3_{sl}$ compute the state labellings ℓ^1_{sl}, ℓ^2_{sl}, and ℓ^3_{sl} for an NFA $\mathcal{A}$ and a PA $\mathcal{P}$, which are defined as follows:

$$\ell^1_{sl}(q) = weight_{\mathcal{P}}(L^{\flat}_{\mathcal{A}}(q)),$$
$$\ell^2_{sl}(q) = \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).\Sigma^*\right),$$
$$\ell^3_{sl}(q) = \ell^2_{sl}(q) - \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q)\right).$$

In the definitions above, the function $weight_{\mathcal{P}}(w)$ for a PA $\mathcal{P} = (\boldsymbol{\alpha}, \boldsymbol{\gamma}, \{\boldsymbol{\Delta}_a\}_{a \in \Sigma})$ and a word $w \in \Sigma^*$ is defined as $weight_{\mathcal{P}}(w) = \boldsymbol{\alpha}^{\top} \cdot \boldsymbol{\Delta}_w \cdot \mathbf{1}$ (i.e. similarly as $\Pr_{\mathcal{P}}(w)$ but with the final weights $\boldsymbol{\gamma}$ discarded), and $weight_{\mathcal{P}}(L)$ for $L \subseteq \Sigma^*$ is defined as $weight_{\mathcal{P}}(L) = \sum_{w \in L} weight_{\mathcal{P}}(w)$.

Intuitively, the state labelling $\ell^1_{sl}(q)$ computes the probability that q is reached from an initial state, so if q is pumped up with all possible word endings, this is the maximum possible error introduced by the added word endings. This has the following sources of imprecision: (1) the probability of some words may be included twice, e.g. when $L^{\flat}_{\mathcal{A}}(q) = \{a, ab\}$, the probabilities of all words from $\{ab\}.\Sigma^*$ are included twice in $\ell^1_{sl}(q)$ because $\{ab\}.\Sigma^* \subseteq \{a\}.\Sigma^*$, and (2) $\ell^1_{sl}(q)$ can also contain probabilities of words already accepted on a run traversing q. The state labelling ℓ^2_{sl} deals with (1) by considering the probability of the language $L^{\flat}_{\mathcal{A}}(q).\Sigma^*$, and ℓ^3_{sl} deals also with (2) by subtracting from the result of ℓ^2_{sl} the probabilities of the words that pass through q and are accepted.

The computation of the state labellings for the self-loop reduction is done in a similar way as the computation of the state labellings for the pruning reduction (cf. Sect. 4.3). For a computation of $weight_{\mathcal{P}}(L)$, one can use the same algorithm as for $\Pr_{\mathcal{P}}(L)$, only the final vector for PA $\mathcal{P}$ is set to $\mathbf{1}$. The correctness of Algorithms 1 and 2 when instantiated using the self-loop reduction is stated in the following lemma.

Lemma 9 *For every $x \in \{1, 2, 3\}$, the functions $reduce_{sl}$, $error_{sl}$, and $label^x_{sl}$ satisfy **C1**. Moreover, consider an NFA $\mathcal{A}$, a PA $\mathcal{P}$, and let $\ell^x_{sl} = label^x_{sl}(\mathcal{A}, \mathcal{P})$ for $x \in \{1, 2, 3\}$. Then, for each $q \in Q[\mathcal{A}]$, we have $\ell^1_{sl}(q) \geq \ell^2_{sl}(q) \geq \ell^3_{sl}(q)$.*

Proof First, we prove the inequalities $\ell^1_{sl}(q) \geq \ell^2_{sl}(q) \geq \ell^3_{sl}(q)$ for each $q \in Q[\mathcal{A}]$, which we then use to prove the first part of the lemma. We start with the equality $weight_{\mathcal{P}}(w) = \Pr_{\mathcal{P}}(w.\Sigma^*)$, which follows from the fact that for each state p of $\mathcal{P}$ the sum of probabilities of all words, when considering p as the only initial state of $\mathcal{P}$, is 1. Then, we obtain the equality

$$\sum_{w \in L^{\flat}_{\mathcal{A}}(q)} weight_{\mathcal{P}}(w) = \sum_{w \in L^{\flat}_{\mathcal{A}}(q)} \Pr_{\mathcal{P}}(w.\Sigma^*),$$

which, in turn, implies

$$\begin{aligned} \ell^1_{sl}(q) = weight_{\mathcal{P}}(L^{\flat}_{\mathcal{A}}(q)) &= \sum_{w \in L^{\flat}_{\mathcal{A}}(q)} \Pr_{\mathcal{P}}\left(w.\Sigma^*\right) \\ &\geq \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).\Sigma^*\right) = \ell^2_{sl}(q). \end{aligned} \tag{5}$$

For example, for $L^{\flat}_{\mathcal{A}}(q) = \{w, wa\}$ where $w \in \Sigma^*$ and $a \in \Sigma$, we have

$$\begin{aligned} weight_{\mathcal{P}}(L^{\flat}_{\mathcal{A}}(q)) &= weight_{\mathcal{P}}(\{w, wa\}) \\ &= weight_{\mathcal{P}}(w) + weight_{\mathcal{P}}(wa) \\ &= \Pr_{\mathcal{P}}(w.\Sigma^*) + \Pr_{\mathcal{P}}(wa.\Sigma^*), \end{aligned} \tag{6}$$

while

$$\Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).\Sigma^*\right) = \Pr_{\mathcal{P}}\left(\{w, wa\}.\Sigma^*\right) = \Pr_{\mathcal{P}}\left(w.\Sigma^*\right).$$

The inequality $\ell^2_{sl} \geq \ell^3_{sl}$ holds trivially.

Second, we prove that the functions $reduce_{sl}$, $error_{sl}$, and $label^x_{sl}$ satisfy the properties of **C1**:

- **C1**(a): To show that $error_{sl}(\mathcal{A}, V, label^x_{sl}(\mathcal{A}, \mathcal{P})) \geq d_{\mathcal{P}}(\mathcal{A}, reduce_{sl}(\mathcal{A}, V))$, we prove that the inequality holds for $\ell^3_{sl} = label^3_{sl}(\mathcal{A}, \mathcal{P})$; the rest follows from $\ell^1_{sl}(q) \geq \ell^2_{sl}(q) \geq \ell^3_{sl}(q)$ proved above.
  Consider some set of states $V \subseteq Q[\mathcal{A}]$ and the set $V' = \min(\lfloor V \rfloor_{sl})$. We can estimate the symmetric difference of the languages of the original and the reduced automaton as

$$\begin{aligned} &L(\mathcal{A}) \triangle L(reduce_{sl}(\mathcal{A}, V)) \\ &\quad = L(\mathcal{A}) \triangle L(reduce_{sl}(\mathcal{A}, V')) && \text{⟅def. of } \sqsubseteq_{sl}\text{⟆} \\ &\quad = L(reduce_{sl}(\mathcal{A}, V'))\backslash L(\mathcal{A}) \\ &\qquad \text{⟅}L(\mathcal{A}) \subseteq L(reduce_{sl}(\mathcal{A}, V'))\text{⟆} \\ &\quad \subseteq \bigcup_{q\in V'} L^{\flat}_{\mathcal{A}}(q).\Sigma^* \backslash \bigcup_{q\in V'} L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q). \\ &\qquad \text{⟅def. of } reduce_{sl}\text{⟆} \end{aligned} \tag{7}$$

  The last inclusion holds because $sl(\mathcal{A}, V)$ adds self-loops to the states in V, so the newly accepted words are for sure those that traverse through V, and they are for sure not those that could be accepted by going through V before the reduction (but they could be accepted without touching V, hence the inclusion). We can estimate the probabilistic distance of $\mathcal{A}$ and $reduce_{sl}(\mathcal{A}, V)$ as

$$\begin{aligned} &d_{\mathcal{P}}(\mathcal{A}, reduce_{sl}(\mathcal{A}, V)) \\ &\leq \Pr_{\mathcal{P}}\left(\bigcup_{q\in V'} L^{\flat}_{\mathcal{A}}(q).\Sigma^* \backslash \bigcup_{q\in V'} L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q)\right) && \text{⟅(7)⟆} \\ &\leq \Pr_{\mathcal{P}}\left(\bigcup_{q\in V'} \left(L^{\flat}_{\mathcal{A}}(q).\Sigma^* \backslash L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q)\right)\right) \\ &\quad \text{⟅properties of union and set difference⟆} \\ &\leq \sum_{q\in V'} \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).\Sigma^* \backslash L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q)\right) \\ &\quad \text{⟅union bound⟆} \\ &= \sum_{q\in V'} \left(\Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).\Sigma^*\right) - \Pr_{\mathcal{P}}\left(L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q)\right)\right) \\ &\quad \text{⟅prop. of Pr and the fact that } L^{\flat}_{\mathcal{A}}(q).L_{\mathcal{A}}(q) \subseteq L^{\flat}_{\mathcal{A}}(q).\Sigma^*\text{⟆} \\ &= \sum\{\ell^3_{sl}(q) \mid q \in \min(\lfloor V \rfloor_{sl})\} \\ &\quad \text{⟅def. of } \ell^3_{sl} \text{ and } V'\text{⟆} \\ &= error_{sl}(\mathcal{A}, V, \ell^3_{sl}). \end{aligned} \tag{8}$$

- **C1**(b): $|reduce_{sl}(\mathcal{A}, Q[\mathcal{A}])| \leq 1$ because, from the definition, $|reduce_{sl}(\mathcal{A}, Q[\mathcal{A}])| = |trim(sl(\mathcal{A}, Q[\mathcal{A}]))| \leq 1$.
- **C1**(c): $reduce_{sl}(\mathcal{A}, \emptyset) = \mathcal{A}$ since

  $$reduce_{sl}(\mathcal{A}, \emptyset) = trim(sl(\mathcal{A}, \emptyset)) = \mathcal{A}.$$

  (We assume that $\mathcal{A}$ is trimmed at the input.) □

5 Reduction of NFAs in network intrusion detection systems

We have implemented our approach in a Python prototype named APPREAL (APProximate REduction of Automata and Languages)[6] and evaluated it on the use case of network intrusion detection using SNORT [50], a popular open-source NIDS. The version of APPREAL used for the evaluation in the current paper is available as an artefact [11] for the TACAS'18 artefact virtual machine [22].

5.1 Network traffic model

The reduction we describe in this paper is driven by a probabilistic model representing a distribution over the words from Σ^*, and the formal guarantees are also wrt this model. We use *learning* to obtain a model of network traffic over the 8-bit ASCII alphabet at a given network point. Our model is created from several gigabytes of network traffic from a measuring point of the CESNET Internet provider connected to a 100 Gbps backbone link. (Unfortunately, we cannot provide the traffic dump since it may contain sensitive data.)

Learning a PA representing the network traffic faithfully is hard. The PA cannot be too specific—although the number of different packets that can occur is finite, it is still extremely large. (A conservative estimate assuming the most common scenario Ethernet/IPv4/TCP would still yield a number over $2^{10,000}$.) If we assigned nonzero probabilities only to the packets from the dump (which are less than 2^{20}), the obtained model would completely ignore virtually all packets that might appear on the network, and, moreover, the model would also be very large (millions of states), making it difficult to use in our algorithms. A generalization of the obtained traffic is therefore needed.

A natural solution is to exploit results from the area of PA learning, such as [10,51]. Indeed, we experimented with

[6] https://github.com/vhavlena/appreal/tree/tacas18

the use of ALERGIA [10], a learning algorithm that constructs a PA from a prefix tree (where edges are labelled with multiplicities) by merging nodes that are "similar." The automata that we obtained were, however, *too* general. In particular, the constructed automata destroyed the structure of network protocols—the merging was too permissive and the generalization merged distant states, which introduced loops over a very large substructure in the automaton. (Such a case usually does not correspond to the design of network protocols.) As a result, the obtained PA more or less represented the Poisson distribution, having essentially no value for us.

In Sect. 5.2, we focus on the detection of malicious traffic transmitted over HTTP. We take advantage of this fact and create a PA representing the traffic while taking into account the structure of HTTP. We start by manually creating a DFA that represents the high-level structure of HTTP. Then, we proceed by feeding 34,191 HTTP packets from our sample into the DFA, at the same time taking notes about how many times every state is reached and how many times every transition is taken. The resulting PA $\mathcal{P}_{HTTP}$ (of 52 states) is then constructed from the DFA and the labels in the obvious way.

The described method yields automata that are much better than those obtained using ALERGIA in our experiments. A disadvantage of the method is that it is only semi-automatic—the basic DFA needed to be provided by an expert. We have yet to find an algorithm that would suit our needs for learning more general network traffic.

5.2 Evaluation

We start this section by introducing the experimental setting, namely, the integration of our reduction techniques into the tool chain implementing efficient regex matching, the concrete settings of APPREAL, and the evaluation environment. Afterwards, we discuss the results evaluating the quality of the obtained approximate reductions as well as of the provided error bounds. Finally, we present the performance of our approach and discuss its key aspects. We selected the most interesting results demonstrating the potential as well as the limitations of our approach.

General setting. SNORT detects malicious network traffic based on *rules* that contain *conditions*. The conditions take into consideration, among others, network addresses, ports, or Perl compatible regular expressions (PCREs) that the packet payload should match. In our evaluation, we select a subset of SNORT rules, extract the PCREs from them, and use NETBENCH [45] to transform them into a single NFA $\mathcal{A}$. Before applying APPREAL, we use the state-of-the-art NFA reduction tool REDUCE [38] to reduce $\mathcal{A}$. REDUCE performs a language-preserving reduction of $\mathcal{A}$ using advanced variants of simulation [37]. (In the experiment reported in Table 3, we skip the use of REDUCE at this step as discussed later in the performance evaluation.) The automaton $\mathcal{A}^{\text{RED}}$ obtained as the result of REDUCE is the input of APPREAL, which performs one of the approximate reductions from Sect. 4 wrt the traffic model $\mathcal{P}_{HTTP}$, yielding $\mathcal{A}^{\text{APP}}$. After the approximate reduction, we, one more time, use REDUCE and obtain the result $\mathcal{A}'$.

Settings of APPREAL In the use case of NIDS pre-filtering, it may be important to never introduce a false negative, i.e. to never drop a malicious packet. Therefore, we focus our evaluation on the *self-loop reduction* (Sect. 4.4). In particular, we use the state labelling function $label^2_{sl}$, since it provides a good trade-off between the precision and the computational demands. (Recall that the computation of $label^2_{sl}$ can exploit the "tentacle" structure of the NFAs we work with.) We give more attention to the *size-driven reduction* (Sect. 4.1) since, in our setting, a bound on the available FPGA resources is typically given and the task is to create an NFA with the smallest error that fits inside. The order $\preceq_{\mathcal{A},\ell^2_{sl}}$ over states used in Sect. 4.1 and Sect. 4.2 is defined as $s \preceq_{\mathcal{A},\ell^2_{sl}} s' \Leftrightarrow \ell^2_{sl}(s) \leq \ell^2_{sl}(s')$.

Evaluation environment All experiments ran on a 64-bit LINUX DEBIAN workstation with the Intel Core(TM) i5-661 CPU running at 3.33 GHz with 16 GiB of RAM.

Description of tables In the caption of every table, we provide the name of the input file (in the directory `regexps/tacas18/` of the repository of APPREAL) with the selection of SNORT regexes used in the particular experiment, together with the type of the reduction (size- or error-driven). All reductions are over-approximating (self-loop reduction). We further provide the size of the input automaton $|\mathcal{A}|$, the size after the initial processing by REDUCE ($|\mathcal{A}^{\text{RED}}|$), and the time of this reduction (*time*(REDUCE)). Finally, we list the times of computing the state labelling $label^2_{sl}$ on $\mathcal{A}^{\text{RED}}$ (*time*($label^2_{sl}$)), the exact probabilistic distance (*time*(Exact)), and also the number of *look-up tables* (*LUTs*($\mathcal{A}^{\text{RED}}$)) consumed on the targeted FPGA (Xilinx Virtex 7 H580T) when $\mathcal{A}^{\text{RED}}$ was synthesized (more on this in Sect. 5.3). The meaning of the columns in the tables is the following:

$\boldsymbol{k}/\boldsymbol{\epsilon}$	is the parameter of the reduction. In particular, $\boldsymbol{k}$ is used for the size-driven reduction and denotes the desired reduction ratio $k = \frac{n}{\lvert\mathcal{A}^{\text{RED}}\rvert}$ for an input NFA $\mathcal{A}^{\text{RED}}$ and the desired size of the output n. On the other hand, $\boldsymbol{\epsilon}$ is the desired maximum error on the output for the error-driven reduction.
$\lvert\boldsymbol{\mathcal{A}}^{\text{APP}}\rvert$	shows the number of states of the automaton $\mathcal{A}^{\text{APP}}$ after the reduction by APPREAL and the time the reduction took. (We omit it when it is not interesting.)

Table 1 Results for the `http-malicious` regex, $|\mathcal{A}_{\mathtt{mal}}| = 249$, $|\mathcal{A}^{\mathrm{RED}}_{\mathtt{mal}}| = 98$, *time*(REDUCE) = 3.5 s, $time(label^2_q) = 38.7$ s, *time*(Exact) = 3.8–6.5 s, and $LUTs(\mathcal{A}^{\mathrm{RED}}_{\mathtt{mal}}) = 382$

k	$\lvert\mathcal{A}^{\mathrm{APP}}_{\mathtt{mal}}\rvert$	$\lvert\mathcal{A}'_{\mathtt{mal}}\rvert$	Error bound	Exact error	Traffic error	LUTs
(a) Size-driven reduction						
0.1	9 (0.65 s)	9 (0.4 s)	0.0704	0.0704	0.0685	–
0.2	19 (0.66 s)	19 (0.5 s)	0.0677	0.0677	0.0648	–
0.3	29 (0.69 s)	26 (0.9 s)	0.0279	0.0278	0.0598	154
0.4	39 (0.68 s)	36 (1.1 s)	0.0032	0.0032	0.0008	–
0.5	49 (0.68 s)	44 (1.4 s)	2.8e−05	2.8e−05	4.1e−06	–
0.6	58 (0.69 s)	49 (1.7 s)	8.7e−08	8.7e−08	0.0	224
0.8	78 (0.69 s)	75 (2.7 s)	2.4e−17	2.4e−17	0.0	297

ϵ	$\lvert\mathcal{A}^{\mathrm{APP}}_{\mathtt{mal}}\rvert$	$\lvert\mathcal{A}'_{\mathtt{mal}}\rvert$	Error bound	Exact error	Traffic error
(b) Error-driven reduction					
0.08	3	3	0.0724	0.0724	0.0720
0.07	4	4	0.0700	0.0700	0.0683
0.04	35	32	0.0267	0.0212	0.0036
0.02	36	33	0.0105	0.0096	0.0032
0.001	41	38	0.0005	0.0005	0.0003
1e−04	47	41	7.7e−05	7.7e−05	1.2e−05
1e−05	51	47	6.6e−06	6.6e−06	0.0

$|\mathcal{A}'|$ contains the number of states of the NFA $\mathcal{A}'$ obtained after applying REDUCE on $\mathcal{A}^{\mathrm{APP}}$ and the time used by REDUCE at this step (omitted when not interesting).

Error bound shows the estimation of the error of $\mathcal{A}'$ as determined by the reduction itself, i.e. it is the probabilistic distance computed by the corresponding function *error* from Sect. 4.

Exact error contains the values of $d_{\mathcal{P}_{HTTP}}(\mathcal{A}, \mathcal{A}')$ that we computed *after* the reduction in order to evaluate the precision of the result given in **Error bound**. The computation of this value is very expensive ($time(Exact)$) since it inherently requires determinization of the whole automaton $\mathcal{A}$. We do not provide it in Table 3 (presenting the results for the automaton $\mathcal{A}_{\mathtt{bd}}$ with 1,352 states) because the determinization ran out of memory. (The step is not required in the reduction process.)

Traffic error shows the error that we obtained when compared $\mathcal{A}'$ with $\mathcal{A}$ on an HTTP traffic sample, in particular the ratio of packets misclassified by $\mathcal{A}'$ to the total number of packets in the sample (242,468). Comparing **Exact error** with **Traffic error** gives us a feedback about the fidelity of the traffic model $\mathcal{P}_{HTTP}$. We note that there are no guarantees on the relationship between **Exact error** and **Traffic error**.

LUTs is the number of LUTs consumed by $\mathcal{A}'$ when synthesized into the target FPGA. Hardware synthesis is a costly step, therefore we provide this value only for selected interesting NFAs.

5.2.1 Approximation errors

Table 1 presents the results of the self-loop reduction for the NFA $\mathcal{A}_{\mathtt{mal}}$ describing regexes from `http-malicious`. We can observe that the differences between the upper bounds on the probabilistic distance and its real value are negligible (typically in the order of 10^{-4} or less). We can also see that the probabilistic distance agrees with the traffic error. This indicates a good quality of the traffic model employed in the reduction process. Further, we can see that our approach can provide useful trade-offs between the reduction error and the reduction factor. Finally, Table 1b shows that a significant reduction is obtained when the error threshold ϵ is increased from 0.04 to 0.07.

Table 2 presents the results of the size-driven self-loop reduction for NFA $\mathcal{A}_{\mathtt{att}}$ describing `http-attacks` regexes. We can observe that the error bounds provide again a very good approximation of the real probabilistic distance.

Table 2 Results for the `http-attacks` regex, size-driven reduction, $|\mathcal{A}_{\texttt{att}}| = 142$, $|\mathcal{A}^{\text{RED}}_{\texttt{att}}| = 112$, *time*(REDUCE) = 7.9 s, $time(label^2_{sl})$ = 28.3 min, *time*(Exact) = 14.0–16.4 min

k	$\lvert\mathcal{A}^{\text{APP}}_{\texttt{att}}\rvert$	$\lvert\mathcal{A}'_{\texttt{att}}\rvert$	Error bound	Exact error	Traffic error
0.1	11 (1.1s)	5 (0.4s)	1.0	0.9972	0.9957
0.2	22 (1.1s)	14 (0.6s)	1.0	0.8341	0.2313
0.3	33 (1.1s)	24 (0.7s)	0.081	0.0770	0.0067
0.4	44 (1.1s)	37 (1.6s)	0.0005	0.0005	0.0010
0.5	56 (1.1s)	49 (1.2s)	3.3e−06	3.3e−06	0.0010
0.6	67 (1.1s)	61 (1.9s)	1.2e−09	1.2e−09	8.7e−05
0.7	78 (1.1s)	72 (2.4s)	4.8e−12	4.8e−12	1.2e−05
0.9	100 (1.1s)	93 (4.7s)	3.7e−16	1.1e−15	0.0

Table 3 Results for `http-backdoor`, size-driven reduction, $|\mathcal{A}_{\texttt{bd}}| = 1,352$, $time(label^2_{sl})$ = 19.9 min, $LUTs(\mathcal{A}^{\text{RED}}_{\texttt{bd}}) = 2,266$

k	$\lvert\mathcal{A}^{\text{APP}}_{\texttt{bd}}\rvert$	$\lvert\mathcal{A}'_{\texttt{bd}}\rvert$	Error bound	Traffic error	LUTs
0.1	135 (1.2 m)	8 (2.6 s)	1.0	0.997	202
0.2	270 (1.2 m)	111 (5.2 s)	0.0012	0.0631	579
0.3	405 (1.2 m)	233 (9.8 s)	3.4e−08	0.0003	894
0.4	540 (1.3 m)	351 (21.7 s)	1.0e−12	0.0003	1,063
0.5	676 (1.3 m)	473 (41.8 s)	1.2e−17	0.0	1,249
0.7	946 (1.4 m)	739 (2.1 m)	8.3e−30	0.0	1,735
0.9	1216 (1.5 m)	983 (5.6 m)	1.3e−52	0.0	2,033

On the other hand, the difference between the probabilistic distance and the traffic error is larger than that for $\mathcal{A}_{\texttt{mal}}$. Since all experiments use the same probabilistic automaton and the same traffic, this discrepancy is accounted to the different set of packets that are incorrectly accepted by $\mathcal{A}^{\text{RED}}_{\texttt{att}}$. If the probability of these packets is adequately captured in the traffic model, the difference between the distance and the traffic error is small and vice versa. This also explains an even larger difference in Table 3 (presenting the results for $\mathcal{A}_{\texttt{bd}}$ constructed from `http-backdoor` regexes) for $k \in \langle 0.2, 0.4\rangle$. Here, the traffic error is very small and caused by a small set of packets (approx. 70), whose probability is not correctly captured in the traffic model. Despite this problem, the results clearly show that our approach still provides significant reductions while keeping the traffic error small: about a fivefold reduction is obtained for the traffic error 0.03 % and a tenfold reduction is obtained for the traffic error 6.3 %. We discuss the practical impact of such a reduction in Sect. 5.3.

5.2.2 Performance of the approximate reduction

In all our experiments (Tables 1, 2, 3), we can observe that the most time-consuming step of the reduction process is the computation of state labellings. (It takes at least 90 % of the total time.) The crucial observation is that the structure of the NFAs fundamentally affects the performance of this step. Although after REDUCE, the size of $\mathcal{A}_{\texttt{mal}}$ is very similar to the size of $\mathcal{A}_{\texttt{att}}$, computing $label^2_{sl}$ takes more time (28.3 min vs. 38.7 s). The key reason behind this slowdown is the determinization (or alternatively disambiguation) process required by the product construction underlying the state labelling computation (cf. Sect. 4.4). For $\mathcal{A}_{\texttt{att}}$, the process results in a significantly larger product when compared to the product for $\mathcal{A}_{\texttt{mal}}$. The size of the product directly determines the time and space complexity of solving the linear equation system required for computing the state labelling.

As explained in Sect. 4, the computation of the state labelling $label^2_{sl}$ can exploit the "tentacle" structure of the NFAs appearing in NIDSes and thus can be done component-wise. On the other hand, our experiments reveal that the use of REDUCE typically breaks this structure and thus the component-wise computation cannot be effectively used. For the NFA $\mathcal{A}_{\texttt{mal}}$, this behaviour does not have any major performance impact as the determinization leads to a moderate-sized automaton and the state labelling computation takes less than 40 s. On the other hand, this behaviour has a dramatic effect for the NFA $\mathcal{A}_{\texttt{att}}$. By disabling the initial application of REDUCE and thus preserving the original structure of $\mathcal{A}_{\texttt{att}}$, we were able to speed up the state label computation from 28.3 to 1.5 min. Note that other steps of the approximate reduction took a similar time as before disabling REDUCE and also that the trade-offs between the error and the reduction factor were similar. Surprisingly, disabling REDUCE caused that the computation of the exact probabilistic distance became computationally infeasible because the determinization ran out of memory.

Due to the size of the NFA $\mathcal{A}_{\texttt{bd}}$, the impact of disabling the initial application of REDUCE is even more fundamental. In particular, computing the state labelling took only 19.9 min, in contrast to running out of memory when the REDUCE is applied in the first step. (Therefore, the input automaton is not processed by REDUCE in Table 3; we still give the number of LUTs of its reduced version for comparison, though.) Note that the size of $\mathcal{A}_{\texttt{bd}}$ also slows down other reduction steps (the greedy algorithm and the final REDUCE reduction). We can, however, clearly see that computing the state labelling is still the most time-consuming step of the process.

5.3 The real impact in an FPGA-accelerated NIDS

To demonstrate the practical usefulness and impact of the proposed approximation techniques, we employ the reduced automata in a real use case from the area of HW-accelerated

deep packet inspection. We consider the framework of [36] implementing a high-speed NIDS pre-filter in an FPGA. The crucial challenge is to obtain a pre-filter with a sufficiently small false positive rate (and no false negatives), while being able to handle the traffic of current networks operating on 100 Gbps and beyond. The implementation of NFAs performing regex matching in FPGAs uses two types of HW resources: LUTs, which are used to build the combinational circuit representing the NFA transition function, and flip-flops, representing NFA states. In our use case, we omit the analysis of flip-flop consumption because it is always dominated by the LUT consumption.

In our setting, the amount of resources available for the FPGA-based regex matching engine is 15,000 LUTs and the frequency of the engine is 200 MHz using a 32-bit-wide data path. As explained in [36], the engine containing a single unit (i.e. the single NFA implementation) can achieve the throughput of 6.4 Gbps (200 MHz $\times$ 32 b). Therefore, 16 units are required for the desired link speed of 100 Gbps and 63 units are needed to handle 400 Gbps. With the given amount of LUTs, the size of a single NFA is thus bounded by 937 LUTs (15,000/16) for 100 Gbps and 238 LUTs for 400 Gbps, respectively. These bounds directly limit the complexity of regexes the engine can handle.

We now analyse the resource consumption of the matching engine for two automata, `http-backdoor` ($\mathcal{A}^{\text{RED}}_{\texttt{bd}}$) and `http-malicious` ($\mathcal{A}^{\text{RED}}_{\texttt{mal}}$), and evaluate the impact of the reduction techniques. Recall that the automata represent two important sets of know network attacks from SNORT [50].

- **100 Gbps**: For this speed, $\mathcal{A}^{\text{RED}}_{\texttt{mal}}$ can be used without any approximate reduction as it is small enough (it has 382 LUTs) to fit in the available space. On the other hand, $\mathcal{A}^{\text{RED}}_{\texttt{bd}}$ without the approximate reduction is way too large to fit. (It has 2,266 LUTs and thus at most 6 units fit inside the available space, yielding the throughput of only 38.4 Gbps, which is unacceptable.) The column **LUTs** in Table 3 shows that using our framework, we are able to reduce $\mathcal{A}^{\text{RED}}_{\texttt{bd}}$ such that it uses 894 LUTs (for $\boldsymbol{k} = 0.3$), and so all of the 16 needed units fit into the FPGA, yielding the throughput over 100 Gbps and the theoretical error bound of a false positive $\leq 3.4 \times 10^{-8}$ wrt the network traffic model $\mathcal{P}_{HTTP}$.
- **400 Gbps**: Regex matching at this speed is extremely challenging. In the case of $\mathcal{A}^{\text{RED}}_{\texttt{bd}}$, the reduction $\boldsymbol{k} = 0.1$ is required to fit 63 units in the available space. As such a reduction has error bound almost 1, this solution is not useful due to a prohibitively high false positive rate. The situation is better for $\mathcal{A}^{\text{RED}}_{\texttt{mal}}$. In the exact version, at most 39 units can fit inside the FPGA with the maximum throughput of 249.6 Gbps. On the other hand, when using our reduced automata, we are able to place 63 units into the FPGA, each of the size 224 LUTs ($\boldsymbol{k} = 0.6$), and achieve a throughput of over 400 Gbps with the theoretical error bound of a false positive $\leq 8.7 \times 10^{-8}$ wrt the model $\mathcal{P}_{HTTP}$.

6 Conclusion

We have proposed a novel approach for approximate reduction of NFAs used in network traffic filtering. Our approach is based on a proposal of a probabilistic distance of the original and reduced automaton using a probabilistic model of the input network traffic, which characterizes the significance of particular packets. We characterized the computational complexity of approximate reductions based on the described distance and proposed a sequence of heuristics allowing one to perform the approximate reduction in an efficient way. Our experimental results are quite encouraging and show that we can often achieve a very significant reduction for a negligible loss of precision. We showed that using our approach, FPGA-accelerated network filtering on large traffic speeds can be applied on regexes of malicious traffic where it could not be applied before.

In the future, we plan to investigate other approximate reductions of the NFAs, maybe using some variant of abstraction from abstract regular model checking [7], adapted for the given probabilistic setting. Another important issue for the future is to develop better ways of learning a suitable probabilistic model of the input traffic.

Acknowledgements We thank Jan Kořenek, Vlastimil Košař, and Denis Matoušek for their help with translating regexes into automata and synthesis of FPGA designs, and Martin Žádník for providing us with the backbone network traffic. We thank Stefan Kiefer for helping us proving the **PSPACE** part of Lemma 1 and Petr Peringer for testing our artefact. We also thank the anonymous reviewers for their useful comments, which improved the quality of the paper. The work on this paper was supported by the Czech Science Foundation project 16-24707Y, the IT4IXS: IT4Innovations Excellence in Science project (LQ1602), and the FIT BUT internal project FIT-S-17-4014.

References

1. Angluin, D.: Learning regular sets from queries and counterexamples. Inf. Comput. **75**(2), 87–106 (1987). https://doi.org/10.1016/0890-5401(87)90052-6
2. Baier, C., Kiefer, S., Klein, J., Klüppelholz, S., Müller, D., Worrell, J.: Markov chains and unambiguous Büchi automata. In: CAV'16, pp. 23–42. Springer (2016)
3. Becchi, M., Crowley, P.: A hybrid finite automaton for practical deep packet inspection. In: CoNEXT'07, p. 1. ACM (2007)
4. Becchi, M., Crowley, P.: An improved algorithm to accelerate regular expression evaluation. In: ANCS'07, pp. 145–154. ACM (2007)
5. Becchi, M., Wiseman, C., Crowley, P.: Evaluating regular expression matching engines on network and general purpose processors. In: Proceedings of the 5th ACM/IEEE Symposium on Architec-

tures for Networking and Communications Systems, ANCS '09, pp. 30–39. ACM (2009)
6. Benedikt, M., Lenhardt, R., Worrell, J.: Model checking Markov chains against unambiguous Büchi automata. CoRR arXiv:1405.4560v2 (2016)
7. Bouajjani, A., Habermehl, P., Rogalewicz, A., Vojnar, T.: Abstract regular (tree) model checking. STTT **14**(2), 167–191 (2012)
8. Brodie, B.C., Taylor, D.E., Cytron, R.K.: A scalable architecture for high-throughput regular-expression pattern matching. In: ISCA'06, pp. 191–202. IEEE Computer Society (2006)
9. Bustan, D., Grumberg, O.: Simulation-based minimization. ACM Trans. Comput. Log. **4**(2), 181–206 (2003)
10. Carrasco, R.C., Oncina, J.: Learning stochastic regular grammars by means of a state merging method. In: Proceedings of the Second International Colloquium on Grammatical Inference and Applications, ICGI '94, pp. 139–152. Springer (1994)
11. Češka, M., Havlena, V., Holík, L., Lengál, O., Vojnar, T.: Approximate reduction of finite automata for high-speed network intrusion detection. In: Figshare (2018). https://doi.org/10.6084/m9.figshare.5907055
12. Češka, M., Havlena, V., Holík, L., Lengál, O., Vojnar, T.: Approximate reduction of finite automata for high-speed network intrusion detection. In: Proceedings of TACAS'18, *LNCS*, vol. 10806. Springer (2018)
13. Champarnaud, J., Coulon, F.: NFA reduction algorithms by means of regular inequalities. Theor. Comput. Sci. **327**(3), 241–253 (2004)
14. Clark, C.R., Schimmel, D.E.: Efficient reconfigurable logic circuits for matching complex network intrusion detection patterns. In: FPL'03, Lecture Notes in Computer Science, vol. 2778, pp. 956–959. Springer (2003)
15. Clemente, L.: Büchi automata can have smaller quotients. In: ICALP'11, Lecture Notes in Computer Science, vol. 6756, pp. 258–270. Springer (2011)
16. Csanky, L.: Fast parallel matrix inversion algorithms. In: 16th Annual Symposium on Foundations of Computer Science, pp. 11–12 (1975). https://doi.org/10.1109/SFCS.1975.14
17. Deza, M.M., Deza, E.: Encyclopedia of Distances. Springer, Berlin (2009)
18. Etessami, K.: A hierarchy of polynomial-time computable simulations for automata. In: CONCUR 2002—Concurrency Theory, 13th International Conference, Brno, Czech Republic, August 20-23, 2002, Proceedings, Lecture Notes in Computer Science, vol. 2421, pp. 131–144. Springer (2002)
19. Fortune, S., Wyllie, J.: Parallelism in random access machines. In: Proceedings of the Tenth Annual ACM Symposium on Theory of Computing, STOC '78, pp. 114–118. ACM, New York, NY, USA (1978). https://doi.org/10.1145/800133.804339
20. Gange, G., Ganty, P., Stuckey, P.J.: Fixing the state budget: approximation of regular languages with small DFAs. In: ATVA'17, Lecture Notes in Computer Science, vol. 10482, pp. 67–83. Springer (2017)
21. Gawrychowski, P., Jez, A.: Hyper-minimisation made efficient. In: MFCS'09, Lecture Notes in Computer Science, vol. 5734, pp. 356–368. Springer (2009)
22. Hartmanns, A., Wendler, P.: TACAS 2018 artifact evaluation VM. In: Figshare (2018). https://doi.org/10.6084/m9.figshare.5896615
23. Hogben, L.: Handbook of Linear Algebra, 2nd edn. CRC Press, Boca Raton (2013)
24. Hopcroft, J.E.: An N log N algorithm for minimizing states in a finite automaton. Technical report (1971)
25. Hutchings, B.L., Franklin, R., Carver, D.: Assisting network intrusion detection with reconfigurable hardware. In: FCCM'02, pp. 111–120. IEEE Computer Society (2002)
26. Jiang, T., Ravikumar, B.: Minimal NFA problems are hard. SIAM J. Comput. **22**(6), 1117–1141 (1993)
27. Kaštil, J., Kořenek, J., Lengál, O.: Methodology for fast pattern matching by deterministic finite automaton with perfect hashing. In: 2009 12th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, pp. 823–829 (2009)
28. Kořenek, J., Kobierský, P.: Intrusion detection system intended for multigigabit networks. In: 2007 IEEE Design and Diagnostics of Electronic Circuits and Systems, pp. 1–4 (2007)
29. Kumar, S., Chandrasekaran, B., Turner, J.S., Varghese, G.: Curing regular expressions matching algorithms from insomnia, amnesia, and acalculia. In: ANCS'07, pp. 155–164. ACM (2007)
30. Kumar, S., Dharmapurikar, S., Yu, F., Crowley, P., Turner, J.S.: Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In: SIGCOMM'06, pp. 339–350. ACM (2006)
31. Kumar, S., Turner, J.S., Williams, J.: Advanced algorithms for fast and scalable deep packet inspection. In: ANCS'06, pp. 81–92. ACM (2006)
32. Liu, C., Wu, J.: Fast deep packet inspection with a dual finite automata. IEEE Trans. Comput. **62**(2), 310–321 (2013)
33. Luchaup, D., De Carli, L., Jha, S., Bach, E.: Deep packet inspection with DFA-trees and parametrized language overapproximation. In: INFOCOM'14, pp. 531–539. IEEE (2014)
34. Malcher, A.: Minimizing finite automata is computationally hard. Theor. Comput. Sci. **327**(3), 375–390 (2004)
35. Maletti, A., Quernheim, D.: Optimal hyper-minimization. CoRR arXiv:1104.3007 (2011)
36. Matoušek, D., Kořenek, J., Puš, V.: High-speed regular expression matching with pipelined automata. In: 2016 International Conference on Field-Programmable Technology (FPT), pp. 93–100 (2016)
37. Mayr, R., Clemente, L.: Advanced automata minimization. In: POPL'13, Transactions on Computer Logic, pp. 63–74. ACM (2013)
38. Mayr, R., et al.: Reduce: A tool for minimizing nondeterministic finite-word and Büchi automata. http://languageinclusion.org/doku.php?id=tools (2017). Accessed 30 Sept 2017
39. Mitra, A., Najjar, W.A., Bhuyan, L.N.: Compiling PCRE to FPGA for accelerating SNORT IDS. In: ANCS'07, pp. 127–136. ACM (2007)
40. Mohri, M.: A disambiguation algorithm for finite automata and functional transducers. In: CIAA'12, pp. 265–277. Springer (2012)
41. Mohri, M.: Edit-distance of weighted automata. In: CIAA'02, Lecture Notes in Computer Science, vol. 2608, pp. 1–23. Springer (2002)
42. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM J. Comput. **16**(6), 973–989 (1987)
43. Papadimitriou, C.M.: Computational Complexity. Addison-Wesley, Reading (1994)
44. Parker, A.J., Yancey, K.B., Yancey, M.P.: Regular language distance and entropy. CoRR arXiv:1602.07715 (2016)
45. Puš, V., Tobola, J., Košař, V., Kaštil, J., Kořenek, J.: Netbench: framework for evaluation of packet processing algorithms. In: Symposium On Architecture For Networking And Communications Systems pp. 95–96 (2011)
46. Shützenberger, M.: On the definition of a family of automata. Inf. Control **4**, 245–270 (1961)
47. Sidhu, R.P.S., Prasanna, V.K.: Fast regular expression matching using FPGAs. In: FCCM'01, pp. 227–238. IEEE Computer Society (2001)
48. Solodovnikov, V.I.: Upper bounds on the complexity of solving systems of linear equations. J. Sov. Math. **29**(4), 1482–1501 (1985)
49. Tan, L., Sherwood, T.: A high throughput string matching architecture for intrusion detection and prevention. In: ISCA'05, pp. 112–122. IEEE Computer Society (2005)
50. The Snort Team: Snort. http://www.snort.org. Accessed 30 Sept 2017

51. Thollard, F., Clark, A.: Learning stochastic deterministic regular languages. In: G. Paliouras, Y. Sakakibara (eds.) Grammatical Inference: Algorithms and Applications: 7th International Colloquium, ICGI 2004, Athens, Greece, October 11–13, 2004. Proceedings, pp. 248–259. Springer Berlin Heidelberg, Berlin, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30195-0_22
52. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite state programs. In: SFCS '85, pp. 327–338. IEEE
53. Yu, F., Chen, Z., Diao, Y., Lakshman, T.V., Katz, R.H.: Fast and memory-efficient regular expression matching for deep packet inspection. In: ANCS'06, pp. 93–102. ACM (2006)

Semi-quantitative Abstraction and Analysis of Chemical Reaction Networks

Milan Češka[1( )] and Jan Křetínský[2]

[1] Brno University of Technology, FIT,
IT4I Centre of Excellence, Brno, Czech Republic
ceskam@fit.vutbr.cz
[2] Technical University of Munich, Munich, Germany

Abstract. Analysis of large continuous-time stochastic systems is a computationally intensive task. In this work we focus on population models arising from chemical reaction networks (CRNs), which play a fundamental role in analysis and design of biochemical systems. Many relevant CRNs are particularly challenging for existing techniques due to complex dynamics including stochasticity, stiffness or multimodal population distributions. We propose a novel approach allowing not only to predict, but also to explain both the transient and steady-state behaviour. It focuses on qualitative description of the behaviour and aims at quantitative precision only in orders of magnitude. First we build a compact understandable model, which we then crudely analyse. As demonstrated on complex CRNs from literature, our approach reproduces the known results, but in contrast to the state-of-the-art methods, it runs with virtually no computational cost and thus offers unprecedented scalability.

1 Introduction

Chemical Reaction Networks (CRNs) are a versatile language widely used for *modelling and analysis* of biochemical systems [12] as well as for high-level *programming* of molecular devices [8,40]. They provide a compact formalism equivalent to Petri nets [37], Vector Addition Systems (VAS) [29] and distributed population protocols [3]. Motivated by numerous potential applications ranging from system biology to synthetic biology, various techniques allowing simulation and formal analysis of CRNs have been proposed [2,9,21,24,39], and embodied in the design process of biochemical systems [20,25,32]. The time-evolution of CRNs is governed by the Chemical Master Equation (CME), which describes the probability of the molecular counts of each chemical species. Many important biochemical systems lead to complex dynamics that includes *state space explosion, stochasticity, stiffness, and multimodality* of the population distributions

This work has been supported by the Czech Science Foundation grant No. GA19-24397S, the IT4Innovations excellence in science project No. LQ1602, and the German Research Foundation (DFG) project KR 4890/2-1 "Statistical Unbounded Verification".

I. Dillig and S. Tasiran (Eds.): CAV 2019, LNCS 11561, pp. 475–496, 2019.
https://doi.org/10.1007/978-3-030-25540-4_28

[23,44], and that fundamentally limits the class of systems the existing techniques can effectively handle. More importantly, biologist and engineers often seek for plausible explanations why the system under study has or has not the required behaviour. In many cases, a set of system simulations/trajectories or population distributions is not sufficient and the ability to provide an accurate explanation for the temporal or steady-state behaviour is another major challenge for the existing techniques.

In order to cope with the computational complexity of the analysis and in order to obtain explanations of the behaviour, we shift the focus from quantitatively precise results to a more qualitative analysis, closer to how a human would behold the system. Yet we insist on providing at least rough timing information on the behaviour as well as rough classification of probability of different behaviours at the extent of "very likely", "few percent", "barely possible", so that we can conclude on issues such as time to extinction or bimodality of behaviour. This gives rise to our *semi-quantitative* approach. We stipulate that analyses in this framework reflect quantities in orders of magnitude, both for time duration and probabilities, but not more than that. This paradigm shift is reflected on two levels: (1) We abstract systems into semi-quantitative models. (2) We analyse systems in a semi-quantitative way. While each of the two can be combined with a traditional abstraction/analysis, when combined together they provide powerful means to understand systems' behaviour with virtually no computational cost.

Semi-quantitative Models. The states of the models contain information on the current amount of objects of each species as an interval spanning often several orders of magnitude, unless instructed otherwise. For instance, if an amount of a certain species is to be closely monitored (as a part of the input specification/property of the system) then this abstraction can be finer. Similarly, whenever the analysis of a previous version of the abstraction points to the lack of precision in certain states, preventing us to conclude which of the possible behaviours is prevalent, the corresponding refinement can take place. Further, the rates of the transitions are also captured only with such imprecision. The crucial point allowing for existence of such models that are small, yet faithful, is our concept of *acceleration*. It captures certain *sequences* of transitions. It eliminates most of the non-determinism that paralyses other types of abstractions, which are too over-approximative, unable to conclude anything, but safety properties.

Semi-quantitative Analysis. Instead of performing exact transient or steady-state analysis, we can consider most probable transitions and then carefully lift this to most probable temporal behaviours. Technically, this is done by *alternating between transient and steady-state analysis* where only some rates and transitions are taken into account at different stages. In order to further facilitate the resulting insight of the human on the result of the analysis, we provide an algorithm to perform this analysis with virtually no computation effort and thus possibly manually. The trivial computations immediately pinpoint why certain

behaviours occur. Moreover, less likely behaviours can also be identified easily, to any desired degree of improbability (dozens of percent, promilles etc.).

To summarise, the first step yields tiny models, allowing for a synoptic observation of the model; due to their size these models can be either analysed easily using standard means, or can be subject to the second step. The second step provides an efficient approximative analysis, which is also very illustrative due to the limited use of quantities. It can be applied to any system; however, it is particularly interesting in connection with the models coming from the first step since (i) no extra effort (size, computation) is wasted on overly precise treatment that is ignored by the other step, and (ii) together they yield an understandable explanation of the behaviour. An entertaining feature of this paradigm is that the stiffer (with rates at hugely different time scales) the system is the easier it is to analyse.

To demonstrate the capabilities of our approach, we consider three challenging and biologically relevant case studies that have been used in literature to evaluate state-of-the-art methods for the CRN analysis. It has been shown that many approaches fail, either due to time-outs or incapability to capture differences in behaviours, and some tailored ones require considerable computational effort, e.g. an hour of computation. Our experiments clearly show that the proposed approach can deliver results that yield qualitatively same information, more understanding and can be computed in minutes by hand (or within a fraction of a second by computer).

Our contribution can be summarized as follows:

- We propose a novel *semi-quantitative* framework for analysis of CRN and similar population models, focusing on explainability of the results and low complexity, with quantitative precision limited to orders of magnitude.
- An algorithm for abstracting CRNs into semi-quantitative models based on interval abstraction of the species population and on transition acceleration.
- An algorithm for semi-quantitative analysis that replaces exact numerical computation by exploring the most probable transitions and alternating transient and steady-state analysis.
- We consider three challenging CRNs thoroughly studied in literature and demonstrate that the semi-quantitative abstraction and analysis gives us a unique tool that is able to accurately predict and explain both transient and steady-state behaviour of complex CRNs in a fraction of a second.

Related Work

To the best of our knowledge, there does not exist any abstraction of CRNs similar to the proposed approach. Indeed, there exist various abstraction and approximation schemes for CRNs that improve the performance and scalability of both the simulation-based and the numerical-based techniques. In the following paragraphs, we discuss the most relevant directions and the links to our approach.

Approximate Semantics for CRNs. For CRNs including large populations of species, fluid (mean-field) approximation techniques can be applied [5] and extended to approximate higher-order moments [15]: these deterministic approximations lead to a set of ordinary differential equations (ODEs). An alternative is to approximate the CME as a continuous-state stochastic process. The Linear Noise Approximation (LNA) is a Gaussian process which has been derived as an approximation of the CME [16,44] and describes the time evolution of expectation and variance of the species in terms of ODEs. Recently, an aggregation scheme over ODEs that aims at understanding the dynamics of large CRNs has been proposed in [10]. In contrast to our approach, the deterministic approximations cannot adequately capture the stochasticity of CRNs caused by low population species.

To mitigate this drawback, various *hybrid models* have been proposed. The common idea of these models is as follows: the dynamics of low population species is described by the discrete stochastic process and the dynamics of large population species is approximated by a continuous process. The particular hybrid models differ in the approximation of the large population species. In [27], a pure deterministic semantics for large population species is used. The moment-based description for medium/high-copy number species was used in [24]. The LNA approximation and an adaptive partitioning of the species according to leap conditions (that is more general than partitioning based on population thresholds) was proposed in [9]. All hybrid models have to deal with interactions between low and large population species. In particular, the dynamics of the stochastic process describing the low-population species is conditioned by the continuous-state describing the concentration of the large-population species. The numerical analysis of such conditioned stochastic process is typically a computationally demanding task that limits the scalability.

In contrast, our approach does not explicitly partition the species, but rather abstracts the concrete species population using an interval abstraction and tries to effectively capture both the stochastic and the deterministic behaviour with the help of the accelerated transitions. As we already emphasised, the proposed abstraction and analysis avoids any numerical computation of precise quantities.

Reduction Techniques for Stochastic Models. A widely studied reduction method for Markov models is state aggregation based on lumping [6] or (bi-)simulation equivalence [4], with the latter notion in its exact [33] or approximate [13] form. Approximate notions of equivalence have led to new abstraction/refinement techniques for the numerical verification of Markov models over finite [14] as well as uncountably-infinite state spaces [1,41,42]. Several approximate aggregation schemes leveraging the structural properties of CRNs were proposed [17,34,45]. Abate et al. proposed an adaptive aggregation that gives formal guarantees on the approximation error, but typically provide lower state space reductions [2]. Our approach shares the idea of abstracting the state space by aggregating some states together. Similarly to [17,34,45], we partition the state space based on the species population, i.e. we also introduce the population levels. In contrast to the aforementioned aggregation schemes, we propose a

novel abstraction of the transition relation based on the acceleration. It allows us to avoid the numerical solution of the approximate CME and thus achieve a better reduction while providing an accurate predication of the system behaviour.

Alternative methods to deal with large/infinite state spaces are based on a state truncation trying to eliminate insignificant states, i.e., states reached only with a negligible probability. These methods, including finite state projections [36], sliding window abstractions [26], or fast adaptive uniformisation [35], are able to quantify the total probability mass that is lost due to the truncation, but typically cannot effectively handle systems involving a stiff behaviour and multimodality [9].

Simulation-Based Analysis. Transient analysis of CRNs can be performed using the Stochastic Simulation Algorithm (SSA) [21]. Note that the SSA produces a single realisation of the stochastic process, whereas the stochastic solution of CME gives the probability distribution of each species over time. Although simulation-based analysis is generally faster than direct solution of the stochastic process underlying the given CRN, obtaining good accuracy necessitates potentially large numbers of simulations and can be very time consuming.

Various partitioning schemes for species and reactions have been proposed for the purpose of speeding up the SSA in multi-scale systems [23,38,39]. For instance, Yao et al. introduced the slow-scale SSA [7], where they distinguish between fast and slow species. Fast species are then treated assuming they reach equilibrium much faster than the slow ones. Adaptive partitioning of the species has been considered in [19,28]. In contrast to the simulation-based analysis, our approach (i) provides a compact explanation of the system behaviour in the form of tiny models allowing for a synoptic observation and (ii) can easily reveal less probable behaviours.

2 Chemical Reaction Networks

In this paper, we assume familiarity with standard verification of (continuous-time) probabilistic systems, e.g. [4]. For more detail, see [11, Appendix].

CRN Syntax. A *chemical reaction network (CRN)* $\mathcal{N} = (\Lambda, \mathcal{R})$ is a pair of finite sets, where Λ is a set of *species*, $|\Lambda|$ denotes its size, and $\mathcal{R}$ is a set of reactions. Species in Λ interact according to the reactions in $\mathcal{R}$. A *reaction* $\tau \in \mathcal{R}$ is a triple $\tau = (r_\tau, p_\tau, k_\tau)$, where $r_\tau \in \mathbb{N}^{|\Lambda|}$ is the *reactant complex*, $p_\tau \in \mathbb{N}^{|\Lambda|}$ is the *product complex* and $k_\tau \in \mathbb{R}_{>0}$ is the coefficient associated with the rate of the reaction. r_τ and p_τ represent the stoichiometry of reactants and products. Given a reaction $\tau_1 = ([1,1,0],[0,0,2],k_1)$, we often refer to it as $\tau_1 : \lambda_1 + \lambda_2 \xrightarrow{k_1} 2\lambda_3$.

CRN Semantics. Under the usual assumption of mass action kinetics, the *stochastic* semantics of a CRN $\mathcal{N}$ is generally given in terms of a discrete-state, continuous-time stochastic process $\mathbf{X}(\mathbf{t}) = (X_1(t), X_2(t), \ldots, X_{|\Lambda|}(t), t \geq 0)$ [16]. The *state change* associated to the reaction τ is defined by $\upsilon_\tau = p_\tau - r_\tau$, i.e. the state $\mathbf{X}$ is changed to $\mathbf{X}' = \mathbf{X} + \upsilon_\tau$, which we denote as $\mathbf{X} \xrightarrow{\tau} \mathbf{X}'$. For example,

for τ_1 as above, we have $\upsilon_{\tau_1} = [-1, -1, 2]$. For a reaction to happen in a state $\mathbf{X}$, all reactants have to be in sufficient numbers. The *reachable state space* of $\mathbf{X}(\mathbf{t})$, denoted as $\mathbf{S}$, is the set of all states reachable by a sequence of reactions from a given *initial state* $\mathbf{X}_0$. The set of reactions changing the state $\mathbf{X}_i$ to the state $\mathbf{X}_j$ is denoted as $\mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j) = \{\tau \mid \mathbf{X}_i \xrightarrow{\tau} \mathbf{X}_j\}$.

The behaviour of the stochastic system $\mathbf{X}(\mathbf{t})$ can be described by the (possibly infinite) continuous-time Markov chain (CTMC) $\gamma(\mathcal{N}) = (\mathbf{S}, \mathbf{X}_0, \mathbf{R})$ where the transition matrix $\mathbf{R}(i, j)$ gives the probability of a transition from $\mathbf{X}_i$ to $\mathbf{X}_j$. Formally,

$$\mathbf{R}(i,j) = \sum_{\tau \in \mathsf{reac}(\mathbf{X}_i, \mathbf{X}_j)} k_\tau \cdot C_{\tau,i} \quad \text{where} \quad C_{\tau,i} = \prod_{\ell=1}^{N} \binom{\mathbf{X}_{i,\ell}}{r_\ell} \tag{R}$$

corresponds to the population dependent term of the *propensity function* where $\mathbf{X}_{i,\ell}$ is ℓth component of the state $\mathbf{X}_i$ and r_ℓ is the stoichiometric coefficient of the ℓ-th reactant in the reaction τ. The CTMC $\gamma(\mathcal{N})$ is the accurate representation of CRN $\mathcal{N}$, but—even when finite—not scalable in practice because of the state space explosion problem [25,31].

3 Semi-quantitative Abstraction

In this section, we describe our abstraction. We derive the desired CTMC conceptually in several steps, which we describe explicitly, although we implement the construction of the final system directly from the initial CRN.

3.1 Over-Approximation by Interval Abstraction and Acceleration

Given a CRN $\mathcal{N} = (\Lambda, \mathcal{R})$, we first consider an interval continuous-time Markov decision process (interval CTMDP[1]), which is a finite abstraction of the infinite $\gamma(\mathcal{N})$. Intuitively, abstract states are given by intervals on sizes of populations with an additional specific that the abstraction captures enabledness of reactions. The transition structure follows the ideas of the standard may abstraction and of the three-valued abstraction of continuous-time systems [30]. A technical difference in the latter point is that we abstract rates into intervals instead of uniformising the chain and then only abstracting transition probabilities into intervals; this is necessary in later stages of the process. The main difference is that we also treat certain sequences of actions, which we call acceleration.

Abstract Domains. The first step is to define the abstract domain for the population sizes. For every species $\lambda \in \Lambda$, we define a finite partitioning A_λ of $\mathbb{N}$ into intervals, reflecting the rough size of the population. Moreover, we want the abstraction to reflect whether a reaction is enabled. Hence we require that

[1] Interval CTMDP is a CTMDP with lower/upper bounds on rates. Since it serves only as an intermediate formalism to ease the presentation, we refrain from formalising it here.

$\{0\} \in A_\lambda$ for the case when the coefficients of this species as a reactant is always 0 or 1; in general, for every $i < \max_{\tau \in \mathcal{R}} r_\tau(\lambda)$ we require $\{i\} \in A_\lambda$.

The abstraction $\alpha_\lambda(n)$ of a number n of a species λ is then the $I \in A_\lambda$ for which $n \in I$. The state space of $\alpha(\mathcal{N})$ is the product $\prod_{\lambda \in \Lambda} A_\lambda$ of the abstract domains with the point-wise defined abstraction $\alpha(\boldsymbol{n})_\lambda = \alpha_\lambda(\boldsymbol{n}_\lambda)$.

The abstract domain for the rates according to (R) is the set of all real intervals.

Transitions from an abstract state are defined as the may abstraction as follows. Since our abstraction reflect enabledness, the same set of action is enabled in all concrete states of a given abstract state. The targets of the action in the abstract setting are abstractions of all possible concrete successors, i.e. $succ(s, a) := \{\alpha(\boldsymbol{n}) \mid \boldsymbol{m} \in s, \boldsymbol{m} \xrightarrow{a} \boldsymbol{n}\}$, in other words, the transitions enabled in at least one of the respective concrete states. The abstract rate is the smallest interval including all the concrete rates of the respective concrete transitions. This can be easily computed by the corner-points abstraction (evaluating only the extremum values for each species) since the stoichiometry of the rates is monotone in the population sizes.

High-Level of Non-determinism. The (more or less) standard style of the abstraction above has several drawbacks—mostly related to the high degree of non-determinism for rates—which we will subsequently discuss.

Firstly, in connection with the abstract population sizes, transitions to different sizes only happen non-deterministically, leaving us unable to determine which behaviour is probable. For example, consider the simple system given by $\lambda \xrightarrow{d} \emptyset$ with $k_d = 10^{-4}$ so the degradation happens on average each 10^4 seconds. Assume population discretisation into $[0], [1..5], [6..20], [21..\infty)$ with abstraction depicted in Fig. 1. While the original system obviously moves from $[6..20]$ to $[1..5]$ very probably in less than $15 \cdot 10^4$ seconds, the abstraction cannot even say that it happens, not to speak of estimating the time.

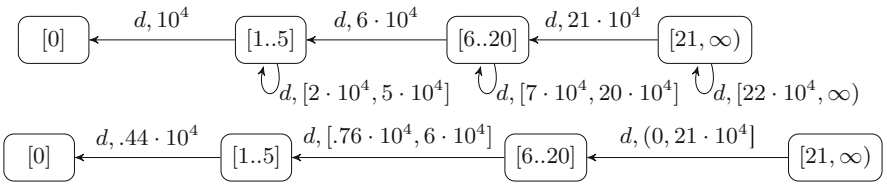


Fig. 1. Above: Interval CTMDP abstraction with intervals on rates and non-determinism. Below: Interval CTMC abstraction arising from acceleration.

Acceleration. To address this issue, we drop the non-deterministic self-loops and transitions to higher/lower populations in the abstract system.[2] Instead,

[2] One can also preserve the non-determinism for the special case when one of the transitions leads to a state where some action ceases to be enabled. While this adds more precision, the non-determinism in the abstraction makes it less convenient to handle.

we *"accelerate"* their effect: We consider sequences of these actions that in the concrete system have the effect of changing the population level. In our example above, we need to take the transition 1 to 13 times from [6..20] with various rates depending on the current concrete population, in order to get to [1..5]. This makes the precise timing more complicated to compute. Nevertheless, the expected time can be approximated easily: here it ranges from $\frac{1}{6} \cdot 10^4 = 0.17 \cdot 10^4$ (for population 6) to roughly $(\frac{1}{20} + \frac{1}{19} + \cdots + \frac{1}{6}) \cdot 10^4 = 1.3 \cdot 10^4$ (for population 20). This results in an interval CTMC.[3]

Concurrency in Acceleration. The accelerated transitions can due to higher number of occurrences be considered continuous or deterministic, as opposed to discrete stochastic changes as distinguished in the hybrid approach. The usual differential equation approach would also take into account other reactions that are modelled deterministically and would combine their effect into one equation. In order to simplify the exposition and computation and—as we see later—without much loss of precision, we can consider only the fastest change (or non-deterministically more of them if their rates are similar).[4]

3.2 Operational Semantics: Concretisation to a Representative

The next disadvantage of classical abstraction philosophy, manifested in the interval CTMC above is that the precise-valued intervals on rates imply high computational effort during the analysis. Although the system is smaller, standard transient analysis is still quite expensive.

Concretisation. In order to deal with this issue, the interval can be approximated roughly by the expected time it would take for an average population in the considered range, in our example the "average" representative is 13. Then the first transition occurs with rate $13 \cdot 10^{-4} = 10^{-3}$ and needs to happen 7 times, yielding expected time $7/13 \cdot 10^4 = 0.5 \cdot 10^4$ (ignoring even the precise slow downs in the rates as the population decreases). Already this very rough computation yields relative precision with factor 3 for all the populations in this interval, thus yielding the correct order of magnitude with virtually no effort. We lift the concretisation naturally to states and denote the concretisation of abstract state s by $\gamma(s)$. The complete procedure is depicted in Algorithm 1.

The concretisation is one of the main points where we deliberately drop a lot of quantitative information, while still preserving some to conclude on big quantitative differences. Of course, the precision improves with more precise abstract domains and also with higher differences on the original rates.

[3] The waiting times are not distributed according to the rates in the intervals. It is only the expected waiting time (reciprocal of the rate) that is preserved. Nevertheless, for ease of exposition, instead of labelling the transitions with expected waiting times we stick to the CTMC style with the reciprocals and formally treat it as if the label was a real rate.

[4] Typically the classical concurrency diamond appears and the effect of the other accelerated reactions happen just after the first one.

Algorithm 1. Semi-quantitative abstraction CTMC $\alpha(\mathcal{N})$

$A \leftarrow \prod_{\lambda \in \Lambda} A_\lambda$ ▷ States
for $\boldsymbol{a} \in A$ **do** ▷ Transitions
  $\boldsymbol{c} \leftarrow \gamma(\boldsymbol{a})$ ▷ Concrete representative
  for each τ enabled in $\boldsymbol{c}$ **do**
    $r \leftarrow$rate of τ in $\boldsymbol{c}$ ▷ According to (R)
    $\boldsymbol{a}' \leftarrow \alpha(\boldsymbol{c} + \upsilon_\tau)$ ▷ Successor
    set $\boldsymbol{a} \xrightarrow{\tau} \boldsymbol{a}'$ with rate r
  for self-loop $\boldsymbol{a} \xrightarrow{\tau} \boldsymbol{a}$ **do** ▷ Accelerate self-loops
    $n_\tau \leftarrow \min\{n \mid \alpha(\boldsymbol{c} + n \cdot \upsilon_\tau) \neq \boldsymbol{a}\}$ ▷ the number of τ to change the abstract state
    $\boldsymbol{a}' \leftarrow \alpha(\boldsymbol{c} + n_\tau \cdot \upsilon_\tau)$ ▷ Acceleration successor
    instead of the self-loop with rate r, set $\boldsymbol{a} \xrightarrow{\tau} \boldsymbol{a}'$ with rate $n_\tau \cdot r$

It remains to determine the representative for the unbounded interval. In order to avoid infinity, we require an additional input for the analysis, which are deemed upper bounds on possible population of each species. In cases when any upper bound is hard to assume, we can analyse the system with a random one and see if the last interval is reachable with significant probability. If yes, then we need to use this upper bound as a new point in the interval partitioning and try a higher upper bound next time. In general, such conditions can be checked in the abstraction and their violation implies a recommendation to refine the abstract domains accordingly.

Orders-of-Magnitude Abstraction. Such an approximation is thus sufficient to determine most of the time whether the acceleration (sequence of actions) happens sooner or later than e.g. another reaction with rate 10^{-6} or 10^{-2}. Note that this *decision* gets more precise not only as we refine the population levels, but also as the system gets stiffer (the concrete values of the rates differ more), which are normally harder to analyse. For the ease of presentation in our case studies, we shall depict only the magnitude of the rates, i.e. the decadic logarithm rounded to an integer.

Non-determinism and Refinement. If two rates are close to each other, say of the same magnitude (or difference 1), such a rough computation (and rough population discretisation) is not precise enough to determine which of the reactions happens with high probability sooner. Both may be happening roughly at the same pace, or with more information we could conclude one of them is considerably faster. This introduces an uncertainty, showing different behaviours are possible depending on the exact quantities. This indicates points where refinement might be needed if more precise results are required. For instance, with rates of magnitudes 2 and 3, the latter should be happing most of the time, the former only with a few percent chance. If we want to know whether it is rather tens of percent or tenths of percent, we should refine the abstraction.

4 Semi-quantitative Analysis

In this section, we present an approximative analysis technique that describes the most probable transient and steady-state behaviour of the system (also with rough timing) and on demand also the (one or more orders of magnitude) less probable behaviours. As such it is robust in the sense that it is well suited to work with imprecise rates and populations. It is computationally easy (can be done in hand in time required for a computer by other methods), while still yielding significant quantitative results ("in orders of magnitude"). It does not provide exact error guarantees since computing them would be almost as expensive as the classical analysis. It only features trivial limit-style bounds: if the population abstraction gets more and more refined, the probabilities converge to those of the original system; further, the higher the separation between the rate magnitudes, the more precise the approximation is since the other factors (and thus the incurred imprecisions) play less significant role.

Intuitively, the main idea—similar to some multi-rate simulation techniques for stiff systems—is to "simulate" "fast" reactions until the steady state and then examine which slower reactions take place. However, "fast" does not mean faster than some constant, but faster than other transitions in a given state. In other words, we are not distinguishing fast and slow reactions, but tailor this to each state separately. Further, "simulation" is not really a stochastic simulation, but a deterministic choice of the fastest available transition. If a transition is significantly faster than others then this yields what a simulation would yield. When there are transitions with similar rates, e.g. with at most one order of magnitude difference, then both are taken into account as described in the following definition.

Pruned System. Consider the underlying graph of the given CTMC. If we keep only the outgoing transitions with the maximum rate in each state, we call the result *pruned*. If there is always (at most) one transition then the graph consists of several paths leading to cycles. In general when more transitions are kept, it has bottom strongly connected components (bottom SCCs, BSCCs) and some transient parts.

We generalise this concept to *n-pruning* that preserves all transitions with a rate that is not more than n orders of magnitude smaller than the maximum rate in the state. Then the pruning above is 0-pruning, 1-pruning preserves also transitions happening up to 10 times slower, which can thus still happen with dozens of percent, 2-pruning is relevant for analysis where behaviour occurring with units of percent is also tracked etc.

Algorithm Idea. Here we explain the idea of Algorithm 2. The transient parts of the pruned system describe the most probable behaviour from each state until the point where visited states start to repeat a lot (steady state of the pruned system). In the original system, the usual behaviour is then to stay in this SCC C until one of the pruned (slower) reactions occurs, say from state s to state t. This may bring us to a different component of the pruned graph and the analysis process repeats. However, t may also bring us back into C, in which case we stay

in the steady-state, which is basically the same as without the transition from s to t. Further, t might be in the transient part leading to C, in which case these states are added to C and the steady state changes a bit, spreading the distribution slightly also to the previously transient states. Finally, t might be leading us into a component D where this run was previous to visiting C. In that case, the steady-state distribution spreads over all the components visited between D and C, putting a probability mass to each with a different order of magnitude depending on all the (magnitudes of) sojourn times in the transient and steady-state phases on the way.

Using the macros defined in the algorithm, the correctness of the computations can be shown as follows. For the time spent in the transient phase (line 16), we consider the slowest sojourn time on the way times the number of such transitions; this is accurate since the other times are by order(s) of magnitude shorter, hence negligible. The steady-state distribution on a BSCC of the

Algorithm 2. Semi-quantitative analysis

1: $W \leftarrow \emptyset$ ▷ worklist of SCCs to process
2: add {initial state} to W and assign iteration 0 to it ▷ artificial SCC to start the process
3: **while** $W \neq \emptyset$ **do**
4: $C \leftarrow$pop W
▷ Compute and output steady state or its approximation
5: steady-state of C is approximately $minStayingRate/(m \cdot stayingRate(\cdot))$
6: **if** C has no exits **then** continue ▷ definitely bottom SCC, final steady state
▷ Compute and output exiting transitions and the time spent in C
7: $exitStates \leftarrow \arg\min_C(stayingRate(\cdot)/exitingRate(\cdot))$ ▷ Probable exit points
8: $minStayingRate \leftarrow$minimum rate in C, $m \leftarrow$#occurrences there
9: $timeToExit \leftarrow stayingRate(s) \cdot m/(|exitStates| \cdot minStayingRate \cdot exitingRate(s))$
for (arbitrary) $s \in exitStates$
10: **for all** $s \in exitsStates$ **do** ▷ Transient analysis
11: $t \leftarrow$target of the exiting transition
12: $T \leftarrow$SCCs reachable in the pruned graph from t
13: thereby newly reached transitions get assigned iteration of $C + 1$
14: **for** $D \in T$ **do**
▷ Compute and output time to get from t to D
15: $minRate \leftarrow$minimum rate on the way from t to D, $m \leftarrow$#occurrences there
16: $transTime \leftarrow m/minRate$
▷ Determine the new SCC
17: **if** $D = C$ **then** ▷ back to the current SCC
18: add to W the union of C and the new transient path τ from t to C
19: in later steady-state computation, the states of τ will have probability
smaller by a factor of $stayingRate(s)/exitingRate(s)$
20: **else if** D was previously visited **then** ▷ alternating between different SCCs
21: add to W the merge of all SCCs visited between D and C (inclusively)
22: in later steady-state computation, reflect all $timeToExit$ and $transTime$
between D and C
23: **else** ▷ new SCC
24: add D to W

MACROS:
$stayingRate(s)$ is the rate of transitions from s in the pruned graph
$exitingRate(s)$ is the maximum rate of transitions from s not in the pruned graph

pruned graph can be approximated by the $minStayingRate/(m \cdot stayingRate(\cdot))$ on line 5. Indeed, it corresponds to the steady-state distribution if the BSCC is a cycle and the *minStayingRate* significantly larger than other rates in the BSCC since then the return time for the states is approximately $m/minStayingRate$ and the sojourn time $1/stayingRate(\cdot)$. The component is exited from s with the proportion given by its steady-state distribution times the probability to take the exit during that time. The former is approximated above; the latter can be approximated by the density in 0, i.e. by $exitingRate(s)$, since the staying rate is significantly faster. Hence the candidates for exiting are maximising $exitingRate(\cdot)/stayingRate(\cdot)$ as on line 7. There are $|exitStates|$ candidates for exit and the time to exit the component by a particular candidate s is the expected number of visits before exit, i.e. $stayingRate(s) \cdot exitingRate(s)$ times the return time $m \cdot minStayingRate$, hence the expression on line 9.

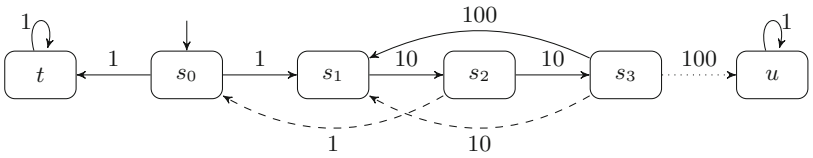


Fig. 2. Alternating transient and steady-state analysis.

For example, consider the system in Fig. 2. Iteration 1 reveals the part with solid lines with two (temporary) BSCCs $\{t\}$ and $\{s_1, s_2, s_3\}$. The former turns out definitely bottom. The latter has a steady state proportional to $(10^{-1}, 10^{-1}, 100^{-1})$. Its most probable exits are the dashed ones, identified in the subsequent iteration 2, probable proportionally to (1/10,10/100); the expected time to take them is $10 \cdot 2/(2 \cdot 10 \cdot 1) = 1 = 100 \cdot 2/(2 \cdot 10 \cdot 10)$. The latter leads back to the current SCC and does not change the set of BSCCs (hence in our examples below we often either skip or merge such iterations for the sake of readability). In contrast, the former leads to a previous SCC; thereafter $\{s_1, s_2, s_3\}$ is no more a bottom SCC and consequently the third exit to u is not even analysed. Nevertheless, it could still happen with minor probability, which can be seen if we consider 1-pruning instead.

5 Experimental Evaluation and Discussion

In order to demonstrate the applicability and accuracy of our approach, we selected the following three biologically relevant case studies. (1) stochastic model of gene expression [22,24], (2) Goutsias's model [23] describing transcription regulation of a repressor protein in bacteriophage λ and (3) viral infection model [43].

Although the underlying CRNs are quite small (up to 5 species and 10 reaction), their analysis is very challenging: (i) the stochasticity has a strong impact

on the dynamics of these systems and thus purely deterministic approximations via ODEs are not accurate, (ii) the systems include species with low, medium, and high populations and thus the resulting state space of the stochastic process is prohibitively large to perform precise numerical analysis and existing reduction/approximation techniques are not sufficient (they are either too imprecise or do not provide sufficient reduction factors), and (iii) the system dynamics leads to bi-modal distributions and/or is affected by stiff reactions.

These models thus represent perfect candidates for evaluating advanced approximation methods including various hybrid approaches [9,24,27]. Although these approaches can handle the models, they typically require tens of minutes or hours of computation time. Similarly simulation-based methods are very time consuming especially in case of very stiff CRN, represented by the viral infection model. We demonstrate that our approach provides accurate predications of the system behaviour and is feasible even when performed manually by a human.

Recall that the algorithm that builds the abstract model of the given CRN takes as input two vectors representing the population discretisation and population bounds. We generally assume that these inputs are provided by users who have a priori knowledge about the system (e.g. in which orders the species population occurs) and that the inputs also reflect the level of details the users are interested in. In the following case studies, we, however, set the inputs only based on the rate orders of the reactions affecting the particular species (unless mentioned otherwise).

5.1 Gene Expression Model

The CRN underlying the gene expression model is described in Table 1. As discussed in [24] and experimentally observed in [18], the system oscillates between two phases characterised by the D_{on} state and the D_{off} state, respectively. Biologists are interested in how the distribution of the D_{on} and D_{off} states is aligned with the distribution of RNA and proteins P, and how the correlation among the distributions depends on the DNA switching rates.

The state vector of the underlying CTMC is given as [P, RNA, D_{off}, D_{on}]. We use very relaxed bounds on the maximal populations, namely the bound 1000 for P and 100 for RNA. Note the DNA invariant $D_{on} + D_{off} = 1$. As in [24], the initial state is given as [10,4,1,0].

We first consider the slow switching rates that lead to a more complicated dynamics including bimodal distributions. In order to demonstrate the refinement step and its effect on the accuracy of the model, we start with a very coarse abstraction. It distinguishes only the zero population and the non-zero populations and thus it is not able to adequately capture the relationship between the DNA state and RNA/P population. The pruned abstract model obtained using Algorithm 1 and 2 is depicted in Fig. 3 (left). The full one before pruning is shown in Fig. 6 [11, Appendix].

The proposed analysis of the model identifies the key trends in the system dynamic. The red transitions, representing iterations 1–3, capture the most probable paths in the system. The green component includes states with DNA on

Table 1. Gene expression. For slow DNA switching, $r_1 = r_2 = 0.05$. For fast DNA switching, $r_1 = r_2 = 1$. The rates are in h^{-1}.

$D_{off} \xrightarrow{r_1} D_{on}$	$D_{on} \xrightarrow{r_2} D_{off}$	$D_{on} \xrightarrow{10} D_{on} + RNA$	$RNA \xrightarrow{1} \emptyset$
$RNA \xrightarrow{4} RNA + P$	$P \xrightarrow{1} \emptyset$	$P + D_{off} \xrightarrow{0.0015} P + D_{on}$	

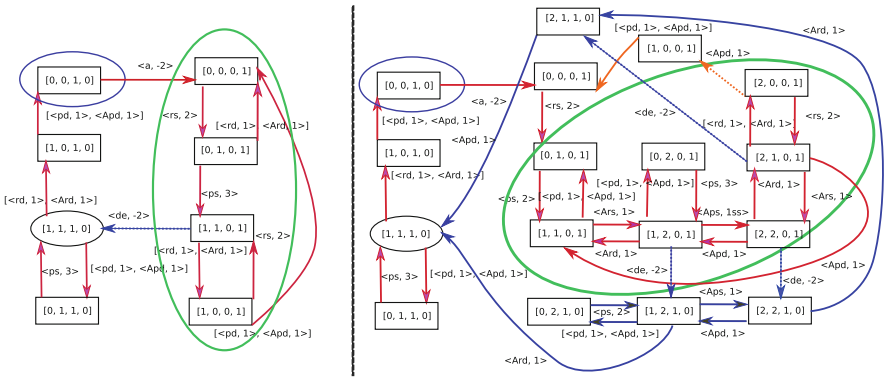


Fig. 3. Pruned abstraction for the gene expression model using the coarse population discretisation (left) and after the refinement (right). The state vector is [P, RNA, D_{off}, D_{on}].

(i.e. $D_{on} = 1$) where the system oscillates. The component is reached via the blue state with D_{off} and no RNAs/P. The blue state is promptly reached from the initial state and then the system waits (roughly 100 h according our rate abstraction) for the next DNA activation. The oscillation is left via a deactivation in the iteration 4 (the blue dotted transition)[5]. The estimation of the exit time computed using Algorithm 2 is also 100 h. The deactivation is then followed by fast red transitions leading to the blue state, where the system waits for the next activation. Therefore, we obtain an oscillation between the blue state and the green component, representing the expected oscillation between the D_{on} and D_{off} states.

As expected, this abstraction does not clearly predict the bimodal distribution on the RNA/P populations as the trivial population levels do not bear any information beside reaction enabledness. In order to obtain a more accurate analysis of the system, we refine the population discretisation using a single level threshold for P and DNA, that is equal to 100 and 10, respectively (the rates in the CRN indicate that the population of P reaches higher values).

Figure 3 (right) depicts the pruned abstract model with the new discretisation (the full model is depicted in Fig. 7 [11, Appendix]. We again obtain the oscillation between the green component representing DNA_{on} states and the blue DNA_{off} state. The states in the green component more accurately predicts

[5] In Fig. 3, the dotted transitions denote exit transitions representing the deactivations.

that in the DNA_{on} states the populations of RNA and P are high and drop to zero only for short time periods. The figure also shows orange transitions within the iteration 2 that extend the green component by two states. Note that the system promptly returns from these states back to the green component. After the deactivation in the iteration 4, the system takes (within the same iteration) the fast transitions (solid blue) leading to the blue component where system waits for another activation and where the mRNA/protein populations decrease. The expected time spent in states on blue solid transitions is small and thus we can reliably predict the bimodal distribution of the mRNA/P populations and its correlation with the DNA state. The refined abstraction also reveals that the switching time from the DNA_{on} mode to the DNA_{off} mode is lower. These predications are in accordance with the results obtained in [24]. See Fig. 8 [11, Appendix] that is adopted from [24] and illustrates these results.

To further test the accuracy of our approach, we consider the fast switching between the DNA states. We follow the study in [24] and increase the rates by two orders of magnitude. We use the refined population discretisation and obtain a very similar abstraction as in Fig. 3 (right). We again obtain the oscillation between the green component (DNA_{on} states and nonzero RNA/protein populations) and the blue state (DNA_{off} and zero RNA/protein populations). The only difference is in fact the transition rates corresponding to the activation and deactivation causing that the switching rate between the components is much faster. As a consequence, the system spends a longer period in the blue transient states with D_{off} and nonzero RNA/protein populations. The time spent in these states decreases the correlation between the DNA state and the RNA/protein populations as well as the bimodality in the population distribution. This is again in the accordance with [24].

To conclude this case study, we observe a very aligned agreement between the results obtained using our approach and results in [24] obtained via advanced and time consuming numerical methods. We would like to emphasise that our abstraction and its solution is obtained within a fraction of a second while the numerical methods have to approximate solutions of equations describing high-order conditional moments of the population distributions. As [24] does not report the runtime of the analysis and the implementation of their methods is not publicly available, we cannot directly compare the time complexity.

5.2 Goutsias's Model

Goutsias's model illustrated in Table 2 is widely used for evaluation of various numerical and simulation based techniques. As showed e.g. in [23], the system has with a high probability the following transient behaviour. In the first phase, the system switches with a high rate between the non-active DNA (denoted DNA) and the active DNA (DNA.D). During this phase the population of RNA, monomers (M) and dimers (D) gradually increase (with only negligible oscillations). After around 15 min, the DNA is blocked (DNA.2D) and the population of RNA decreases while the population of M and D is relatively stable. After all RNA degrades (around another 15 min) the system switches to the third

Table 2. Goutsias' Model. The rates are in s^{-1}

$\text{RNA} \xrightarrow{0.043} \text{RNA} + \text{M}$	$\text{M} \xrightarrow{7\times10^{-4}} \emptyset$	$\text{RNA} \xrightarrow{4\times10^{-3}} \emptyset$
$\text{DNA} + \text{D} \xrightarrow{0.002} \text{DNA.D}$	$\text{DNA.D} \xrightarrow{0.48} \text{DNA} + \text{D}$	
$\text{DNA.D} + \text{D} \xrightarrow{2\times10^{-4}} \text{DNA.2D}$	$\text{M+M} \xrightarrow{0.083} \text{D}$	$\text{D} \xrightarrow{0.5} \text{M} + \text{M}$
$\text{DNA.2D} \xrightarrow{9\times10^{-12}} \text{DNA.D} + \text{D}$	$\text{DNA.D} \xrightarrow{0.072} \text{RNA} + \text{DNA.D}$	

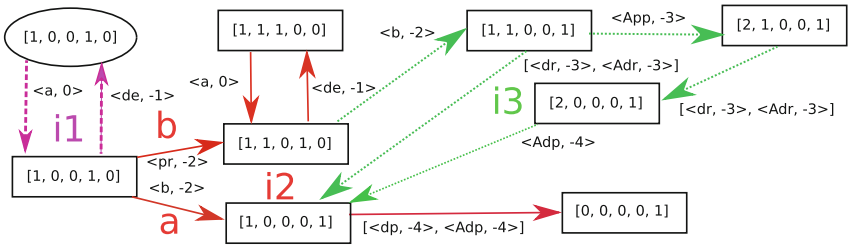


Fig. 4. Pruned abstraction for the Goutsias' model. The state vector is [M + D, RNA, DNA, DNA.D, DNA.2D]

phase where the population of M and D slowly decreases. Further, there is a non-negligible probability that the DNA is blocked at the beginning while the population of RNA is still small and the system promptly dies out.

Although the system is quite suitable for the hybrid approaches (there is no strong bimodality and only a limited stiffness), the analysis still takes 10 to 50 min depending on the required precision [27]. We demonstrate that our approach is able to accurately predict the main transient behaviour as well as the non-negligible probability that the system promptly dies out.

The state vector is given as [M, D, RNA, DNA, DNA.D, DNA.2D] and the initial state is set to [2, 6, 0, 1, 0, 0] as in [27]. We start our analysis with a coarse population discretisation with a single threshold 100 for M and D and a single threshold 10 for RNA. We relax the bounds, in particular, 1000 for M and D, and 100 for RNA. Note that these numbers were selected solely based on the rate orders of the relevant reactions. Note the DNA invariant DNA + DNA.D + DNA.2D = 1.

Figure 4 illustrates the pruned abstract model we obtained (the full model is depicted in Fig. 9 [11, Appendix]. For a better visualisation, we merged the state components corresponding to M and D into one component with M + D. As there is the fast reversible dimerisation, the actual distributions between the population of M and D does not affect the transient behaviour we are interested in.

The analysis of the model shows the following transient behaviour. The purple dotted loop in the iteration `i1` represents (de-)activation of the DNA. The expected exit time of this loop is 100 s. According to our abstraction, there are two options (with the same probability) to exit the loop: (1) the path `a` rep-

resents the DNA blocking followed by the quick extinction and (2) the path b corresponds to the production of RNA and its followed by the red loop in the i2 that again represents (de-)activation of the DNA. Note that according our abstraction, this loop contains states with the populations of M/D as well as RNA up to 100 and 10, respectively.

The expected exit time of this loop is again 100 s and there are two options how to leave the loop: (1) the path within the iteration $i3$ (taken with roughly 90%) represents again the DNA blocking and it is followed by the extension of RNA and consequently by the extension of M/D in about 1000 s and (2) the path within the iteration 5 (shown in the full graph in Fig. 9 [11, Appendix]) taken with roughly 10% represents the series of protein productions and leads to the states with a high number of proteins (above 100 in our population discretisation). Afterwards, there is again a series of DNA (de-)activations followed by the DNA blocking and the extinction of RNA. As before, this leads to the extinction of M/D in about 1000 s.

Although this abstraction already shows the transient behaviour leading to the extinction in about 30 min, it introduces the following inaccuracy with respect to the known behaviour: (1) the probability of the fast extinction is higher and (2) we do not observe the clear bell-shape pattern on the RNA (i.e. the level 2 for the RNA is not reached in the abstraction). As in the previous case study, the problem is that the population discretisation is too coarse. It causes that the total rate of the DNA blocking (affected by the M/D population via the mass action kinetics) is too high in the states with the M/D population level 1. This can be directly seen in the interval CTMC representation where the rate spans many orders of magnitude, incurring too much imprecision. The refinement of the M/D population discretisation eliminates the first inaccuracy. To obtain the clear bell-shape patter on RNA, one has to refine also the RNA population discretisation.

5.3 Viral Infection

The viral infection model described in Table 3 represents the most challenging system we consider. It is highly stochastic, extremely stiff, with all species presenting high variance and some also very high molecular populations. Moreover, there is a bimodal distribution on the RNA population. As a consequence, the solution of the full CME, even using advanced reduction and aggregation techniques, is prohibitive due to state-space explosion and stochastic simulation are very time consuming. State-of-the-art hybrid approaches integrating the LNA and an adaptive population partitioning [9] can handle this system but also need a very long execution time. For example, a transient analysis up to time $t = 50$ requires around 20 min and up to $t = 200$ more than an hour.

To evaluate the accuracy of our approach on this challenging model, we also focus on the same transient analysis, namely, we are interested in the distribution of RNA at time $t = 200$. The analysis in [9] predicts a bimodal distribution where, the probability that RNA is zero in around 20% and the remaining probability has Gaussian distribution with mean around 17 and the probability that there

Table 3. Viral Infection. The rates are day^{-1}

$$\mathrm{DNA} + \mathrm{P} \xrightarrow{7\times10^{-6}} \mathrm{V} \qquad \mathrm{DNA} \xrightarrow{0.025} \mathrm{DNA} + \mathrm{R} \qquad \mathrm{RNA} \xrightarrow{0.25} \emptyset$$
$$\mathrm{RNA} \xrightarrow{1} \mathrm{RNA} + \mathrm{DNA} \qquad \mathrm{RNA} \xrightarrow{1000} \mathrm{RNA.} + \mathrm{P} \qquad \mathrm{P} \xrightarrow{1.99} \emptyset$$

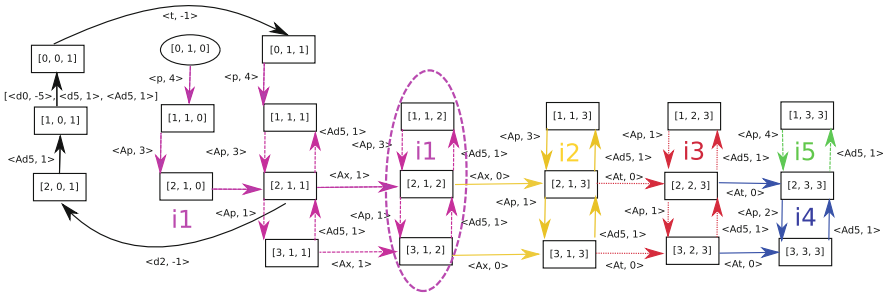


Fig. 5. Pruned abstraction for the viral infection model. The state vector is [P, RNA, DNA].

is more than 30 RNAs is close to zero. This is confirmed by simulation-based analysis in [23] showing also the gradual growth of the RNA population. The simulation-based analysis in [43], however, estimates a lower probability (around 3%) that RNA is 0 and higher mean of the remaining Gaussian distribution (around 23). Recall that obtaining accurate results using simulations is extremely time consuming due to very stiff reactions (a single simulation for $t = 200$ takes around 20 s).

In the final experiments, we analyse the distribution of RNA at time $t = 200$ using our approach. The state vector is given as [P, RNA, DNA] and we start with the concrete state [0, 1, 0]. To sufficiently reason about the RNA population and to handle the very high population of the proteins, we use the following population discretisation: thresholds {10, 1000} for P, {10, 30} for RNA, and {10, 100} for DNA. As before, we use very relaxed bounds 10000, 100, and 1000 for P, RNA, and D, respectively. Note that we ignore the population of the virus V as it does not affect the dynamics of the other species. This simplification makes the visualisation of our approach more readable and has no effect on the complexity of the analysis.

Figure 5 illustrates the obtained abstract model enabling the following transient analysis (the full model is depicted in Fig. 10 [11, Appendix]. In a few days the system reaches from the initial state the loop (depicted by the purple dashed ellipse) within the iteration *i1*. The loop includes states where RNA has level 1, DNA has level 2 and P oscillates between the levels 2 and 3. Before entering the loop, there is a non-negligible probability (orders of percent) that the RNA drops to 0 via the full black branch that returns to transient part of the loop in *i1*. In this branch the system can also die out (not shown in this figure, see the full model) with probability in the order of tenths of percent.

The average exit time of the loop in *i1* is in the order of 10 days and the system goes to the yellow loop within the iteration *i2*, where the DNA level is increased to 3 (RNA level is unchanged and P again oscillates between the levels 2 and 3). The average exit time of the loop in *i2* is again in the order of 10 days and systems goes to the dotted red loop within iteration *i3*. The transition represents the sequence of RNA synthesis that leads to RNA level 2. P oscillates as before. Finally, the system leaves the loop in *i3* (this takes another dozen days) and reaches RNA level 3 in iterations *i4* and *i5* where the DNA level remains at the level 3 and P oscillates. The iteration *i4* and *i5* thus roughly correspond to the examined transient time $t = 200$.

The analysis clearly demonstrates that our approach leads to the behaviour that is well aligned with the previous experiments. We observed growth of the RNA population with a non-negligible probability of its extinction. The concrete quantities (i.e. the probability of the extinction and the mean RNA population) are closer to the analysis in [43]. The quantities are indeed affected by the population discretisation and can be further refined. We would like to emphasise that in contrast to the methods presented in [9,23,43] requiring hours of intensive numerical computation, our approach can be done even manually on the paper.

References

1. Abate, A., Katoen, J.P., Lygeros, J., Prandini, M.: Approximate model checking of stochastic hybrid systems. Eur. J. Control **16**, 624–641 (2010)
2. 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
3. Angluin, D., Aspnes, J., Eisenstat, D., Ruppert, E.: The computational power of population protocols. Distrib. Comput. **20**(4), 279–304 (2007)
4. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press, Cambridge (2008)
5. 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
6. Buchholz, P.: Exact performance equivalence: an equivalence relation for stochastic automata. Theor. Comput. Sci. **215**(1–2), 263–287 (1999)
7. Cao, Y., Gillespie, D.T., Petzold, L.R.: The slow-scale stochastic simulation algorithm. J. Chem. Phys. **122**(1), 014116 (2005)
8. Cardelli, L.: Two-domain DNA strand displacement. Math. Struct. Comput. Sci. **23**(02), 247–271 (2013)
9. Cardelli, L., Kwiatkowska, M., Laurenti, L.: A stochastic hybrid approximation for chemical kinetics based on the linear noise approximation. In: Bartocci, E., Lio, P., Paoletti, N. (eds.) CMSB 2016. LNCS, vol. 9859, pp. 147–167. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45177-0_10
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. Češka, M., Křetínský, J.: Semi-quantitative abstraction and analysis of chemical reaction networks. Technical report abs/1905.09914, arXiv.org (2019)
12. Chellaboina, V., Bhat, S.P., Haddad, W.M., Bernstein, D.S.: Modeling and analysis of mass-action kinetics. IEEE Control Syst. Mag. **29**(4), 60–78 (2009)
13. Desharnais, J., Laviolette, F., Tracol, M.: Approximate analysis of probabilistic processes: logic, simulation and games. In: Quantitative Evaluation of SysTems (QEST), pp. 264–273. IEEE (2008)
14. D'Innocenzo, A., Abate, A., Katoen, J.P.: Robust PCTL model checking. In: Hybrid Systems: Computation and Control (HSCC), pp. 275–285. ACM (2012)
15. Engblom, S.: Computing the moments of high dimensional solutions of the master equation. Appl. Math. Comput. **180**(2), 498–515 (2006)
16. Ethier, S.N., Kurtz, T.G.: Markov Processes: Characterization and Convergence, vol. 282. Wiley, New York (2009)
17. Ferm, L., Lötstedt, P.: Adaptive solution of the master equation in low dimensions. Appl. Numer. Math. **59**(1), 187–204 (2009)
18. Gandhi, S.J., Zenklusen, D., Lionnet, T., Singer, R.H.: Transcription of functionally related constitutive genes is not coordinated. Nat. Struct. Mol. Biol. **18**(1), 27 (2011)
19. Ganguly, A., Altintan, D., Koeppl, H.: Jump-diffusion approximation of stochastic reaction dynamics: error bounds and algorithms. Multiscale Model. Simul. **13**(4), 1390–1419 (2015)
20. Giacobbe, M., Guet, C.C., Gupta, A., Henzinger, T.A., Paixão, T., Petrov, T.: Model checking gene regulatory networks. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 469–483. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46681-0_47
21. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. J. Phys. Chem. **81**(25), 2340–2361 (1977)
22. Golding, I., Paulsson, J., Zawilski, S.M., Cox, E.C.: Real-time kinetics of gene activity in individual bacteria. Cell **123**(6), 1025–1036 (2005)
23. Goutsias, J.: Quasiequilibrium approximation of fast reaction kinetics in stochastic biochemical systems. J. Chem. Phys. **122**(18), 184102 (2005)
24. Hasenauer, J., Wolf, V., Kazeroonian, A., Theis, F.: Method of conditional moments (MCM) for the chemical master equation. J. Math. Biol. 1–49 (2013). https://doi.org/10.1007/s00285-013-0711-5
25. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theor. Comput. Sci. **391**(3), 239–257 (2008)
26. Henzinger, T.A., Mateescu, M., Wolf, V.: Sliding window abstraction for infinite Markov chains. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 337–352. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02658-4_27
27. Henzinger, T.A., Mikeev, L., Mateescu, M., Wolf, V.: Hybrid numerical solution of the chemical master equation. In: Computational Methods in Systems Biology (CMSB), pp. 55–65. ACM (2010)
28. Hepp, B., Gupta, A., Khammash, M.: Adaptive hybrid simulations for multiscale stochastic reaction networks. J. Chem. Phys. **142**(3), 034118 (2015)
29. Karp, R.M., Miller, R.E.: Parallel program schemata. J. Comput. Syst. Sci. **3**(2), 147–195 (1969)
30. Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for continuous-time Markov chains. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 311–324. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-73368-3_37

31. Kwiatkowska, M., Thachuk, C.: Probabilistic model checking for biology. Softw. Syst. Saf. **36**, 165 (2014)
32. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of DNA strand displacement devices using probabilistic model checking. J. R. Soc. Interface **9**(72), 1470–1485 (2012)
33. Larsen, K.G., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. **94**(1), 1–28 (1991)
34. Madsen, C., Myers, C., Roehner, N., Winstead, C., Zhang, Z.: Utilizing stochastic model checking to analyze genetic circuits. In: Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp. 379–386. IEEE (2012)
35. Mateescu, M., Wolf, V., Didier, F., Henzinger, T.A.: Fast adaptive uniformization of the chemical master equation. IET Syst. Biol. **4**(6), 441–452 (2010)
36. Munsky, B., Khammash, M.: The finite state projection algorithm for the solution of the chemical master equation. J. Chem. Phys. **124**, 044104 (2006)
37. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
38. Rao, C.V., Arkin, A.P.: Stochastic chemical kinetics and the quasi-steady-state assumption: application to the Gillespie algorithm. J. Chem. Phys. **118**(11), 4999–5010 (2003)
39. Salis, H., Kaznessis, Y.: Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions. J. Chem. Phys. **122**(5), 054103 (2005)
40. Soloveichik, D., Seelig, G., Winfree, E.: DNA as a universal substrate for chemical kinetics. Proc. Natl. Acad. Sci. U. S. A. **107**(12), 5393–5398 (2010)
41. Soudjani, S.E.Z., Abate, A.: Adaptive and sequential gridding procedures for the abstraction and verification of stochastic processes. SIAM J. Appl. Dyn. Syst. **12**(2), 921–956 (2013)
42. Esmaeil Zadeh Soudjani, S., Abate, A.: Precise approximations of the probability distribution of a Markov process in time: an application to probabilistic invariance. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 547–561. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-54862-8_45
43. Srivastava, R., You, L., Summers, J., Yin, J.: Stochastic vs. deterministic modeling of intracellular viral kinetics. J. Theor. Biol. **218**(3), 309–321 (2002)
44. Van Kampen, N.G.: Stochastic Processes in Physics and Chemistry, vol. 1. Elsevier, New York (1992)
45. Zhang, J., Watson, L.T., Cao, Y.: Adaptive aggregation method for the chemical master equation. Int. J. Comput. Biol. Drug Des. **2**(2), 134–148 (2009)

Shepherding Hordes of Markov Chains

Milan Češka[1], Nils Jansen[2], Sebastian Junges[3(✉)], and Joost-Pieter Katoen[3]

[1] Brno University of Technology, Brno, Czech Republic
[2] Radboud University, Nijmegen, The Netherlands
[3] RWTH Aachen University, Aachen, Germany
sebastian.junges@cs.rwth-aachen.de

Abstract. This paper considers large families of Markov chains (MCs) that are defined over a set of parameters with finite discrete domains. Such families occur in software product lines, planning under partial observability, and sketching of probabilistic programs. Simple questions, like 'does at least one family member satisfy a property?', are NP-hard. We tackle two problems: distinguish family members that satisfy a given quantitative property from those that do not, and determine a family member that satisfies the property optimally, i.e., with the highest probability or reward. We show that combining two well-known techniques, MDP model checking and abstraction refinement, mitigates the computational complexity. Experiments on a broad set of benchmarks show that in many situations, our approach is able to handle families of millions of MCs, providing superior scalability compared to existing solutions.

1 Introduction

Randomisation is key to research fields such as dependability (uncertain system components), distributed computing (symmetry breaking), planning (unpredictable environments), and probabilistic programming. Families of alternative designs differing in the structure and system parameters are ubiquitous. Software dependability has to cope with configuration options, in distributed computing the available memory per process is highly relevant, in planning the observability of the environment is pivotal, and program synthesis is all about selecting correct program variants. The automated analysis of such families has to face a formidable challenge—in addition to the state-space explosion affecting each family member, the family size typically grows exponentially in the number of features, options, or observations. This affects the analysis of (quantitative) software product lines [18,28,43,45,46], strategy synthesis in planning under partial observability [12,14,29,36,41], and probabilistic program synthesis [9,13,27,40].

This paper considers families of Markov chains (MCs) to describe configurable probabilistic systems. We consider finite MC families with finite-state family members. Family members may have different transition probabilities and distinct topologies—thus different reachable state spaces. The latter aspect

This work has been supported by the DFG RTG 2236 "UnRAVeL" and the Czech Science Foundation grant No. Robust 17-12465S.

T. Vojnar and L. Zhang (Eds.): TACAS 2019, Part II, LNCS 11428, pp. 172–190, 2019.
https://doi.org/10.1007/978-3-030-17465-1_10

goes beyond the class of parametric MCs as considered in parameter synthesis [10,22,24,31] and model repair [6,16,42].

For an MC family $\mathfrak{D}$ and quantitative specification φ, with φ a reachability probability or expected reward objective, we consider the following synthesis problems: (a) does some member in $\mathfrak{D}$ satisfy a threshold on φ? (aka: *feasibility* synthesis), (b) which members of $\mathfrak{D}$ satisfy this threshold on φ and which ones do not? (aka: *threshold synthesis*), and (c) which family member(s) satisfy φ optimally, e.g., with highest probability? (aka: *optimal synthesis*).

The simplest synthesis problem, feasibility, is NP-complete and can naively be solved by analysing all individual family members—the so-called *one-by-one* approach. This approach has been used in [18] (and for qualitative systems in e.g. [19]), but is infeasible for large systems. An alternative is to model the family $\mathfrak{D}$ by a single Markov decision process (MDP)—the so-called *all-in-one* MDP [18]. The initial MDP state non-deterministically chooses a family member of $\mathfrak{D}$, and then evolves in the MC of that member. This approach has been implemented in tools such as ProFeat [18], and for purely qualitative systems in [20]. The MDP representation avoids the individual analysis of all family members, but its size is proportional to the family size. This approach therefore does not scale to large families. A symbolic BDD-based approach is only a partial solution as family members may induce different reachable state-sets.

This paper introduces an *abstraction-refinement* scheme over the MDP representation[1]. The abstraction *forgets* in which family member the MDP operates. The resulting *quotient* MDP has a single representative for every reachable state in a family member. It typically provides a very compact representation of the family $\mathfrak{D}$ and its analysis using off-the-shelf MDP model-checking algorithms yields a speed-up compared to the all-in-one approach. Verifying the quotient MDP yields under- and over-approximations of the min and max probability (or reward), respectively. These bounds are safe as all *consistent* schedulers, i.e., those that pick actions according to a single family member, are contained in all schedulers considered on the quotient MDP. (CEGAR-based MDP model checking for partial information schedulers, a slightly different notion than restricting schedulers to consistent ones, has been considered in [30]. In contrast to our setting, [30] considers history-dependent schedulers and in this general setting no guarantee can be given that bounds on suprema converge [29]).

Model-checking results of the quotient MDP do provide useful insights. This is evident if the resulting scheduler is consistent. If the verification reveals that the min probability exceeds r for a specification φ with a $\leq r$ threshold, then—even for inconsistent schedulers—it holds that all family members violate φ. If the model checking is inconclusive, i.e., the abstraction is too coarse, we iteratively refine the quotient MDP by splitting the family into sub-families. We do so in an efficient manner that avoids rebuilding the sub-families. Refinement employs a light-weight analysis of the model-checking results.

[1] Classical CEGAR for model checking of software product lines has been proposed in [21]. This uses feature transition systems, is purely qualitative, and exploits existential state abstraction.

We implemented our abstraction-refinement approach using the Storm model checker [25]. Experiments with case studies from software product lines, planning, and distributed computing yield possible speed-ups of up to 3 orders of magnitude over the one-by-one and all-in-one approaches (both symbolic and explicit). Some benchmarks include families of millions of MCs where family members are thousands of states. The experiments reveal that—as opposed to parameter synthesis [10,24,31]—the threshold has a major influence on the synthesis times.

To summarise, this work presents: (a) MDP-based abstraction-refinement for various synthesis problems over large families of MCs, (b) a refinement strategy that mitigates the overhead of analysing sub-families, and (c) experiments showing substantial speed-ups for many benchmarks. Extra material can be found in [1,11].

2 Preliminaries

We present the basic foundations for this paper, for details, we refer to [4,5].

Probabilistic models. A *probability distribution* over a finite or countably infinite set X is a function $\mu\colon X \to [0,1]$ with $\sum_{x\in X} \mu(x) = \mu(X) = 1$. The set of all distributions on X is denoted $Distr(X)$. The support of a distribution μ is $\text{supp}(\mu) = \{x \in X \mid \mu(x) > 0\}$. A distribution is *Dirac* if $|\text{supp}(\mu)| = 1$.

Definition 1 (MC). *A* discrete-time Markov chain *(MC) D is a triple $(S, s_0, \mathbf{P})$, where S is a finite set of states, $s_0 \in S$ is an initial state, and $\mathbf{P}\colon S \to Distr(S)$ is a transition probability matrix.*

MCs have unique distributions over successor states at each state. Adding nondeterministic choices over distributions leads to Markov decision processes.

Definition 2 (MDP). *A* Markov decision process *(MDP) is a tuple $M = (S, s_0, Act, \mathcal{P})$ where S, s_0 as in Definition 1, Act is a finite set of actions, and $\mathcal{P}\colon S \times Act \nrightarrow Distr(S)$ is a partial transition probability function.*

The *available actions* in $s \in S$ are $Act(s) = \{a \in Act \mid \mathcal{P}(s,a) \neq \bot\}$. An MDP with $|Act(s)| = 1$ for all $s \in S$ is an MC. For MCs (and MDPs), a state-reward function is $rew\colon S \to \mathbb{R}_{\geq 0}$. The reward $rew(s)$ is earned upon leaving s.

A *path* of an MDP M is an (in)finite sequence $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, where $s_i \in S$, $a_i \in Act(s_i)$, and $\mathcal{P}(s_i, a_i)(s_{i+1}) \neq 0$ for all $i \in \mathbb{N}$. For finite π, $\text{last}(\pi)$ denotes the last state of π. The set of (in)finite paths of M is Paths^M_{fin} (Paths^M). The notions of paths carry over to MCs (actions are omitted). Schedulers resolve all choices of actions in an MDP and yield MCs.

Definition 3 (Scheduler). *A* scheduler *for an MDP $M = (S, s_0, Act, \mathcal{P})$ is a function $\sigma\colon \mathsf{Paths}^M_{fin} \to Act$ such that $\sigma(\pi) \in Act(\text{last}(\pi))$ for all $\pi \in \mathsf{Paths}^M_{fin}$. Scheduler σ is* memoryless *if $\text{last}(\pi) = \text{last}(\pi') \implies \sigma(\pi) = \sigma(\pi')$ for all $\pi, \pi' \in \mathsf{Paths}^M_{fin}$. The set of all schedulers of M is Σ^M.*

Definition 4 (Induced Markov Chain). *The MC* induced by MDP M and $\sigma \in \Sigma^M$ *is given by* $M_\sigma = (\mathsf{Paths}^M_{fin}, s_0, \mathbf{P}^\sigma)$ *where:*

$$\mathbf{P}^\sigma(\pi, \pi') = \begin{cases} \mathcal{P}(\text{last}(\pi), \sigma(\pi))(s') & \text{if } \pi' = \pi \xrightarrow{\sigma(\pi)} s' \\ 0 & \text{otherwise.} \end{cases}$$

Specifications. For a MC D, we consider unbounded reachability specifications of the form $\varphi = \mathbb{P}_{\sim\lambda}(\Diamond G)$ with $G \subseteq S$ a set of goal states, $\lambda \in [0,1] \subseteq \mathbb{R}$, and $\sim \in \{<, \leq, \geq, >\}$. The probability to satisfy the path formula $\phi = \Diamond G$ in D is denoted by $\texttt{Prob}(D, \phi)$. If φ holds for D, that is, $\texttt{Prob}(D, \phi) \sim \lambda$, we write $D \models \varphi$. Analogously, we define expected reward specifications of the form $\varphi = \mathbb{E}_{\sim\kappa}(\Diamond G)$ with $\kappa \in \mathbb{R}_{\geq 0}$. We refer to λ/κ as *thresholds*. While we only introduce reachability specifications, our approaches may be extended to richer logics like arbitrary PCTL [32], PCTL* [3], or ω-regular properties.

For an MDP M, a specification φ holds ($M \models \varphi$) if and only if it holds for the induced MCs of all schedulers. The maximum probability $\texttt{Prob}^{\max}(M, \phi)$ to satisfy a path formula ϕ for an MDP M is given by a maximising scheduler $\sigma^{\max} \in \Sigma^M$, that is, there is no scheduler $\sigma' \in \Sigma^M$ such that $\texttt{Prob}(M_{\sigma^{\max}}, \phi) < \texttt{Prob}(M_{\sigma'}, \phi)$. Analogously, we define the minimising probability $\texttt{Prob}^{\min}(M, \phi)$, and the maximising (minimising) expected reward $\texttt{ExpRew}^{\max}(M, \phi)$ ($\texttt{ExpRew}^{\min}(M, \phi)$).

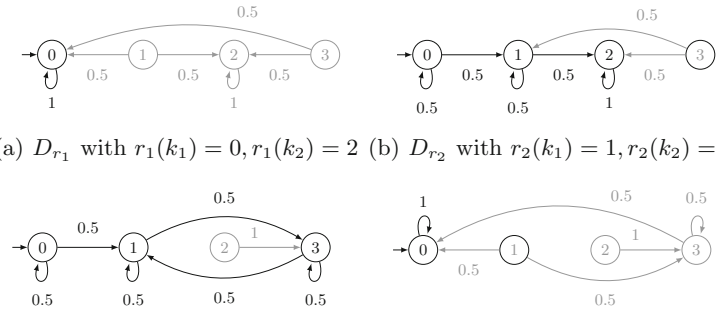
The probability (expected reward) to satisfy path formula ϕ from state $s \in S$ in MC D is $\texttt{Prob}(D, \phi)(s)$ ($\texttt{ExpRew}(D, \phi)(s)$). The notation is analogous for maximising and minimising probability and expected reward measures in MDPs. Note that the expected reward $\texttt{ExpRew}(D, \phi)$ to satisfy path formula ϕ is only defined if $\texttt{Prob}(D, \phi) = 1$. Accordingly, the expected reward for MDP M under scheduler $\sigma \in \Sigma^M$ requires $\texttt{Prob}(M_\sigma, \phi) = 1$.

3 Families of MCs

We present our approaches on the basis of an explicit representation of a *family of MCs* using a parametric transition probability function. While arbitrary probabilistic programs allow for more modelling freedom and complex parameter structures, the explicit representation alleviates the presentation and allows to reason about practically interesting synthesis problems. In our implementation, we use a more flexible high-level modelling language, cf. Sect. 5.

Definition 5 (Family of MCs). *A* family of MCs *is defined as a tuple* $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ *where* S *is a finite set of states,* $s_0 \in S$ *is an initial state,* K *is a finite set of discrete parameters such that the domain of each parameter* $k \in K$ *is* $T_k \subseteq S$*, and* $\mathfrak{P} \colon S \to Distr(K)$ *is a family of transition probability matrices.*

The transition probability function of MCs maps states to distributions over successor states. For families of MCs, this function maps states to distributions over parameters. Instantiating each of these parameters with a value from its domain yields a "concrete" MC, called a *realisation*.



(a) D_{r_1} with $r_1(k_1) = 0, r_1(k_2) = 2$ (b) D_{r_2} with $r_2(k_1) = 1, r_2(k_2) = 2$

(c) D_{r_3} with $r_3(k_1) = 1, r_3(k_2) = 3$ (d) D_{r_4} with $r_4(k_1) = 0, r_4(k_2) = 3$

Fig. 1. The four different realisations of $\mathfrak{D}$.

Definition 6 (Realisation). *A* realisation *of a family $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ is a function $r\colon K \to S$ where $\forall k \in K\colon r(k) \in T_k$. A realisation r yields a MC $D_r = (S, s_0, \mathfrak{P}(r))$, where $\mathfrak{P}(r)$ is the transition probability matrix in which each $k \in K$ in $\mathfrak{P}$ is replaced by $r(k)$. Let $\mathcal{R}^{\mathfrak{D}}$ denote the* set of all realisations *for $\mathfrak{D}$.*

As a family $\mathfrak{D}$ of MCs is defined over finite parameter domains, the number of family members (i.e. realisations from $\mathcal{R}^{\mathfrak{D}}$) of $\mathfrak{D}$ is finite, viz. $|\mathfrak{D}| := |\mathcal{R}^{\mathfrak{D}}| = \prod_{k \in K} |T_k|$, but exponential in $|K|$. Subsets of $\mathcal{R}^{\mathfrak{D}}$ induce so-called *subfamilies* of $\mathfrak{D}$. While all these MCs share the same state space, their *reachable* states may differ, as demonstrated by the following example.

Example 1 (Family of MCs). Consider a family of MCs $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ where $S = \{0, 1, 2, 3\}$, $s_0 = 0$, and $K = \{k_0, k_1, k_2\}$ with domains $T_{k_0} = \{0\}, T_{k_1} = \{0, 1\}$, and $T_{k_2} = \{2, 3\}$. The parametric transition function $\mathfrak{P}$ is defined by:

$$\mathfrak{P}(0) = 0.5\colon k_0 + 0.5\colon k_1 \qquad \mathfrak{P}(1) = 0.5\colon k_1 + 0.5\colon k_2$$
$$\mathfrak{P}(2) = 1\colon k_2 \qquad \mathfrak{P}(3) = 0.5\colon k_1 + 0.5\colon k_2$$

Figure 1 shows the four MCs that result from the realisations $\{r_1, r_2, r_3, r_4\} = \mathcal{R}^{\mathfrak{D}}$ of $\mathfrak{D}$. States that are unreachable from the initial state are greyed out.

We state two synthesis problems for families of MCs. The first is to identify the set of MCs satisfying and violating a given specification, respectively. The second is to find a MC that maximises/minimises a given objective. We call these two problems *threshold synthesis* and *max/min synthesis*.

Problem 1 (Threshold synthesis). *Let $\mathfrak{D}$ be a family of MCs and φ a probabilistic reachability or expected reward specification. The* threshold synthesis problem *is to partition $\mathcal{R}^{\mathfrak{D}}$ into T and F such that $\forall r \in T\colon D_r \vDash \varphi$ and $\forall r \in F\colon D_r \nvDash \varphi$.*

As a special case of the threshold synthesis problem, the *feasibility synthesis problem* is to find just one realisation $r \in \mathcal{R}^{\mathfrak{D}}$ such that $D_r \vDash \varphi$.

Problem 2 (Max synthesis). *Let $\mathfrak{D}$ a family of MCs and $\phi = \Diamond G$ for $G \subseteq S$. The* max synthesis problem *is to find a realisation $r^* \in \mathcal{R}^{\mathfrak{D}}$ such that $\texttt{Prob}(D_{r^*}, \phi) = \max_{r \in \mathcal{R}^{\mathfrak{D}}} \{\texttt{Prob}(D_r, \phi)\}$. The problem is defined analogously for an expected reward measure or minimising realisations.*

Example 2 (Synthesis problems). Recall the family of MCs $\mathfrak{D}$ from Example 1. For the specification $\varphi = \mathbb{P}_{\geq 0.1}(\Diamond\{1\})$, the solution to the threshold synthesis problem is $T = \{r_2, r_3\}$ and $F = \{r_1, r_4\}$, as the goal state 1 is not reachable for D_{r_1} and D_{r_4}. For $\phi = \Diamond\{1\}$, the solution to the max synthesis problem on $\mathfrak{D}$ is r_2 or r_3, as D_{r_2} and D_{r_3} have probability one to reach state 1.

Approach 1 (One-by-one [18]). *A straightforward solution to both synthesis problems is to enumerate all realisations $r \in \mathcal{R}^{\mathfrak{D}}$, model check the MCs D_r, and either compare all results with the given threshold or determine the maximum.*

We already saw that the number of realisations is exponential in $|K|$.

Theorem 1. *The feasibility synthesis problem is NP-complete.*

The theorem even holds for almost-sure reachability properties. The proof is a straightforward adaption of results for augmented interval Markov chains [17, Theorem 3], partial information games [15], or partially observable MDPs [14].

4 Guided Abstraction-Refinement Scheme

In the previous section, we introduced the notion of a family of MCs, two synthesis problems and the one-by-one approach. Yet, for a sufficiently high number of realisations such a straightforward analysis is not feasible. We propose a novel approach allowing us to more efficiently analyse families of MCs.

4.1 All-in-one MDP

We first consider a single MDP that subsumes all individual MCs of a family $\mathfrak{D}$, and is equipped with an appropriate action and state labelling to identify the underlying realisations from $\mathcal{R}^{\mathfrak{D}}$.

Definition 7 (All-in-one MDP [18,28,43]). *The* all-in-one MDP *of a family $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ of MCs is given as $M^{\mathfrak{D}} = (S^{\mathfrak{D}}, s_0^{\mathfrak{D}}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}})$ where $S^{\mathfrak{D}} = S \times \mathcal{R}^{\mathfrak{D}} \cup \{s_0^{\mathfrak{D}}\}$, $Act^{\mathfrak{D}} = \{a^r \mid r \in \mathcal{R}^{\mathfrak{D}}\}$, and $\mathcal{P}^{\mathfrak{D}}$ is defined as follows:*

$$\mathcal{P}^{\mathfrak{D}}(s_0^{\mathfrak{D}}, a^r)((s_0, r)) = 1 \quad \text{and} \quad \mathcal{P}^{\mathfrak{D}}((s, r), a^r)((s', r)) = \mathfrak{P}(r)(s)(s').$$

Example 3 (All-in-one MDP). Figure 2 shows the all-in-one MDP $M^{\mathfrak{D}}$ for the family $\mathfrak{D}$ of MCs from Example 1. Again, states that are not reachable from the initial state $s_0^{\mathfrak{D}}$ are marked grey. For the sake of readability, we only include the transitions and states that correspond to realisations r_1 and r_2.

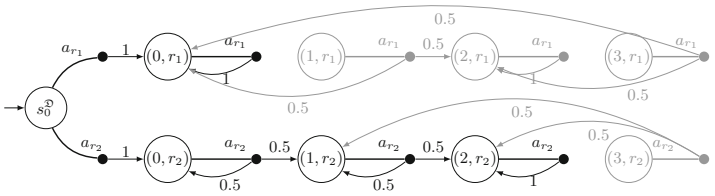


Fig. 2. Reachable fragment of the all-in-one MDP $M^{\mathfrak{D}}$ for realisations r_1 and r_2.

From the (fresh) initial state $s_0^{\mathfrak{D}}$ of the MDP, the choice of an action a_r corresponds to choosing the realisation r and entering the concrete MC D_r. This property of the all-in-one MDP is formalised as follows.

Corollary 1. *For the all-in-one MDP $M^{\mathfrak{D}}$ of family $\mathfrak{D}$ of MCs*[2]*:*

$$\{M^{\mathfrak{D}}_{\sigma^r} \mid \sigma^r \text{ memoryless deterministic scheduler}\} = \{D_r \mid r \in \mathcal{R}^{\mathfrak{D}}\}.$$

Consequently, the feasibility synthesis problem for φ has the solution $r \in \mathcal{R}^{\mathfrak{D}}$ iff there exists a memoryless deterministic scheduler σ^r such that $M^{\mathfrak{D}}_{\sigma^r} \vDash \varphi$.

Approach 2 (All-in-one [18]). *Model checking the all-in-one MDP determines max or min probability (or expected reward) for all states, and thereby for all realisations, and thus provides a solution to both synthesis problems.*

As also the all-in-one MDP may be too large for realistic problems, we merely use it as formal starting point for our abstraction-refinement loop.

4.2 Abstraction

First, we define a predicate abstraction that at each state of the MDP *forgets* in which realisation we are, i.e., abstracts the second component of a state (s, r).

Definition 8 (Forgetting). *Let $M^{\mathfrak{D}} = (S^{\mathfrak{D}}, s_0^{\mathfrak{D}}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}})$ be an all-in-one MDP.* Forgetting *is an equivalence relation $\sim_f \subseteq S^{\mathfrak{D}} \times S^{\mathfrak{D}}$ satisfying*

$$(s, r) \sim_f (s', r') \iff s = s' \text{ and } s_0^{\mathfrak{D}} \sim_f (s_0^{\mathfrak{D}}, r)\ \forall r \in \mathcal{R}^{\mathfrak{D}}.$$

Let $[s]_\sim$ denote the equivalence class wrt. $\sim_f$ containing state $s \in S^{\mathfrak{D}}$.

Forgetting induces the quotient MDP $M^{\mathfrak{D}}_{\sim} = (S^{\mathfrak{D}}_{\sim}, [s_0^{\mathfrak{D}}]_\sim, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}}_{\sim})$, *where* $\mathcal{P}^{\mathfrak{D}}_{\sim}([s]_\sim, a_r)([s']_\sim) = \mathfrak{P}(r)(s)(s')$.

At each state of the quotient MDP, the actions correspond to any realisation. It includes states that are unreachable in every realisation.

Remark 1 (Action space). According to Definition 8, for every state $[s]_\sim$ there are $|\mathfrak{D}|$ actions. Many of these actions lead to the same distributions over successor states. In particular, two different realisations r and r' lead to the same distribution in s if $r(k) = r'(k)$ for all $k \in K$ where $\mathfrak{P}(s)(k) \neq 0$. To avoid this spurious blow-up of actions, we *a-priori* merge all actions yielding the same distribution.

[2] The original initial state s_0 of the family of MCs needs to be the initial state of $M^{\mathfrak{D}}_{\sigma^r}$.

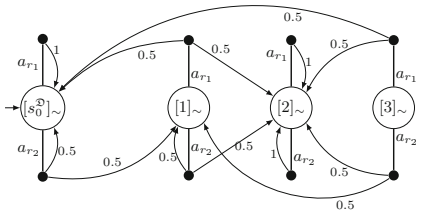


Fig. 3. The quotient MDP $M^{\mathfrak{D}}_{\sim}$ for realisations r_1 and r_2.

The quotient MDP under forgetting involves that the available actions allow to switch realisations and thereby create induced MCs different from any MC in $\mathfrak{D}$. We formalise the notion of a consistent realisation with respect to parameters.

Definition 9 (Consistent realisation). *For a family $\mathfrak{D}$ of MCs and $k \in K$, k-*realisation-consistency *is an equivalence relation $\approx_k \subseteq \mathcal{R}^{\mathfrak{D}} \times \mathcal{R}^{\mathfrak{D}}$ satisfying:*

$$r \approx_k r' \iff r(k) = r'(k).$$

Let $[r]_{\approx_k}$ denote the equivalence class w.r.t. $\approx_k$ containing $r \in \mathcal{R}^{\mathfrak{D}}$.

Definition 10 (Consistent scheduler). *For quotient MDP $M^{\mathfrak{D}}_{\sim}$ after forgetting and $k \in K$, a scheduler $\sigma \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ is* k-consistent *if for all $\pi, \pi' \in \mathsf{Paths}_{fin}^{M^{\mathfrak{D}}_{\sim}}$:*

$$\sigma(\pi) = a_r \wedge \sigma(\pi') = a_{r'} \implies r \approx_k r' \ .$$

A scheduler is K-consistent *(short:* consistent*) if it is k-consistent for all $k \in K$.*

Lemma 1. *For the quotient MDP $M^{\mathfrak{D}}_{\sim}$ of family $\mathfrak{D}$ of MCs:*

$$\{\left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}} \mid \sigma^{r^*} \text{ consistent scheduler}\} = \{D_r \mid r \in \mathcal{R}^{\mathfrak{D}}\}.$$

Proof (Idea). For $\sigma^r \in \Sigma^{M^{\mathfrak{D}}}$, we construct $\sigma^{r^*} \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ such that $\sigma^{r^*}([s]_\sim) = a_r$ for all s. Clearly σ^{r^*} is consistent and $M^{\mathfrak{D}}_{\sigma^r} = \left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}}$ is obtained via a map between (s, r) and $[s]_\sim$. For $\sigma^{r^*} \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$, we construct $\sigma^r \in \Sigma^{M^{\mathfrak{D}}}$ such that if $\sigma^{r^*}([s]_\sim) = a_r$ then $\sigma^r(s_0^{\mathfrak{D}}) = a_r$. For all other states, we define $\sigma^r((s, r')) = a^{r'}$ independently of σ^{r^*}. Then $M^{\mathfrak{D}}_{\sigma^r} = \left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}}$ is obtained as above.

The following theorem is a direct corollary: we need to consider exactly the consistent schedulers.

Theorem 2. *For all-in-one MDP $M^{\mathfrak{D}}$ and specification φ, there exists a memoryless deterministic scheduler $\sigma^r \in \Sigma^{M^{\mathfrak{D}}}$ such that $M^{\mathfrak{D}}_{\sigma^r} \vDash \varphi$ iff there exists a consistent deterministic scheduler $\sigma^{r^*} \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ such that $\left(M^{\mathfrak{D}}_{\sim}\right)_{\sigma^{r^*}} \vDash \varphi$.*

Example 4. Recall the all-in-one MDP $M^{\mathfrak{D}}$ from Example 3. The quotient MDP $M^{\mathfrak{D}}_{\sim}$ is depicted in Fig. 3. Only the transitions according to realisations r_1 and r_2 are included. Transitions from previously unreachable states, marked grey in Example 3, are now available due to the abstraction. The scheduler $\sigma \in \Sigma^{M^{\mathfrak{D}}_{\sim}}$ with $\sigma([s^{\mathfrak{D}}_0]_{\sim}) = a_{r_2}$ and $\sigma([1]_{\sim}) = a_{r_1}$ is *not* k_1*-consistent* as different values are chosen for k_1 by r_1 and r_2. In the MC $M^{\mathfrak{D}}_{\sim\sigma}$ induced by σ and $M^{\mathfrak{D}}_{\sim}$, the probability to reach state $[2]_{\sim}$ is one, while under realisation r_1, state 2 is not reachable.

Approach 3 (Scheduler iteration). *Enumerating all consistent schedulers for $M^{\mathfrak{D}}_{\sim}$ and analysing the induced MC provides a solution to both synthesis problems.*

However, optimising over exponentially many consistent schedulers solves the NP-complete feasibility synthesis problem, rendering such an iterative approach unlikely to be efficient. Another natural approach is to employ solving techniques for NP-complete problems, like satisfiability modulo linear real arithmetic.

Approach 4 (SMT). *A dedicated SMT-encoding (in [11]) of the induced MCs of consistent schedulers from $M^{\mathfrak{D}}_{\sim}$ that solves the feasibility problem.*

4.3 Refinement Loop

Although iterating over consistent schedulers (Approach 3) is not feasible, model checking of $M^{\mathfrak{D}}_{\sim}$ still provides useful information for the analysis of the family $\mathfrak{D}$. Recall the feasibility synthesis problem for $\varphi = \mathbb{P}_{\leq\lambda}(\phi)$. If $\texttt{Prob}^{\max}(M^{\mathfrak{D}}_{\sim}, \phi) \leq \lambda$, then all realisations of $\mathfrak{D}$ satisfy φ. On the other hand, $\texttt{Prob}^{\min}(M^{\mathfrak{D}}_{\sim}, \phi) > \lambda$ implies that there is no realisation satisfying φ. If λ lies between the min and max probability, and the scheduler inducing the min probability is not consistent, we cannot conclude anything yet, i.e., the abstraction is too coarse. A natural countermeasure is to refine the abstraction represented by $M^{\mathfrak{D}}_{\sim}$, in particular, split the set of realisations leading to two synthesis sub-problems.

Definition 11 (Splitting). *Let $\mathfrak{D}$ be a family of MCs, and $\mathcal{R} \subseteq \mathcal{R}^{\mathfrak{D}}$ a set of realisations. For $k \in K$ and predicate A_k over S,* splitting *partitions $\mathcal{R}$ into*

$$\mathcal{R}_{\top} = \{r \in \mathcal{R} \mid A_k(r(k))\} \quad \textit{and} \quad \mathcal{R}_{\bot} = \{r \in \mathcal{R} \mid \neg A_k(r(k))\}.$$

Splitting the set of realisations, and considering the subfamilies separately, rather than splitting states in the quotient MDP, is crucial for the performance of the synthesis process as we avoid rebuilding the quotient MDP in each iteration. Instead, we only restrict the actions of the MDP to the particular subfamily.

Definition 12 (Restricting). *Let $M^{\mathfrak{D}}_{\sim} = (S^{\mathfrak{D}}_{\sim}, [s^{\mathfrak{D}}_0]_{\sim}, Act^{\mathfrak{D}}, \mathcal{P}^{\mathfrak{D}}_{\sim})$ be a quotient MDP and $\mathcal{R} \subseteq \mathcal{R}^{\mathfrak{D}}$ a set of realisations. The* restriction *of $M^{\mathfrak{D}}_{\sim}$ wrt. $\mathcal{R}$ is the MDP $M^{\mathfrak{D}}_{\sim}[\mathcal{R}] = (S^{\mathfrak{D}}_{\sim}, [s^{\mathfrak{D}}_0]_{\sim}, Act^{\mathfrak{D}}[\mathcal{R}], \mathcal{P}^{\mathfrak{D}}_{\sim})$ where $Act^{\mathfrak{D}}[\mathcal{R}] = \{a_r \mid r \in \mathcal{R}\}$.*[3]

[3] Naturally, $\mathcal{P}^{\mathfrak{D}}_{\sim}$ in $M^{\mathfrak{D}}_{\sim}[\mathcal{R}]$ is restricted to $Act^{\mathfrak{D}}[\mathcal{R}]$.

Algorithm 1. Threshold synthesis

Input: A family $\mathfrak{D}$ of MCs with the set $\mathcal{R}^{\mathfrak{D}}$ of realisations, and specification $\mathbb{P}_{\leq\lambda}(\phi)$
Output: A partition of $\mathcal{R}^{\mathfrak{D}}$ into subsets T and F according to Problem 1.

$F \leftarrow \emptyset,\ T \leftarrow \emptyset,\ U \leftarrow \{\mathcal{R}^{\mathfrak{D}}\}$
$M^{\mathfrak{D}}_{\sim} \leftarrow \texttt{buildQuotientMDP}(\mathfrak{D}, \mathcal{R}^{\mathfrak{D}}, \sim_f)$ ▷ Applying Def. 7 and 8
while $U \neq \emptyset$ **do**
  select $\mathcal{R} \in U$ **and** $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathcal{R}\}$
  $M^{\mathfrak{D}}_{\sim}[\mathcal{R}] \leftarrow \texttt{restrict}(M^{\mathfrak{D}}_{\sim}, \mathcal{R})$ ▷ Applying Def. 12
  $(\max, \sigma_{\max}) \leftarrow \texttt{solveMaxMDP}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)$
  $(\min, \sigma_{\min}) \leftarrow \texttt{solveMinMDP}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)$
  if $\max < \lambda$ **then** $T \leftarrow T \cup \mathcal{R}$
  if $\min > \lambda$ **then** $F \leftarrow F \cup \mathcal{R}$
  if $\min \leq \lambda \leq \max$ **then**
    $U \leftarrow U \cup \texttt{split}(\mathcal{R}, \texttt{selPredicate}(\max, \sigma_{\max}, \min, \sigma_{\min}))$ ▷ See Sect. 4.4
return T, F

The splitting operation is the core of the proposed abstraction-refinement. Due to space constraints, we do not consider feasibility separately.

Algorithm 1 illustrates the *threshold synthesis* process. Recall that the goal is to decompose the set $\mathcal{R}^{\mathfrak{D}}$ into realisations satisfying and violating a given specification, respectively. The algorithm uses a set U to store subfamilies of $\mathcal{R}^{\mathfrak{D}}$ that have not been yet classified as satisfying or violating. It starts building the quotient MDP with merged actions. That is, we never construct the all-in-one MDP, and we merge actions as discussed in Remark 1. For every $\mathcal{R} \in U$, the algorithm restricts the set of realisations to obtain the corresponding subfamily. For the restricted quotient MDP, the algorithm runs standard MDP model checking to compute the max and min probability and corresponding schedulers, respectively. Then, the algorithm either classifies $\mathcal{R}$ as satisfying/violating, or splits it based on a suitable predicate, and updates U accordingly. We describe the splitting strategy in the next subsection. The algorithm terminates if U is empty, i.e., all subfamilies have been classified. As only a finite number of subfamilies of realisations has to be evaluated, termination is guaranteed.

The refinement loop for max synthesis is very similar, cf. Algorithm 2. Recall that now the goal is to find the realisation r^* that maximises the satisfaction probability $\max^*$ of a path formula. The difference between the algorithms lies in the interpretation of the results of the underlying MDP model checking. If the max probability for $\mathcal{R}$ is below $\max^*$, $\mathcal{R}$ can be discarded. Otherwise, we check whether the corresponding scheduler $\sigma_{\max}$ is consistent. If consistent, the algorithm updates r^* and $\max^*$, and discards $\mathcal{R}$. If the scheduler is not consistent but $\min > \max^*$ holds, we can still update $\max^*$ and improve the pruning process, as it means that some realisation (we do not know which) in $\mathcal{R}$ induces a higher probability than $\max^*$. Regardless whether $\max^*$ has been updated, the algorithm has to split $\mathcal{R}$ based on some predicate, and analyse its subfamilies as they may include the maximising realisation.

Algorithm 2. Max synthesis

Input: A family $\mathfrak{D}$ of MCs with the set $\mathcal{R}^{\mathfrak{D}}$ of realisations, and a path formula ϕ
Output: A realisation $r^* \in \mathcal{R}^{\mathfrak{D}}$ according to Problem 2.

1: $\max^* \leftarrow -\infty$, $U \leftarrow \{\mathcal{R}^{\mathfrak{D}}\}$
2: $M^{\mathfrak{D}}_{\sim} \leftarrow \texttt{buildQuotientMDP}(\mathfrak{D}, \mathcal{R}^{\mathfrak{D}}, \sim_f)$ ▷ Applying Def. 7 and 8
3: **while** $U \neq \emptyset$ **do**
4: **select** $\mathcal{R} \in U$ **and** $\mathcal{U} \leftarrow \mathcal{U} \setminus \{\mathcal{R}\}$
5: $M^{\mathfrak{D}}_{\sim}[\mathcal{R}] \leftarrow \texttt{restrict}(M^{\mathfrak{D}}_{\sim}, \mathcal{R})$ ▷ Applying Def. 12
6: $(\max, \sigma_{\max}) \leftarrow \texttt{solveMaxMDP}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)$
7: $(\min, \sigma_{\min}) \leftarrow \texttt{solveMinMDP}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)$
8: **if** $\max > \max^*$ **then**
9: **if** $\texttt{isConsistent}(\sigma_{\max})$ **then** $r^* \leftarrow q_{\max}, \max^* \leftarrow \max$
10: **else**
11: **if** $\min > \max^*$ **then** $\max^* \leftarrow \min$
12: $U \leftarrow U \cup \texttt{split}(\mathcal{R}, \texttt{selPredicate}(\max, \sigma_{\max}, \min, \sigma_{\min}))$ ▷ See Sect. 4.4
13: **return** r^*

4.4 Splitting Strategies

If verifying the quotient MDP $M^{\mathfrak{D}}_{\sim}[\mathcal{R}]$ cannot classify the (sub-)realisation $\mathcal{R}$ as satisfying or violating, we split $\mathcal{R}$, while we guide the splitting strategy by using the obtained verification results. The splitting operation chooses a suitable parameter $k \in K$ and predicate A_k that partition the realisations $\mathcal{R}$ into $\mathcal{R}_\top$ and $\mathcal{R}_\bot$ (see Definition 11). A good splitting strategy globally reduces the number of model-checking calls required to classify all $r \in \mathcal{R}$.

The two key aspects to locally determine a good k are: (1) the *variance*, that is, how the splitting may narrow the difference between $\max = \texttt{Prob}^{\max}(M^{\mathfrak{D}}_{\sim}[\mathcal{X}], \phi)$ and $\min = \texttt{Prob}^{\min}(M^{\mathfrak{D}}_{\sim}[\mathcal{X}], \phi)$ for both $\mathcal{X} = \mathcal{R}_\top$ or $\mathcal{X} = \mathcal{R}_\bot$, and (2) the *consistency*, that is, how the splitting may reduce the inconsistency of the schedulers $\sigma_{\max}$ and $\sigma_{\min}$. These aspects cannot be evaluated precisely without applying all the split operations and solving the new MDPs $M^{\mathfrak{D}}_{\sim}[\mathcal{R}_\bot]$ and $M^{\mathfrak{D}}_{\sim}[\mathcal{R}_\top]$. Therefore, we propose an efficient strategy that selects k and A_k based on a light-weighted analysis of the model-checking results for $M^{\mathfrak{D}}_{\sim}[\mathcal{R}]$. The strategy applies two *scores* $\texttt{variance}(k)$ and $\texttt{consistency}(k)$ that estimate the influence of k on the two key aspects. For any k, the scores are accumulated over all *important states* s (reachable via $\sigma_{\max}$ or $\sigma_{\min}$, respectively) where $\mathfrak{P}(s)(k) \neq 0$. A state s is important for $\mathcal{R}$ and some $\delta \in \mathbb{R}_{\geq 0}$ if

$$\frac{\texttt{Prob}^{\max}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)(s) - \texttt{Prob}^{\min}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)(s)}{\texttt{Prob}^{\max}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi) - \texttt{Prob}^{\min}(M^{\mathfrak{D}}_{\sim}[\mathcal{R}], \phi)} \geq \delta$$

where $\texttt{Prob}^{\min}(.)(s)$ and $\texttt{Prob}^{\max}(.)(s)$ is the min and max probability in the MDP with initial state s. To reduce the overhead of computing the scores, we simplify the scheduler representation. In particular, for $\sigma_{\max}$ and every $k \in K$, we extract a map $C^k_{\max} \colon T_k \to \mathbb{N}$, where $C^k_{\max}(t)$ is the number of important states for which $\sigma_{\max}(s) = a_r$ with $r(k) = t$. The mapping $C^k_{\min}$ represents $\sigma_{\min}$.

We define $\texttt{variance}(k) = \sum_{t \in T_k} |C^k_{\max}(t) - C^k_{\min}(t)|$, leading to high scores if the two schedulers vary a lot. Further, we define $\texttt{consistency}(k) = \texttt{size}\left(C^k_{\max}\right) \cdot \texttt{max}\left(C^k_{\max}\right) + \texttt{size}\left(C^k_{\min}\right) \cdot \texttt{max}\left(C^k_{\min}\right)$, where $\texttt{size}\,(C) = |\{t \in T_k \mid C(t) > 0\}| - 1$ and $\texttt{max}\,(C) = \max_{t \in T_k}\{C(t)\}$, leading to high scores if the parameter has clear favourites for $\sigma_{\max}$ and $\sigma_{\min}$, but values from its full range are chosen.

As indicated, we consider different strategies for the two synthesis problems. For threshold synthesis, we favour the impact on the variance as we principally do not need consistent schedulers. For the max synthesis, we favour the impact on the consistency, as we need a consistent scheduler inducing the max probability.

Predicate A_k is based on reducing the variance: The strategy selects $T' \subset T_k$ with $|T'| = \frac{1}{2}\lceil |T_k| \rceil$, containing those t for which $C^k_{\max}(t) - C^k_{\min}(t)$ is the largest. The goal is to get a set of realisations that induce a large probability (the ones including T' for parameter k) and the complement inducing a small probability.

Approach 5 (MDP-based abstraction refinement). *The methods underlying Algorithms 1 and 2, together with the splitting strategies, provide solutions to the synthesis problems and are referred to as* MDP abstraction *methods.*

5 Experiments

We implemented the proposed synthesis methods as a Python prototype using Storm [25]. In particular, we use the Storm Python API for model-adaption, -building, and -checking as well as for scheduler extraction. For SMT solving, we use Z3 [39] via pySMT [26]. The tool-chain takes a PRISM [38] or JANI [8] model with open integer constants, together with a set of expressions with possible values for these constants. The model may include the parallel composition of several modules/automata. The open constants may occur in guards[4], probability definitions, and updates of the commands/edges. Via adequate annotations, we identify the parameter values that yield a particular action. The annotations are key to interpret the schedulers, and to restrict the quotient without rebuilding.

All experiments were executed on a Macbook MF839LL/A with 8 GB RAM memory limit and a 12 h time out. All algorithms can significantly benefit from coarse-grained parallelisation, which we therefore do not consider here.

5.1 Research Questions and Benchmarks

The goal of the experimental evaluation is to answer the research question: *How does the proposed MDP-based abstraction methods (Approaches 3–5) cope with the inherent complexity (i.e. the NP-hardness) of the synthesis problems (cf. Problems 1 and 2)?* To answer this question, we compare their performance with Approaches 1 and 2 [18], representing state-of-the-art solutions and the base-line algorithms. The experiments show that the performance of the

[4] Slight care by the user is necessary to avoid deadlocks.

Table 1. Benchmarks and timings for Approaches 1–3

Bench.	Range	$\|K\|$	$\|\mathcal{D}\|$	Member size		Quotient size			Run time		
				Avg. $\|S\|$	Avg. $\|T\|$	$\|S\|$	$\|A\|$	$\|T\|$	1-by-1	All-in-1	Sched. Enum.
Pole	[3.35, 3.82]	17	1327104	5689	16896	6793	7897	22416	130k*	MO	26k
Maze	[9.8, 9800]	20	1048576	134	211	203	277	409	28k*	TO	2.7k
Herman	[1.86, 2.44]	9	576	5287	6948	21313	102657	184096	55*	72	246
DPM	[68, 210]	9	32768	5572	18147	35154	66096	160146	2.9k*	MO	7.2k
BSN	[0, 0.988]	10	1024	116	196	382	457	762	31*	2	2

MDP abstraction significantly varies for different case studies. Thus, we consider benchmarks from various application domains to *identify the key characteristics of the synthesis problems affecting the performance of our approach.*

Benchmarks description. We consider the following case studies: *Maze* is a planning problem typically considered as POMDP, e.g. in [41]. The family describes all MCs induced by small-memory [14,35] observation-based deterministic strategies (with a fixed upper bound on the memory). We are interested in the expected time to the goal. In [35], parameter synthesis was used to find randomised strategies, using [22]. *Pole* considers balancing a pole in a noisy and unknown environment (motivated by [2,12]). At deploy time, the controller has a prior over a finite set of environment behaviours, and should optimise the expected behavior without depending on the actual (hidden) environment. The family describes schedulers that do not depend on the hidden information. We are interested in the expected time until failure. *Herman* is an asynchronous encoding of the distributed Herman protocol for self-stabilising rings [33,37]. The protocol is extended with a bit of memory for each station in the ring, and the choice to flip various unfair coins. Nodes in the ring are anonymous, they all behave equivalently (but may change their local memory based on local events). The family describes variations of memory-updates and coin-selection, but preserves anonymity. We are interested in the expected time until stabilisation. *DPM* considers a partial information scheduler for a disk power manager motivated by [7,27]. We are interested in the expected energy consumption. *BSN* (Body sensor network, [43]) describes a network of connected sensors that identify health-critical situations. We are interested in the reliability. The family contains various configurations of the used sensors. *BSN* is the largest software product line benchmark used in [18]. We drop some implications between features (parameters for us) as this is not yet supported by our modelling language. We thereby extended the family.

Table 1 shows the relevant statistics for each benchmark: the benchmark name, the (approximate) range of the min and max probability/reward for the given family, the number of non-singleton parameters $|K|$, and the number of family members $|\mathfrak{D}|$. Then, for the family members the average number of states and transitions of the MCs, and the states, actions ($= \sum_{s \in S} |Act(s)|$), and transitions of the quotient MDP. Finally, it lists in seconds the run time of the base-line

Table 2. Results for threshold synthesis via abstraction-refinement

Inst	λ	# Below	# Subf below	# Above	# Subf above	Singles	# Iter	Time	Build	Check	Anal.	Speedup
Pole	3.37	697	176	1326407	2186	920	4723	308	117	60	118	**421**
	3.73	1307077	7854	20027	3279	1294	22265	1.7k	576	317	396	**77**
	3.76	1322181	3140	4923	1025	1022	8329	584	187	114	197	**222**
	3.79	1326502	572	602	123	74	1389	58	23	10	23	**2.2k**
Maze	10	4	3	1048572	92	4	189	5	<1	3	<1	**26k**
	20	4247	2297	1044329	4637	3400	13867	114	21	43	29	**246**
	30	18188	9934	1030388	18004	14010	55875	608	80	127	270	**46**
	8000	1046285	846	2291	1125	969	3941	136	9	106	13	**1.0k**
Herman	1.9	6	6	570	368	320	747	333	303	11	18	**0.2**
	1.71	0	0	576	258	184	515	232	206	8	17	**0.3**
DPM	80	160	141	32608	1292	356	2865	1.0k	602	322	64	**3**
	70	6	6	32762	443	40	897	380	190	156	32	**8**
	60	0	0	32768	104	6	207	99	42	48	8	**29**
BSN	.965	544	81	480	81	25	321	2	<1	<1	<1	**1**
	.985	994	41	30	8	5	97	<1	<1	<1	<1	**3**

algorithms and the consistent scheduler enumeration[5]. The base-line algorithms employ the one-by-one and the all-in-one technique, using either a BDD or a sparse matrix representation. We report the best results. MOs indicate breaking the memory limit. Only the all-in-one approach required significant memory. As expected, the SMT-based implementation provides an inferior performance and thus we do not report its results.

5.2 Results and Discussion

To simplify the presentation, we focus primarily on the threshold synthesis problem as it allows a compact presentation of the key aspects. Below, we provide some remarks about the performance for the max and feasibility synthesis.

Results. Table 2 shows results for threshold synthesis. The first two columns indicate the benchmark and the various thresholds. For each threshold λ, the table lists the number of family members below (above) λ, each with the number of subfamilies that together contain these instances, and the number of singleton subfamilies that were considered. The last table part gives the number of iterations of the loop in Algorithm 1, and timing information (total, build/restrict times, model checking times, scheduler analysis times). The last column gives the speed-up over the best base-line (based on the estimates).

Key observations. The speed-ups drastically vary, which shows that the MDP abstraction often achieves a superior performance but may also lead to a performance degradation in some cases. We identify four key factors.

[5] Values with a * are estimated by sampling a large fraction of the family.

Iterations. As typical for CEGAR approaches, the key characteristic of the benchmark that affects the performance is the number N of iterations in the refinement loop. The abstract action introduces an overhead per iteration caused by performing two MDP verification calls and by the scheduler analysis. The run time for *BSN*, with a small $|\mathfrak{D}|$ is actually significantly affected by the initialisation of various data structures; thus only a small speedup is achieved.

Abstraction size. The size of the quotient, compared to the average size of the family members, is relevant too. The quotient includes at least all reachable states of all family members, and may be significantly larger if an inconsistent scheduler reaches states which are unreachable under any consistent scheduler. The existence of such states is a common artefact from encoding families in high-level languages. Table 1, however, indicates that we obtain a very compact representation for *Maze* and *Pole*.

Thresholds. The most important aspect is the threshold λ. If λ is closer to the optima, the abstraction requires a smaller number of iterations, which directly improves the performance. We emphasise that in various domains, thresholds that ask for close-to-optimal solutions are indeed of highest relevance as they typically represent the system designs developers are most interested in [44]. *Why do thresholds affect the number of iterations?* Consider a family with $T_k = \{0, 1\}$ for each k. Geometrically, the set $\mathcal{R}^{\mathfrak{D}}$ can be visualised as $|K|$-dimensional cube. The cube-vertices reflect family members. Assume for simplicity that one of these vertices is optimal with respect to the specification. Especially in benchmarks where parameters are equally important, the induced probability of a vertex roughly corresponds to the Manhattan distance to the optimal vertex. Thus, vertices above the threshold induce a diagonal hyperplane, which our splitting method approximates with orthogonal splits. Splitting diagonally is not possible, as it would induce optimising over observation-based schedulers. Consequently, we need more and more splits the more the diagonal goes through the middle of the cube. *Even when splitting optimally, there is a combinatorial blow-up in the required splits when the threshold is further from the optimal values.* Another effect is that thresholds far from optima are more affected by the over-approximation of the MDP model-checking results and thus yield more inconclusive answers.

Refinement strategy. So far, we reasoned about optimal splits. Due to the computational overhead, our strategy cannot ensure optimal splits. Instead, the strategy depends mostly on information encoded in the computed MDP strategies. *In models where the optimal parameter value heavily depends on the state, the obtained schedulers are highly inconsistent and carry only limited information for splitting.* Consequently, in such benchmarks we split sub-optimally. The sub-optimality has a major impact on the performance for *Herman* as all obtained strategies are highly inconsistent – they take a different coin for each node, which is good to speed up the stabilisation of the ring.

Summary. MDP abstraction is not a silver bullet. It has a lot of potential in threshold synthesis when the threshold is close to the optima. Consequently,

feasibility synthesis with unsatisfiable specifications is handled perfectly well by MDP abstraction, while this is the worst-case for enumeration-based approaches. Likewise, *max synthesis* can be understood as threshold synthesis with a shifting threshold $\max^*$: If the $\max^*$ is quickly set close to max, MDP abstraction yields superior performance. Roughly, we can quickly approximate $\max^*$ when some of the parameter values are clearly beneficial for the specification.

6 Conclusion and Future Work

We contributed to the efficient analysis of families of Markov chains. In particular, we discussed and implemented existing approaches to solve practically interesting synthesis problems, and devised a novel abstraction refinement scheme that mitigates the computational complexity of the synthesis problems, as shown by the empirical evaluation. In the future, we will include refinement strategies based on counterexamples as in [23,34].

References

1. Repository with benchmarks. https://github.com/moves-rwth/shepherd
2. Arming, S., Bartocci, E., Chatterjee, K., Katoen, J.-P., Sokolova, A.: Parameter-independent strategies for pMDPs via POMDPs. In: McIver, A., Horvath, A. (eds.) QEST 2018. LNCS, vol. 11024, pp. 53–70. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99154-2_4
3. Aziz, A., Singhal, V., Balarin, F., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: It usually works: the temporal logic of stochastic systems. In: Wolper, P. (ed.) CAV 1995. LNCS, vol. 939, pp. 155–165. Springer, Heidelberg (1995). https://doi.org/10.1007/3-540-60045-0_48
4. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. In: Clarke, E., Henzinger, T., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 963–999. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_28
5. Baier, C., Katoen, J.: Principles of Model Checking. MIT Press, Cambridge (2008)
6. 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
7. Benini, L., Bogliolo, A., Paleologo, G., Micheli, G.D.: Policy optimization for dynamic power management. IEEE Trans. Comput.-Aided Des. Integr. Circ. Syst. **8**(3), 299–316 (2000)
8. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_9
9. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. J. Syst. Softw. **143**, 140–158 (2018)

10. Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., Brim, L.: Precise parameter synthesis for stochastic biochemical systems. Acta Informatica **54**(6), 589–623 (2017)
11. Češka, M., Jansen, N., Junges, S., Katoen, J.P.: Shepherding hordes of Markov chains. CoRR abs/1902.xxxxx (2019)
12. Chades, I., Carwardine, J., Martin, T.G., Nicol, S., Sabbadin, R., Buffet, O.: MOMDPs: a solution for modelling adaptive management problems. In: AAAI. AAAI Press (2012)
13. Chasins, S., Phothilimthana, P.M.: Data-driven synthesis of full probabilistic programs. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 279–304. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63387-9_14
14. Chatterjee, K., Chmelik, M., Davies, J.: A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In: AAAI, pp. 3225–3232. AAAI Press (2016)
15. Chatterjee, K., Kößler, A., Schmid, U.: Automated analysis of real-time scheduling using graph games. In: HSCC, pp. 163–172. ACM (2013)
16. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M.Z., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: TASE, pp. 85–92. IEEE (2013)
17. Chonev, V.: Reachability in augmented interval Markov chains. CoRR abs/1701.02996 (2017)
18. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Formal Asp. Comput. **30**(1), 45–75 (2018)
19. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.: Model checking software product lines with SNIP. STTT **14**(5), 589–612 (2012)
20. Classen, A., Cordy, M., Heymans, P., Legay, A., Schobbens, P.: Formal semantics, modular specification, and symbolic verification of product-line behaviour. Sci. Comput. Program. **80**, 416–439 (2014)
21. Cordy, M., Heymans, P., Legay, A., Schobbens, P.Y., Dawagne, B., Leucker, M.: Counterexample guided abstraction refinement of product-line behavioural models. In: SIGSOFT FSE, pp. 190–201. ACM (2014)
22. Cubuktepe, M., Jansen, N., Junges, S., Katoen, J.-P., Topcu, U.: Synthesis in pMDPs: a tale of 1001 parameters. In: Lahiri, S.K., Wang, C. (eds.) ATVA 2018. LNCS, vol. 11138, pp. 160–176. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-01090-4_10
23. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.-P.: Fast debugging of PRISM models. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 146–162. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_11
24. 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
25. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
26. Gario, M., Micheli, A.: PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms. In: SMT Workshop 2015 (2015)
27. Gerasimou, S., Calinescu, R., Tamburrelli, G.: Synthesis of probabilistic models for quality-of-service software engineering. Autom. Softw. Eng. **25**(4), 785–831 (2018)

28. Ghezzi, C., Sharifloo, A.M.: Model-based verification of quantitative non-functional properties for software product lines. Inf. Softw. Technol. **55**(3), 508–524 (2013)
29. Giro, S., D'Argenio, P.R., Fioriti, L.M.F.: Distributed probabilistic input/output automata: expressiveness, (un)decidability and algorithms. Theor. Comput. Sci. **538**, 84–102 (2014)
30. Giro, S., Rabe, M.N.: Verification of partial-information probabilistic systems using counterexample-guided refinements. In: Chakraborty, S., Mukund, M. (eds.) ATVA 2012. LNCS, vol. 7561, pp. 333–348. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33386-6_26
31. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. Softw. Tools Technol. Transfer **13**(1), 3–19 (2011)
32. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**(5), 512–535 (1994)
33. Herman, T.: Probabilistic self-stabilization. Inf. Process. Lett. **35**(2), 63–67 (1990)
34. Jansen, N., et al.: Symbolic counterexample generation for large discrete-time Markov chains. Sci. Comput. Program. **91**, 90–114 (2014)
35. Junges, S., et al.: Finite-state controllers of POMDPs using parameter synthesis. In: UAI, pp. 519–529. AUAI Press (2018)
36. Kochenderfer, M.J.: Decision Making Under Uncertainty: Theory and Application, 1st edn. The MIT Press, Cambridge (2015)
37. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic verification of Herman's self-stabilisation algorithm. Formal Aspects Comput. **24**(4), 661–670 (2012)
38. 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
39. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
40. Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Efficient synthesis of probabilistic programs. In: PLDI, pp. 208–217. ACM (2015)
41. Norman, G., Parker, D., Zou, X.: Verification and control of partially observable probabilistic systems. Real-Time Syst. **53**(3), 354–402 (2017)
42. 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
43. Rodrigues, G.N., et al.: Modeling and verification for probabilistic properties in software product lines. In: HASE, pp. 173–180. IEEE (2015)
44. Skaf, J., Boyd, S.: Techniques for exploring the suboptimal set. Optim. Eng. **11**(2), 319–337 (2010)
45. Vandin, A., ter Beek, M.H., Legay, A., Lluch-Lafuente, A.: QFLan: a tool for the quantitative analysis of highly reconfigurable systems. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 329–337. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_19
46. Varshosaz, M., Khosravi, R.: Discrete time Markov chain families: modeling and verification of probabilistic software product lines. In: SPLC Workshops, pp. 34–41. ACM (2013)

Counterexample-Driven Synthesis for Probabilistic Program Sketches

Milan Češka[1], Christian Hensel[2], Sebastian Junges[2( )], and Joost-Pieter Katoen[2]

[1] Brno University of Technology, FIT, IT4I Centre of Excellence, Brno, Czech Republic
[2] RWTH Aachen University, Aachen, Germany
sebastian.junges@cs.rwth-aachen.de

Abstract. Probabilistic programs are key to deal with uncertainty in, e.g., controller synthesis. They are typically small but intricate. Their development is complex and error prone requiring quantitative reasoning over a myriad of alternative designs. To mitigate this complexity, we adopt counterexample-guided inductive synthesis (CEGIS) to automatically synthesise finite-state probabilistic programs. Our approach leverages efficient model checking, modern SMT solving, and counterexample generation at program level. Experiments on practically relevant case studies show that design spaces with millions of candidate designs can be fully explored using a few thousand verification queries.

1 Introduction

With the ever tighter integration of computing systems with their environment, quantifying (and minimising) the probability of encountering an anomaly or unexpected behaviour becomes crucial. This insight has led to a growing interest in probabilistic programs and models in the software engineering community. Henzinger [43] for instance argues that "the Boolean partition of software into correct and incorrect programs falls short of the practical need to assess the behaviour of software in a more nuanced fashion [...]." In [60], Rosenblum advocates taking a more probabilistic approach in software engineering. Concrete examples include quantitative analysis of software product lines [32,40,59,66,67], synthesis of probabilities for adaptive software [19,23], and probabilistic model checking at runtime to support verifying dynamic reconfigurations [20,37].

Synthesis of Probabilistic Programs. Probabilistic programs are a prominent formalism to deal with uncertainty. Unfortunately, such programs are rather intricate. Their development is complex and error prone requiring quantitative reasoning over many alternative designs. One remedy is the exploitation of probabilistic model checking [6] using a *Markov chain* as the operational model of a

This work has been supported by the Czech Science Foundation grant No. GA19-24397S, the IT4Innovations excellence in science project No. LQ1602, the DFG RTG 2236 "UnRAVeL", and the ERC Advanced Grant 787914 "FRAPPANT",.

M. H. ter Beek et al. (Eds.): FM 2019, LNCS 11800, pp. 101–120, 2019.
https://doi.org/10.1007/978-3-030-30942-8_8

program. One may then apply model checking on each design, or some suitable representation thereof [27,32]. Techniques such as parameter synthesis [26,42,58] and model repair [9,31] have been successful, but they only allow to amend or infer transition probabilities, whereas the control structure—the topology of the probabilistic model—is fixed.

Counter-Example-Guided Inductive Synthesis. This paper aims to overcome the existing limitation, by adopting the paradigm of *CounterExample-Guided Inductive Synthesis* (CEGIS, cf. Fig. 1) [1, 3,63,64] to finite-state probabilistic models and programs. The program synthesis challenge is to automatically provide a probabilistic program satisfying all properties, or to return that such a program is non-existing. In the syntax-based setting, we start with a sketch, a program with holes, and iteratively searches for good—or even optimal—instantiations of these holes. Rather than checking all instantiations, the design space is pruned by potentially ruling out many instantiations (dashed area) at once. From every realisation that was verified and rejected, a counterexample (CE) is derived, e.g., a program run violating the specification. An SMT (satisfiability modulo theory)-based synthesiser uses the CE to prune programs that also violate the specification. These programs are safely removed from the design space. The synthesis and verification step are repeated until either a satisfying program is found or the entire design space is pruned implying the non-existence of such a program.

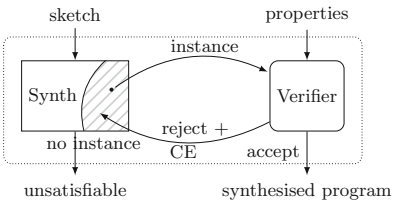


Fig. 1. CEGIS for synthesis.

Problem Statement and Program-Level Approach. This paper tailors and generalises CEGIS to probabilistic models and programs. The input is a sketch—a probabilistic program with holes, where each hole can be replaced by finitely many options—, a set of quantitative properties that the program needs to fulfil, and a budget. All possible realisations have a certain cost and the synthesis provides a realisation that fits within the budget. Programs are represented in the PRISM modelling language [50] and properties are expressed in PCTL (Probabilistic Computational Tree Logic) extended with rewards, as standard in probabilistic model checking [34,50]. Program sketches succinctly describe the design space of the system by providing the program-level structure but leaving some parts (e.g., command guards or variable assignments) unspecified.

Outcomes. To summarise, this paper presents a novel synthesis framework for probabilistic programs that adhere to a given set of quantitative requirements and a given budget. We use families of Markov chains to formalise our problem, and then formulate a CEGIS-style algorithm on these families. Here, CEs are subgraphs of the Markov chains. In the second part, we then generalise the approach to reason on probabilistic programs with holes. While similar in spirit,

we rely on program-level CEs [33,71], and allow for a more flexible sketching language. To the best of our knowledge, this is the first lifting of CEGIS to probabilistic programs. The CEGIS approach is sound and complete: either an admissible program does exist and it is computed, or no such program exists and the algorithm reports this. We provide a prototype implementation built on top of the model checker Storm [34] and the SMT-tool Z3 [56]. Experiments with different examples demonstrate scalability: design spaces with millions of realisations can be fully explored by a few thousand verification queries and result in a speedup of orders of magnitude.

Related Work. We build on the significant body of research that employs formal methods to analyse quality attributes of alternative designs, e.g. [8,10, 16,38,65,72]. Enumerative approaches based on Petri nets [54], stochastic models [19,61] and timed automata [44,52], and the corresponding tools for simulation and verification (e.g. Palladio [10], PRISM [50], UPPAAL [44]) have long been used.

For non-probabilistic systems, CEGIS can find programs for a variety of challenging problems [62,63]. Meta-sketches and the *optimal and quantitative synthesis problem* in a non-probabilistic setting have been proposed [17,25,30].

A prominent representation of sets of alternative designs are modal transition systems [5,49,53]. In particular, *parametric* modal transition systems [11] and synthesis therein [12] allow for similar dependencies that occur in program-level sketches. Probabilistic extensions are considered in, e.g. [35], but not in conjunction with synthesis. Recently [36] proposed to exploit relationships between model and specification, thereby reducing the number of model-checking instances. In the domain of quantitative reasoning, sketches and likelihood computation are used to find probabilistic programs that best match available data [57]. The work closest to our approach synthesises probabilistic systems from specifications and parametric templates [39]. The principal difference to our approach is the use of counterexamples. The authors leverage evolutionary optimisation techniques without pruning. Therefore, completeness is only achieved by exploring all designs, which is practically infeasible. An extension to handle parameters affecting transition probabilities (rates) has been integrated into the evolutionary-driven synthesis [21,23] and is available in RODES [22]. Some papers have considered the analysis of sets of alternative designs within the quantitative verification of software product lines [40,59,67]. The typical approach is to analyse all individual designs (product configurations) or build and analyse a single (so-called *all-in-one*) Markov decision process describing all the designs simultaneously. Even with symbolic methods, this hardly scales to large sets of alternative designs. These techniques have recently been integrated into ProFeat [32] and QFLan [66]. An abstraction-refinement scheme has recently been explored in [27]. It iteratively analyses an abstraction of a (sub)set of designs—it is an orthogonal and slightly restricted approach to the inductive method presented here (detailed differences are discussed later). An incomplete method in [45] employs abstraction targeting a particular case study. SMT-based encodings for synthesis in Markov models have been used in, e.g. [24,46]. These

encodings are typically monolithic—they do not prune the search space via CEs. Probabilistic CEs have been recently used to ensure that controllers obtained via learning from positive examples meet given safety properties [74]. In contrast, we leverage program-level CEs that can be used to prune the design space.

2 Preliminaries and Problem Statement

We start with basics of probabilistic model checking, for details, see [6,7], and then formalise families of Markov chains. Finally, we define some synthesis problems.

Probabilistic Models and Specifications. A *probability distribution* over a finite set X is a function $\mu\colon X \to [0,1]$ with $\sum_{x\in X} \mu(X) = 1$. Let $Distr(X)$ denote the set of all distributions on X.

Definition 1 (MC). *A* discrete-time Markov chain *(MC) D is a tuple (S, s_0, P) with finite set S of states, initial state $s_0 \in S$, and transition probabilities $P\colon S \to Distr(S)$. We write $P(s,t)$ to denote $P(s)(t)$.*

For $S' \subseteq S$, the set $\mathsf{Succ}(S') := \{t \in S \mid \exists s \in S'.\ P(s,t) > 0\}$ denotes the successor states of S'. A *path* of an MC D is an (in)finite sequence $\pi = s_0 s_1 s_2 \ldots$, where $s_i \in S$, and $s_{i+1} \in \mathsf{Succ}(s_i)$ for all $i \in \mathbb{N}$.

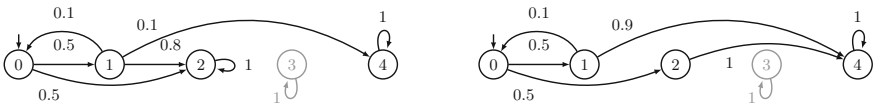
Definition 2 (sub-MC). *Let $D = (S, s_0, P)$ be an MC with* critical states *$C \subseteq S$, $s_0 \in C$. The* sub-MC *of D, C is the MC $D{\downarrow}C = (C \cup \mathsf{Succ}(C), s_0, P')$ with: $P'(s,t) = P(s,t)$ for $s \in C$, $P'(s,s) = 1$ for $s \in \mathsf{Succ}(C)\backslash C$, and $P'(s,t) = 0$ otherwise.*

Specifications. For simplicity, we focus on reachability properties $\varphi = \mathbb{P}_{\sim\lambda}(\Diamond G)$ for a set $G \subseteq S$ of goal states, threshold $\lambda \in [0,1] \subseteq \mathbb{R}$, and comparison relation $\sim\ \in \{<, \leq, \geq, >\}$. The interpretation of φ on MC D is as follows. Let $\mathtt{Prob}(D, \Diamond G)$ denote the probability to reach G from D's initial state. Then, $D \models \varphi$ if $\mathtt{Prob}(D, \Diamond G) \sim \lambda$. A specification is a set $\Phi = \{\varphi_i\}_{i\in I}$ of properties, and $D \models \Phi$ if $\forall i \in I.\ D \models \varphi_i$. Upper-bounded properties (with $\sim\ \in \{<, \leq\}$) are safety properties, and lower-bounded properties are liveness properties. Extensions to expected rewards are straightforward.

Families of Markov Chains. We recap an explicit representation of a *family of MCs* using a parametric transition function, as in [27].

Definition 3 (Family of MCs). *A* family of MCs *is a tuple $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ with S, s_0 as before, a finite set of parameters K where the domain for each parameter $k \in K$ is $T_k \subseteq S$, and transition probability function $\mathfrak{P}\colon S \to Distr(K)$.*

The transition probability function of MCs maps states to distributions over successor states. For families, this function maps states to distributions over parameters. Instantiating each parameter with a value from its domain yields a "concrete" MC, called a *realisation*.



(a) D_{r_1} with $r_1(k_2) = 2, r_1(k_3) = 2$ (b) D_{r_2} with $r_2(k_2) = 2, r_2(k_3) = 4$

(c) D_{r_3} with $r_3(k_2) = 3, r_3(k_3) = 2$ (d) D_{r_4} with $r_4(k_2) = 3, r_4(k_3) = 4$

Fig. 2. The four different realisations of family $\mathfrak{D}$.

Definition 4 (Realisation). *A* realisation *of a family* $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ *is a function* $r\colon K \to S$ *where* $\forall k \in K\colon r(k) \in T_k$. *A realisation* r *yields an MC* $D_r := (S, s_0, \mathfrak{P}(r))$, *where* $\mathfrak{P}(r)$ *is the transition probability matrix in which each* $k \in K$ *in* $\mathfrak{P}$ *is replaced by* $r(k)$. *Let* $\mathcal{R}^{\mathfrak{D}}$ *denote the* set of all realisations *for* $\mathfrak{D}$.

As a family $\mathfrak{D}$ has finite parameter domains, the number of family members (i.e. realisations from $\mathcal{R}^{\mathfrak{D}}$) of $\mathfrak{D}$ is finite, but exponential in $|K|$. While all MCs share their state space, their *reachable* states may differ.

Example 1. Consider the family of MCs $\mathfrak{D} = (S, s_0, K, \mathfrak{P})$ where $S = \{0, \ldots, 4\}$, $s_0 = 0$, and $K = \{k_0, \ldots, k_5\}$ with $T_{k_0} = \{0\}$, $T_{k_1} = \{1\}$, $T_{k_2} = \{2, 3\}$, $T_{k_3} = \{2, 4\}$, $T_{k_4} = \{3\}$ and $T_{k_5} = \{4\}$, and $\mathfrak{P}$ given by:

$$\mathfrak{P}(0) = 0.5\colon k_1 + 0.5\colon k_2 \quad \mathfrak{P}(1) = 0.1\colon k_0 + 0.8\colon k_3 + 0.1\colon k_5 \quad \mathfrak{P}(2) = 1\colon k_3$$
$$\mathfrak{P}(3) = 1\colon k_4 \qquad\qquad \mathfrak{P}(4) = 1\colon k_5$$

Figure 2 shows the four MCs of $\mathfrak{D}$. Unreachable states are greyed out.

The function $c\colon \mathcal{R}^{\mathfrak{D}} \to \mathbb{N}$ assigns *realisation costs*. Attaching costs to realisations is a natural way to distinguish preferable realisations. We stress the difference with rewards in MCs; the latter impose a cost structure on paths in MCs.

Problem Statement *Synthesis Problems.* Let $\mathfrak{D}$ be a family, and Φ be a set of properties, and $B \in \mathbb{N}$ a budget. Consider the synthesis problems:

1. *Feasibility synthesis:* Find a realisation $r \in \mathcal{R}^{\mathfrak{D}}$ with $D_r \models \Phi$ and $c(r) \leq B$.
2. *Max synthesis:* For given $G \subseteq S$, find $r^* \in \mathcal{R}^{\mathfrak{D}}$ with

$$r^* := \operatorname*{argmax}_{r \in \mathcal{R}_{\mathfrak{D}}} \{\texttt{Prob}(D_r, \Diamond G) \mid D_r \models \Phi \text{ and } c(r) \leq B\}.$$

The problem in feasibility synthesis is to determine a realisation satisfying all $\varphi \in \Phi$, or return that no such realisation exists. The problem in max synthesis is to find a realisation that maximises the reachability probability of reaching G. It can analogously be defined for minimising such probabilities. As families are finite, such optimal realisations r^* always exist (if there exists a feasible solution). It is beneficial to consider a variant of the max-synthesis problem in which the realisation r^* is not required to achieve the maximal reachability probability, but it suffices to be close to it. This notion of ε *-maximal synthesis* for a given $0 < \varepsilon \leq 1$ amounts to find a realisation r^* with $\texttt{Prob}(D_{r^*}, \Diamond G) \geq (1-\varepsilon) \cdot \max_{r \in \mathcal{R}^{\mathfrak{D}}} \{\texttt{Prob}(D_r, \phi)\}$.

Problem Statement and Structure. In this paper, we propose novel synthesis algorithms for the probabilistic systems that are based on two concepts, CEGIS [63] and syntax-guided synthesis [3]. To simplify the presentation, we start with CEGIS in Sect. 3 and adopt it for MCs and the feasibility problem. In Sect. 4, we lift and tune CEGIS, in particular towards probabilistic program sketches.

3 CEGIS for Markov Chain Families

We follow the typical separation of concerns as in oracle-guided inductive synthesis [4,39,41]: a *synthesiser* selects single realisations r that have not been considered before, and a *verifier* checks whether the MC D_r satisfies the specification Φ (cf. Fig. 1 on page 1). If a realisation violates the specification, the verifier returns a *conflict* representing the core part of the MC causing the violation.

3.1 Conflicts and Synthesiser

To formalise conflicts, a *partial realisation* of a family $\mathfrak{D}$ is a function $\bar{r}\colon K \to S \cup \{\bot\}$ such that $\forall k \in K.\ \bar{r}(k) \in T_k \cup \{\bot\}$. For any partial realisations $\bar{r}_1$, $\bar{r}_2$, let $\bar{r}_1 \subseteq \bar{r}_2$ iff $\bar{r}_1(k) \in \{\bar{r}_2(k), \bot\}$ for all $k \in K$.

Definition 5 (Conflict). *Let $r \in \mathcal{R}^{\mathfrak{D}}$ be a realisation with $D_r \not\models \varphi$ for $\varphi \in \Phi$. A partial realisation $\bar{r}_\varphi \subseteq r$ is a* conflict *for the property φ iff $D_{r'} \not\models \varphi$ for each realisation $r' \supseteq \bar{r}_\varphi$. A set of conflicts is called a* conflict set*.*

To explore all realisations, the synthesiser starts with $Q := \mathcal{R}^{\mathfrak{D}}$ and picks some realisation $r \in Q$.[1] Either $D_r \models \Phi$ and we immediately return r, or a conflict is found: then Q is pruned by removing all conflicts that the verifier found. If Q is empty, we are done: each realisation violates a property $\varphi \in \Phi$.

[1] We focus on program-level synthesis, and refrain from discussing important implementation aspects—like how to represent Q—here.

3.2 Verifier

Definition 6. *A verifier is sound and complete, if for family $\mathfrak{D}$, realisation r, and specification Φ, the verifier terminates, the returned conflict set is empty iff $D_r \models \Phi$, and if it is not empty, it contains a conflict $\bar{r}_\varphi \subseteq r$ for some $\varphi \in \Phi$.*

Algorithm 1 outlines a basic verifier. It uses an off-the-shelf probabilistic model-checking procedure CHECK(D_r, φ) to determine all violated $\varphi \in \Phi$. The algorithm then iterates over the violated φ and computes critical sets C of D_r that induce sub-MCs such that $D_r \downarrow C \not\models \varphi$ (line 6). The critical sets for safety properties can be obtained via standard methods [2], and support for liveness properties is discussed at the end of the section.

(a) Fragment of D_{r_1} (b) Sub-MC of D_{r_1} with $C = \{0\}$

Fig. 3. Fragment and corresponding sub-MC that suffices to refute Φ

Algorithm 1. Verifier

```
1: function VERIFY(family 𝔇, realisation r, specification Φ)
2:   Violated ← ∅; Conflict ← ∅; D_r ← GENERATEMC(𝔇, r);
3:   for all φ ∈ Φ do
4:     if not CHECK(D_r, φ) then Violated ← Violated ∪ {φ}
5:   for all φ ∈ Violated do
6:     C_φ ← COMPUTECRITICALSET(D_r, φ)
7:     Conflict ← Conflict ∪ GENERATECONFLICT(𝔇, r, C_φ)
8:   return Conflict
```

Example 2. Reconsider $\mathfrak{D}$ from Example 1 with $\Phi := \{\varphi := \mathbb{P}_{\leq 2/5}(\Diamond\{2\})\}$. Assume the synthesiser picks realisation r_1. The verifier builds D_{r_1} and determines $D_{r_1} \not\models \Phi$. Observe that the verifier does not need the full realisation D_{r_1} to refute Φ. In fact, the paths in the fragment of D_{r_1} in Fig. 3a (ignoring the outgoing transitions of states 1 and 2) suffice to show that the probability to reach state 2 exceeds $2/5$. Formally, the fragment in Fig. 3b is a sub-MC $D_{r_1}\downarrow C$ with critical states $C = \{0\}$. The essential property is [70]:

If a sub-MC of a MCD refutes a safety property φ, then D refutes φ too.

Observe that $D_{r_1} \downarrow C$ is part of D_{r_2} too. Formally, the sub-MC of $D_{r_2} \downarrow C$ is isomorphic to $D_{r_1}\downarrow C$ and therefore also violates Φ. Thus, $D_{r_2} \not\models \Phi$.

Finally, the verifier translates the obtained critical set C for realisation r to a conflict $\mathit{Conflict}(C, r) \subseteq r$ and stores it in the conflict set Conflict (line 7). The procedure GENERATECONFLICT$(\mathfrak{D}, r, C)$ identifies the subset of parameters K that occur in the sub-MCs $D_r \downarrow C$ and returns the corresponding partial realisation. The proposition below clarifies the relation between critical sets and conflicts.

Proposition 1. *If C is a critical set for D_r and φ, with $D_r \not\models \varphi$ then C is also a critical set for each $D_{r'}$, $r' \supseteq \mathit{Conflict}(C, r)$, and furthermore $D_{r'} \not\models \varphi$ holds.*

Example 3. Recall from Example 2 that $D_{r_2} \not\models \Phi$. This can be concluded *without constructing* D_{r_2}. Just considering r_2, $\mathfrak{D}$ and C suffices: First, take all parameters occurring in $\mathfrak{P}(c)$ for any $c \in C$. This yields $\{k_1, k_2\}$. The partial realisation $\bar{r} := \{k_1 \mapsto 1, k_2 \mapsto 2\}$ is a conflict. The values for the other parameters do not affect the shape of the sub-MC induced by C. Realisation $r_2 \supseteq \bar{r}$ only varies from r_1 in the value of k_3, but $\bar{r}(k_3) = \bot$, i.e., k_3 is not included in the conflict. This suffices to conclude $D_{r_2} \not\models \Phi$.

Conflicts for Liveness Properties. To support liveness properties such as $\varphi := \mathbb{P}_{>\lambda}(\Diamond G)$, we first consider a (standard) dual safety property $\varphi' := \mathbb{P}_{<1-\lambda}(\Diamond B)$, where B is the set of all states that do not have a path to G. Observe that B can be efficiently computed using graph algorithms. We have to be careful, however.

Example 4. Consider D_{r_1}, and let $\varphi := \mathbb{P}_{>3/5}(\Diamond\{4\})$. $D_{r_1} \not\models \varphi$. Then, $\varphi' = \mathbb{P}_{<2/5}(\Diamond\{2\})$, which is refuted with critical set $C = \{0\}$ as before. Although $D_{r_2} \downarrow C$ is again isomorphic to $D_{r_1} \downarrow C$, we have $D_{r_2} \models \varphi$. The problem here is that state 2 is in B for D_{r_1} as $r_1(k_3) = 2$, but not in B for D_{r_2}, as $r_2(k_3) = 4$.

To prevent the problem above, we ensure that the states in B cannot reach G in other realisations, by including B in the critical set of φ: Let C be the critical set for the dual safety property φ'. We define $B \cup C$ as critical states for φ. Together, we reach states B with a critical probability mass[2], and never leave B.

Example 5. In D_{r_1}, we compute critical states $\{0, 2\}$, preventing the erroneous reasoning from the previous example. For D_{r_4}, we compute $C' = \{0\} \cup \{3\}$ as critical states, and as $D_{r_4} \downarrow C'$ is isomorphic to $D_{r_3} \downarrow C'$, we obtain that $D_{r_3} \not\models \varphi$.

4 Syntax-Guided Synthesis for Probabilistic Programs

Probabilistic models are typically specified by means of a program-level modelling language, such as PRISM [50], PIOA [73], JANI [18], or MODEST [15]. We propose a *sketching language* based on the PRISM modelling language. A sketch, a syntactic template, defines a high-level structure of the model and represents a-priori knowledge about the system under development. It effectively

[2] A good implementation takes a subset of $B' \subseteq B$ by considering the $\texttt{Prob}(D, \Diamond B')$.

```
hole X either { XA is 1 cost 3, 2}
hole Y either { YA is 1, 3 }
hole Z either { 1, 2 }
constraint  !(XA && YA);
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=X + 0.5: s'=Y;
s = 1 -> s'=s+Z;
s >= 2 -> s'=s;
endmodule
```

(a) Program sketch $\mathfrak{S}_H$

```
module rex
s : [0..3] init 0;
s = 0 -> 0.5: s'=1 + 0.5: s'=3;
s = 1 -> s'=3;
s >= 2 -> s'=s;
endmodule
```

(b) Instance $\mathfrak{S}_H(\{X{\mapsto}1, Z{\mapsto}2, Y{\mapsto}3\})$

Fig. 4. Running example

restricts the size of the design space and also allows to concisely add constraints and costs to its members. The proposed language is easily supported by model checkers and in particular by methods for generating CEs [33,71]. Below, we describe the language, and adapt CEGIS from state level to program level. In particular, we employ so-called *program-level CEs*, rather than CEs on the state level.

4.1 A Program Sketching Language

Let us briefly recap how the model-based concepts translate to language concepts in the PRISM guarded-command language. A PRISM program consists of one or more reactive modules that may interact with each other. Consider a single module. This is not a restriction, every PRISM program can be flattened into this form. A module has a set of bounded variables spanning its state space. Transitions between states are described by guarded commands of the form:

$$\texttt{guard} \;\; \rightarrow \;\; p_1 : \texttt{update}_1 + \ldots\ldots + p_n : \texttt{update}_n$$

The guard is a Boolean expression over the module's variables of the model. If the guard evaluates to true, the module can evolve into a successor state by updating its variables. An update is chosen according to the probability distribution given by expressions $p_1, \ldots, p_n$. In every state enabling the guard, the evaluation of $p_1, \ldots, p_n$ must sum up to one. Overlapping guards yield non-determinism and are disallowed here.

Roughly, a program $\mathcal{P}$ thus is a tuple (Var, E) of variables and commands. For a program $\mathcal{P}$, the *underlying MC* $[\![\,\mathcal{P}\,]\!]$ are $\mathcal{P}$'s semantics. We lift specifications: Program $\mathcal{P}$ satisfies a specification Φ, iff $[\![\,\mathcal{P}\,]\!] \models \Phi$, etc.

A sketch is a program that contains *holes*. Holes are the program's open parts and can be replaced by one of finitely many options. Each option can *optionally* be named and associated with a cost. They are declared as:

$$\texttt{hole}\ h\ \texttt{either}\{\, x_1\ \texttt{is}\ \ \texttt{expr}_1\ \texttt{cost}\ c_1, \ldots, x_k\ \texttt{is}\ \ \texttt{expr}_k\ \texttt{cost}\ c_k \,\}$$

where h is the hole identifier, x_i is the option name, $\texttt{expr}_i$ is an expression over the program variables describing the option, and c_i is the cost, given as expressions over natural numbers. A hole h can be used in commands in a similar

Algorithm 2. Synthesiser (feasibility synthesis)

1: **function** SYNTHESIS(program sketch $\mathfrak{S}_H$, specification Φ, budget B)
2: $\psi \leftarrow$ INITIALISE($\mathfrak{S}_H, B$)
3: $R \leftarrow$ GETREALISATION(ψ)
4: **while** $R \neq$ `Unsat` **do**
5: $C \leftarrow$ VERIFY($\mathfrak{S}_H(R), \Phi$)
6: **if** $C = \varnothing$ **then return** R
7: $\psi \leftarrow \psi \wedge \left(\bigwedge_{\bar{R} \in C} \text{LEARNFROMCONFLICT}(\mathfrak{S}_H, \bar{R}) \right)$
8: $R \leftarrow$ GETREALISATION(ψ)
9: **return** `Unsat`

way as a constant, and may occur multiple times within multiple commands, in both guards and updates. The option names can be used to describe constraints on realisations. These propositional formulae over option names restrict realisations, e.g.,

$$\texttt{constraint}(x_1 \vee x_2) \implies x_3$$

requires that whenever the options x_1 or x_2 are taken for some (potentially different) holes, option x_3 is also to be taken.

Definition 7 (Program sketch). *A* (PRISM program) sketch *is a tuple* $\mathfrak{S}_H := (\mathcal{P}_H, \mathsf{Option}_H, \Gamma, \mathsf{cost})$ *where* $\mathcal{P}_H$ *is a program with a set* H *of holes with options* Option_H, Γ *are constraints over* Option_H, *and* $\mathsf{cost}\colon \mathsf{Option}_H \to \mathbb{N}$ *option-costs.*

Example 6. We consider a small running example to illustrate the main concepts. Figure 4a depicts the program sketch $\mathfrak{S}_H$ with holes $H = \{\mathrm{X}, \mathrm{Y}, \mathrm{Z}\}$. For X, the options are $\mathsf{Option}_\mathrm{X} = \{1, 2\}$. The constraint forbids XA and YA both being one; it ensures a non-trivial random choice in state `s=0`.

Remark 1. Below, we formalise notions previously used on families. Due to flexibility of sketching (in particular in combination with multiple modules), it is *not* straightforward to provide family semantics to sketches, but the concepts are analogous. In particular: holes and parameters are similar, parameter domains are options, and family realisations and sketch realisations both yield concrete instances from a family/sketch. The synthesis problems carry over naturally.

Definition 8 (Realisations of sketches). *Let* $\mathfrak{S}_H := (\mathcal{P}_H, \mathsf{Option}_H, \Gamma, \mathsf{cost})$ *be a sketch, a* sketch realisation *on holes* H *is a function* $R\colon H \to \mathsf{Option}_H$ *with* $\forall h \in H.\ R(h) \in \mathsf{Option}_h$ *and that satisfies all constraints in* Γ. *The* sketch instance $\mathfrak{S}_H(R)$ *for realisation* R *is the program (without holes)* $\mathcal{P}_H[H/R]$ *in which each hole* $h \in H$ *in* $\mathcal{P}_H$ *is replaced by* $R(h)$. *The cost* $c(R)$ *is the sum of the cost of the selected options,* $c(R) := \sum_{h \in H} \mathsf{cost}(R(h))$.

Example 7. We continue Example 6. The program in Fig. 4b reflects $\mathfrak{S}_H(R)$ for realisation $R = \{X \mapsto 1, Z \mapsto 2, Y \mapsto 3\}$, with $c(R) = 3$ as $\mathsf{cost}(R(X)) = 3$ and all other options have cost zero. For realisation $R' = \{Y, Z \mapsto 1, X \mapsto 2\}$, $c(R') = 0$.

The assignment $\{X, Y, Z \mapsto 1\}$ violates the constraint and is not a realisation. In total, $\mathfrak{S}_H$ represents $6 = 2^3 - 2$ programs and their underlying MCs.

4.2 A Program-Level Synthesiser

Feasibility synthesis. The synthesiser follows the steps in Alglorithm 2. During the synthesis process, the synthesiser stores and queries the set of realisations not yet pruned. These remaining realisations are represented by (the satisfying assignments of) the first-order formula ψ over hole-assignments. Iteratively extending ψ with conjunctions thus prunes the remaining design space.

We give a brief overview, before detailing the steps. InitialiSe$(\mathfrak{S}_H, B)$ constructs ψ such that it represents *all* sketch realisations that satisfy the constraints in the sketch $\mathfrak{S}_H$ within the budget B. GetRealisation(ψ) exploits an SMT-solver for linear (bounded) integer arithmetic to obtain a realisation R consistent with ψ, or `Unsat` if no such realisation exists. As long as new realisations are found, the verifier analyses them (line 5) and returns a conflict set C. If $C = \varnothing$, then $\mathfrak{S}_H(R)$ satisfies the specification Φ and the search is terminated. Otherwise, the synthesiser updates ψ based on the conflicts (line 7). R is always pruned.

InitialiSe$(\mathfrak{S}_H, B)$: Let hole $h \in H$ have (ordered) options $\mathsf{Option}_h = \{o_h^1, \ldots, o_h^n\}$. To encode realisation R, we introduce integer-valued meta-variables $K_H := \{\kappa_h \mid h \in H\}$ with the semantics that $\kappa_h = i$ whenever hole h has value o_h^i, i.e., $R(h) = o_h^i$. We set $\psi := \psi_{\text{opti}} \wedge \psi_\Gamma \wedge \psi_{\text{cost}}$, where ψ_{opti} ensures that each hole is assigned to some option, ψ_Γ ensures that the sketch's constraints Γ are satisfied, and ψ_{cost} ensures that the budget is respected. These sub-formulae are:

$$\psi_{\text{opti}} := \bigwedge_{h \in H} 1 \leq \kappa_h \leq |\mathsf{Option}_h|, \qquad \psi_\Gamma := \bigwedge_{\gamma \in \Gamma} \gamma[N_h^i / \kappa_h = i],$$

$$\psi_{\text{cost}} := \sum_{h \in H} \omega_h \leq B \wedge \left(\bigwedge_{h \in H} \bigwedge_{i=1}^{|\mathsf{Option}_h|} \kappa_h = i \rightarrow \omega_h = \mathsf{cost}(o_h^i) \right),$$

where $\gamma[N_h^i / \kappa_h = i]$ denotes that in every constraint $\gamma \in \Gamma$ we replace each option name N_h^i for an option o_h^i with $\kappa_h = i$, and ω_h are fresh variables storing the cost for the selected option at hole h.

Example 8. For sketch $\mathfrak{S}_H$ in Fig. 4a, we obtain (with slight simplifications)

$$\psi := 1 \leq \kappa_X \leq 2 \wedge 1 \leq \kappa_Y \leq 2 \wedge 1 \leq \kappa_Z \leq 2 \wedge \neg(\kappa_X = 1 \wedge \kappa_Y = 1) \wedge$$
$$\omega_X + \omega_Y + \omega_Z \leq B \wedge \kappa_X = 1 \rightarrow \omega_X = 3 \wedge \kappa_X = 2 \rightarrow \omega_X = 0 \wedge \omega_Y = 0 = \omega_Z.$$

GetRealisation(ψ): To obtain a realisation R, we check satisfiability of ψ. The solver either returns `Unsat` indicating that the synthesiser is finished, or `Sat`, together with a satisfying assignment $\alpha_R \colon K_H \to \mathbb{N}$. The assignment α_R uniquely identifies a realisation R by $R(h) := o_h^{\alpha_R(\kappa_h)}$. The sum over all ω_H gives $c(R)$.

Algorithm 3. Synthesiser (max synthesis)

1: **function** SYNTHESIS($\mathfrak{S}_H$, Φ, B, goal predicate G, tolerance ε)
2: $\lambda^* \leftarrow \infty$, $R^* \leftarrow$ `Unsat`, $\psi \leftarrow$ INITIALISE($\mathfrak{S}_H, B$)
3: $R \leftarrow$ GETREALISATION(ψ)
4: **while** $R \neq$ `Unsat` **do**
5: $C, \lambda_{\text{new}} \leftarrow$ OPTIMISEVERIFY($\mathfrak{S}_H(R), \Phi, G, \lambda^*, \varepsilon$)
6: **if** $C = \varnothing$ **then** $\lambda^*, R^* \leftarrow \lambda_{\text{new}}, R$
7: $\psi \leftarrow \psi \wedge \left(\bigwedge_{\bar{R} \in C}$ LEARNFROMCONFLICT($\mathfrak{S}_H, \bar{R}$)$\right)$
8: $R \leftarrow$ GETREALISATION(ψ)
9: **return** R^*

```
const int X = 1, Y = 3;
...
module rex
s : [0..3] init 0;
s=0 -> 0.5: s'=X + 0.5: s'=Y;
endmodule
```

(a) CE for upper bound

```
...
module rex
s : [0..3] init 0;
s=0 -> 0.5:s'=X + 0.5:s'=Y;
s=3 -> s'=3
endmodule
```

(b) CE for lower bound

Fig. 5. CEs for (a) $\mathbb{P}_{\leq 0.4}[F\ \texttt{s=3}]$ and (b) $\mathbb{P}_{>0.6}[F\ \texttt{s=2}]$.

VERIFY($\mathfrak{S}_H(r), \Phi$): invokes any sound and complete verifier, e.g., an adaption of the verifier from Sect. 3.2 as presented in Sect. 4.3.

LEARNFROMCONFLICT($\mathfrak{S}_H, \bar{R}$): For a conflict[3] $\bar{R} \in C$, we add the formula

$$\neg\Big(\bigwedge_{h \in H, \bar{R}(h) \neq \bot} \kappa_h = \alpha_{\bar{R}}(\kappa_h) \Big).$$

The formula excludes realisations $R' \supseteq \bar{R}$. Intuitively, the formula states that the realisations remaining in the design space (encoded by the updated ψ) cannot assign the h as in $\bar{R}$ (for holes where $\bar{R}(h) \neq \bot$).

Example 9. Consider ψ from Example 8. The satisfying assignment (for $B \geq 3$) is $\{\kappa_X \mapsto 1, \kappa_Y, \kappa_Z \mapsto 2, \omega_X \mapsto 3, \omega_Y, \omega_Z \mapsto 0\}$ represents R, $c(R) = 3$ from Example 6. Consider $\Phi = \{\mathbb{P}_{\leq 0.4}[\Diamond\ \texttt{s=3}]\}$. The verifier (for now, magically) constructs a conflict set $\{\bar{R}\}$ with $\bar{R} = \{\mathrm{Y} \mapsto 3\}$. The synthesiser updates $\psi \leftarrow \psi \wedge \kappa_Y \neq 2$ (recall that $\kappa_Y = 2$ encodes $Y \mapsto 3$). A satisfying assignment $\{\kappa_X, \kappa_Y, \kappa_Z \mapsto 1\}$ for ψ encodes R' from Example 7. As $\mathfrak{S}_H(R') \models \Phi$, the verifier reports no conflict.

Optimal Synthesis. We adapt the synthesiser to support max synthesis, cf. Alglorithm 3. Recall the problem aims at maximizing the probability of reaching

[3] As in Sect. 3.1: A *partial realisation* for $\mathfrak{S}_H$ is a function $\bar{R}\colon H \to \mathsf{Option}_H \cup \{\bot\}$ s.t. $\forall h \in H.\ \bar{R}(h) \in \mathsf{Option}_h \cup \{\bot\}$. For partial realisations $\bar{R}_1, \bar{R}_2$, let $\bar{R}_1 \subseteq \bar{R}_2$ iff $\forall h \in H.\ \bar{R}_1(h) \in \{\bar{R}_2(h), \bot\}$. Let R be a realisation s.t. $\mathfrak{S}_H(R) \not\models \varphi$ for $\varphi \in \Phi$. Partial realisation $\bar{R}_\varphi \subseteq R$ is a *conflict* for φ iff $\forall R' \supseteq \bar{R}_\varphi\ \mathfrak{S}_H(R') \not\models \varphi$.

states described by a predicate G, w.r.t. the tolerance $\varepsilon \in (0,1)$. Algorithm 3 stores in λ^* the maximal probability $\mathsf{Prob}(\mathfrak{S}_H(R), \Diamond G)$ among all considered realisations R, and this R in R^*. In each iteration, an optimising verifier is invoked (line 5) on realisation R. If $\mathfrak{S}_H(R) \models \Phi$ and $\mathsf{Prob}(\mathfrak{S}_H(R), \Diamond G) > \lambda^*$, it returns an empty conflict set *and* $\lambda_{\text{new}} := \mathsf{Prob}(\mathfrak{S}_H(R), \Diamond G)$. Otherwise, it reports a conflict set for $\Phi \cup \{\mathbb{P}_{\geq(1-\varepsilon)\cdot\lambda^*}(\Diamond G)\}$.

4.3 A Program-Level Verifier

We now adapt the state-level verifier from Sect. 3.2 in Alglorithm 1 to use program-level counterexamples [71] for generating conflicts, [68, Appendix] contains details.

GENERATEMC$(\mathfrak{S}_H, R)$: This procedure first constructs the instance $\mathfrak{S}_H(R)$, i.e., a program without holes, from $\mathfrak{S}_H$ and R, as in Fig. 4b: Constraints in the sketch are removed, as they are handled by the synthesiser. This approach allows us to use any model checker supporting PRISM programs. The realisation is passed separately, the sketch is parsed *once* and then appropriately instantiated. The instance is then translated into the underlying MC $[\![\, \mathfrak{S}_H(R) \,]\!]$ via standard procedures, with transitions annotated with their generating commands.

COMPUTECRITICALSET(D, φ) computes program-level CEs as analogue of critical sets. They are defined on commands rather than on states. Let $\mathcal{P} = (\text{Var}, E)$ be a program with commands E. Let $\mathcal{P}_{|E'} := (\text{Var}, E')$ denote the restriction of $\mathcal{P}$ to E' (with variables and initial states as in $\mathcal{P}$). Building $\mathcal{P}_{|E'}$ may introduce deadlocks in $[\![\, \mathcal{P}_{|E'} \,]\!]$ (just like a critical set introduces deadlocks). To remedy this, we use the standard operation fixdl, which takes a program and adds commands that introduce self-loops for states without enabled guard.

Definition 9. *For program $\mathcal{P} = (\text{Var}, E)$ and specification Φ with $\mathcal{P} \not\models \Phi$, a program-level CE $E' \subseteq E$ is a set of commands, such that for all (non-overlapping) programs $\mathcal{P}' = (\text{Var}, E'')$ with $E'' \supseteq E'$ (i.e, extending $\mathcal{P}'$), $\mathsf{fixdl}(\mathcal{P}') \not\models \Phi$.*

Example 10. Reconsider $\Phi = \{\mathbb{P}_{\leq 0.4}[\Diamond\ \texttt{s=3}]\}$. Figure 5a shows a CE for $\mathfrak{S}_H(R)$ in Fig. 4. The probability to reach `s=3` in the underlying MC is $0.5 > 0.4$.

For safety properties, program-level CEs coincide with high-level CEs proposed in [71], their extension to liveness properties follows the ideas on families. The program-level CEs are computed by an extension of the MaxSat [14] approach from [33], [68, Appendix] contains details and extensions.

GENERATECONFLICT$(\mathfrak{S}_H, R, E)$ generates conflicts from commands: we map commands in $\mathfrak{S}_H(R)_{|E}$ to the commands from $\mathfrak{S}_H$, i.e., we restore the information about the critical holes corresponding to the part of the design space that can be pruned by CE E. Formally, $\mathit{Conflict}(E, R)(h) = R(h)$ for all $h \in H$ that appear in restriction $\mathfrak{S}_{H|E}$.

Proposition 2. *If E is a CE for $\mathfrak{S}_H(R)$, then E is also a CE for each $\mathfrak{S}_H(R')$, $R' \supseteq \mathit{Conflict}(E, R)$.*

Example 11. The CEs in Fig. 5a contain commands which depend on the realisations for holes X and Y. For these fixed values, the program violates the specification *independent of the value for Z*, so Z is not in the conflict $\{\mathrm{X} \mapsto 1, \mathrm{Y} \mapsto 3\}$.

5 Experimental Evaluation and Discussion

Implementation. We evaluate the synthesis framework with a prototype[4] using the SMT-solver Z3 [56], and (an extension of) the model checker Storm [34].

Case Studies. We consider the following three case studies:

Dynamic Power Management (DPM). The goal of this adapted DPM problem [13] is to trade-off power consumption for performance. We sketch a controller that decides based on the current workload, inspired by [39]. The fixed environment contains no holes. The goal is to synthesise the guards and updates to satisfy a specification with properties such as φ_1: the expected number of lost requests is below λ, and φ_2: the expected energy consumption is below κ.

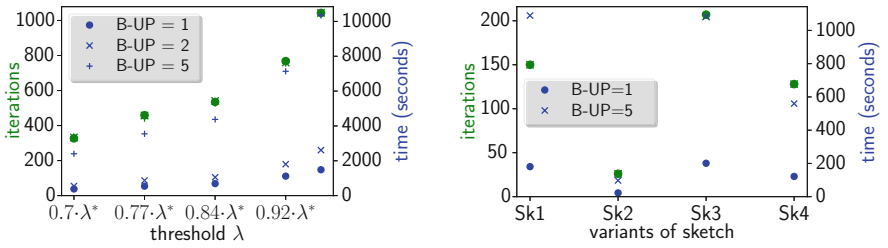
Intrusion describes a network (adapted from [51]), in which the controller tries to infect a target node via intermediate nodes. A failed attack makes a node temporarily harder to intrude. We sketched a partial strategy aiming to minimise the expected time to intrusion. Constraints encode domain specific knowledge.

Grid is based on a classical benchmark for solving partially observable MDPs (POMDPs) [48]. To solve POMDPs, the task is to find an observation-based strategy, which is undecidable for the properties we consider. Therefore, we resort to finding a deterministic k-state strategy [55] s.t. in expectation, the strategy requires less than λ steps to the target. This task is still hard: finding a memoryless, observation-based strategy is already NP-hard [29,69]. We create a family describing all k-state strategies (for some fixed k) for the POMDP. Like in [47] actions are reflected by parameters, while parameter dependencies ensure that the strategy is observation-based.

Evaluation. We compare w.r.t. an enumerative approach. That baseline linearly depends on the number of realisations, and the underlying MCs' size. We focus on sketches where all realisations are explored, as relevant for optimal synthesis. For concise presentation we use `Unsat` variants of feasibility synthesis. Enumerative methods perform mostly independent of the order of enumerating realisations. We evaluate results for *DPM*, and summarise further results. All results are obtained on a Macbook MF839LL/A, within 3 h and using less than 8 GB RAM.

DPM has 9 holes with 260 K realisations, and MCs have 5 K (reachable) states on average, ranging from 2 K to 8 K states. *The performance of CEGIS significantly depends on the specification, namely, on the thresholds appearing in the*

[4] https://github.com/moves-rwth/sketching.



(a) Performance for varying λ (b) Performance for varying $\mathfrak{S}_H$

Fig. 6. Performance (runtime and iterations) on DPM (Color figure online)

properties. Fig. 6a shows how the number of iterations (left axis, green circle) and the runtime in seconds (right axis, blue) change for varying λ for property φ_1 (stars and crosses are explained later). We obtain a speedup of 100× over the baseline for $\lambda = 0.7{\cdot}\lambda^*$, dropping to 23× for $\lambda = 0.95{\cdot}\lambda^*$, where λ^* is the minimal probabilty over all realisations. The strong dependency between performance and "unsatisfiability" is not surprising. The more unsatisfiable, the smaller the conflicts (as in [33]). Small conflicts have a double beneficial effect. First, the prototype uses an optimistic verifier searching for minimal conflicts; small conflicts are found faster than large ones. Second, small conflicts prune more realisations. A slightly higher number of small conflicts yields a severe decrease in iterations. Thus *the further the threshold from the optimum, the better the performance.*

Reconsider Fig. 6a, crosses and stars correspond to a variant in which we have blown up the state space of the underlying MCs by a factor B-UP. Observe that performance degrades similarly for the baseline and our algorithm, which means that *the speedup w.r.t. the baseline is not considerably affected by the size of the underlying MCs.* This observation holds for various models and specifications.

Varying the sketch tremendously affects performance, cf. Fig. 6b for the performance on variants of the original sketch with some hole substituted by one of its options. The framework performs significantly better on sketches with holes that lie in local regions of the MC. Holes relating to states all-over the MC are harder to prune. Finally, our prototype generally performs better with specifications that have multiple (conflicting) properties: Some realisations can be effectively pruned by conflicts w.r.t. property φ_1, whereas other realisations are easily pruned by conflicts w.r.t., e.g., property φ_2.

Intrusion has 26 holes and 6800 K realisations, the underlying MCs have only 500 states on average. We observe an even more significant effect of the property thresholds on the performance, as the number of holes is larger (recall the optimistic verifier). We obtain a speedup of factor 2200, 250 and 5 over the baseline, for thresholds $0.7{\cdot}\lambda^*$, $0.8{\cdot}\lambda^*$ and $0.9{\cdot}\lambda^*$, respectively. For $0.7{\cdot}\lambda^*$, many conflicts contain only 8 holes. Blowing up the model does not affect the obtained speedups. Differences among variants are again significant, albeit less extreme.

Grid is structurally different: only 6 holes in 3 commands and 1800 realisations, but MCs having 100 K states on average. Observe that reaching the targets on expectation below some threshold implies that the goal must almost surely be reached. The MCs' topology and the few commands make pruning hard: our algorithm needs more than 400 iterations. Still, we obtain a 3× speedup for $\lambda = 0.98{\cdot}\lambda^*$. Pruning mostly follows from reasoning about realisations that do not reach the target almost surely. Therefore, the speedup is mostly independent of the relation between λ and λ^*.

Discussion. *Optimistic verifiers* search for a minimal CE and thus solve an NP-hard problem [28,71]. In particular, we observed a lot of overhead when the smallest conflict is large, and any small CE that can be cheaply computed might be better for the performance (much like the computation of unsatisfiable cores in SMT solvers). Likewise, reusing information about holes from previous runs might benefit the performance. Improvements in concise sketching, and exploiting the additional structure, will also improve performance.

Sketching. Families are simpler objects than sketches, but their explicit usage of states make them inadequate for modelling. Families can be lifted to a (restricted) sketching class, as in [27]. However, additional features like conflicts significantly ease the modelling process. Consider *intrusion*: Without constraints, the number of realisations grows to $6{\cdot}10^{11}$. Put differently, the constraint allows to discard over 99.99% of the realisations up front. Moreover, constraints can exclude realisations that would yield unsupported programs, e.g, programs with infinite state spaces. While modelling concise sketches with small underlying MCs, it may be hard to avoid such invalid realisations without the use of constraints.

Comparison with CEGAR. We also compared with our CEGAR-prototype [27], which leverages an abstraction-refinement loop for the synthesis. We observed that there are synthesis problems where CEGIS significantly outperforms CEGAR and vice versa. Details, including an evaluation of the strengths and weaknesses of CEGIS compared to CEGAR, are reported in [68, Appendix]. In our future work, we will explore how to effectively combine both approaches to improve the performance and scalability of the synthesis process.

References

1. Abate, A., David, C., Kesseli, P., Kroening, D., Polgreen, E.: Counterexample guided inductive synthesis modulo theories. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 270–288. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_15
2. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.-P., Wimmer, R.: Counterexample generation for discrete-time Markov models: an introductory survey. In: Bernardo, M., Damiani, F., Hähnle, R., Johnsen, E.B., Schaefer, I. (eds.) SFM 2014. LNCS, vol. 8483, pp. 65–121. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07317-0_3

3. Alur, R., et al.: Syntax-guided synthesis. In: Dependable Software Systems Engineering, NATO Science for Peace and Security Series, vol. 40, pp. 1–25. IOS Press (2015)
4. Alur, R., Singh, R., Fisman, D., Solar-Lezama, A.: Search-based program synthesis. Commun. ACM **61**(12), 84–93 (2018)
5. Antonik, A., Huth, M., Larsen, K.G., Nyman, U., Wasowski, A.: 20 years of modal and mixed specifications. Bull. EATCS **95**, 94–129 (2008)
6. Baier, C., de Alfaro, L., Forejt, V., Kwiatkowska, M.: Model checking probabilistic systems. Handbook of Model Checking, pp. 963–999. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-10575-8_28
7. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
8. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. IEEE Trans. Softw. Eng. **30**(5), 295–310 (2004)
9. 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
10. Becker, S., Koziolek, H., Reussner, R.: The Palladio component model for model-driven performance prediction. J. Syst. Softw. **82**(1), 3–22 (2009)
11. Benes, N., Kretínský, J., Larsen, K.G., Møller, M.H., Sickert, S., Srba, J.: Refinement checking on parametric modal transition systems. Acta Inf. **52**(2–3), 269–297 (2015)
12. Beneš, N., Křetínský, J., Guldstrand Larsen, K., Møller, M.H., Srba, J.: Dual-priced modal transition systems with time durations. In: Bjørner, N., Voronkov, A. (eds.) LPAR 2012. LNCS, vol. 7180, pp. 122–137. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28717-6_12
13. Benini, L., Bogliolo, A., Paleologo, G., Micheli, G.D.: Policy optimization for dynamic power management. IEEE Trans. CAD Integr. Circ. Syst. **8**(3), 299–316 (2000)
14. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
15. Bohnenkamp, H.C., D'Argenio, P.R., Hermanns, H., Katoen, J.P.: MODEST: a compositional modeling formalism for hard and softly timed systems. IEEE Trans. Softw. Eng. **32**(10), 812–830 (2006)
16. Bondy, A.B.: Foundations of Software and System Performance Engineering. Addison Wesley, Boston (2014)
17. Bornholt, J., Torlak, E., Grossman, D., Ceze, L.: Optimizing synthesis with metasketches. In: POPL, pp. 775–788. ACM (2016)
18. Budde, C.E., Dehnert, C., Hahn, E.M., Hartmanns, A., Junges, S., Turrini, A.: JANI: quantitative model and tool interaction. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 151–168. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-662-54580-5_9
19. Calinescu, R., Ghezzi, C., Johnson, K., et al.: Formal verification with confidence intervals to establish quality of service properties of software systems. IEEE Trans. Reliab. **65**(1), 107–125 (2016)
20. Calinescu, R., Ghezzi, C., Kwiatkowska, M.Z., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Commun. ACM **55**(9), 69–77 (2012)

21. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Designing robust software systems through parametric Markov chain synthesis. In: ICSA, pp. 131–140. IEEE (2017)
22. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: RODES: a robust-design synthesis tool for probabilistic systems. In: Bertrand, N., Bortolussi, L. (eds.) QEST 2017. LNCS, vol. 10503, pp. 304–308. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66335-7_20
23. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. J. Syst. Softw. **143**, 140–158 (2018)
24. Cardelli, L., et al.: Syntax-guided optimal synthesis for chemical reaction networks. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 375–395. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_20
25. Černý, P., Chatterjee, K., Henzinger, T.A., Radhakrishna, A., Singh, R.: Quantitative synthesis for concurrent programs. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 243–259. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22110-1_20
26. Češka, M., Dannenberg, F., Paoletti, N., Kwiatkowska, M., Brim, L.: Precise parameter synthesis for stochastic biochemical systems. Acta Inf. **54**(6), 589–623 (2017)
27. Češka, M., Jansen, N., Junges, S., Katoen, J.-P.: Shepherding hordes of Markov chains. In: Vojnar, T., Zhang, L. (eds.) TACAS 2019. LNCS, vol. 11428, pp. 172–190. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-17465-1_10
28. Chadha, R., Viswanathan, M.: A counterexample-guided abstraction-refinement framework for markov decision processes. ACM Trans. Comput. Log. **12**(1), 1:1–1:49 (2010)
29. Chatterjee, K., Chmelik, M., Davies, J.: A symbolic SAT-based algorithm for almost-sure reachability with small strategies in POMDPs. In: AAAI, pp. 3225–3232. AAAI Press (2016)
30. Chaudhuri, S., Clochard, M., Solar-Lezama, A.: Bridging boolean and quantitative synthesis using smoothed proof search. In: POPL. ACM (2014)
31. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M.Z., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: TASE, pp. 85–92. IEEE (2013)
32. Chrszon, P., Dubslaff, C., Klüppelholz, S., Baier, C.: ProFeat: feature-oriented engineering for family-based probabilistic model checking. Formal Aspects Comput. **30**(1), 45–75 (2018)
33. Dehnert, C., Jansen, N., Wimmer, R., Ábrahám, E., Katoen, J.-P.: Fast debugging of PRISM models. In: Cassez, F., Raskin, J.-F. (eds.) ATVA 2014. LNCS, vol. 8837, pp. 146–162. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11936-6_11
34. Dehnert, C., Junges, S., Katoen, J.-P., Volk, M.: A *storm* is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63390-9_31
35. Delahaye, B., et al.: Abstract probabilistic automata. Inf. Comput. **232**, 66–116 (2013)
36. Dureja, R., Rozier, K.Y.: More scalable LTL model checking via discovering design-space dependencies (D^3). In: Beyer, D., Huisman, M. (eds.) TACAS 2018. LNCS, vol. 10805, pp. 309–327. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89960-2_17

37. Filieri, A., Tamburrelli, G., Ghezzi, C.: Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Trans. Softw. Eng. **42**(1), 75–99 (2016)
38. Fiondella, L., Puliafito, A. (eds.): Principles of Performance and Reliability Modeling and Evaluation. SSRE. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-30599-8
39. Gerasimou, S., Tamburrelli, G., Calinescu, R.: Search-based synthesis of probabilistic models for quality-of-service software engineering. In: ASE, pp. 319–330. IEEE Computer Society (2015)
40. Ghezzi, C., Sharifloo, A.M.: Model-based verification of quantitative non-functional properties for software product lines. Inf. Softw. Technol. **55**(3), 508–524 (2013)
41. Gulwani, S., Polozov, O., Singh, R.: Program synthesis. Found. Trends Programm. Lang. **4**(1–2), 1–119 (2017)
42. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric markov models. Softw. Tools Technol. Transf. **13**(1), 3–19 (2011)
43. Henzinger, T.A.: Quantitative reactive modeling and verification. Comput. Sci.-Res. Dev. **28**(4), 331–344 (2013)
44. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using UPPAAL. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal Methods and Testing. LNCS, vol. 4949, pp. 77–117. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78917-8_3
45. Jansen, N., Humphrey, L.R., Tumova, J., Topcu, U.: Structured synthesis for probabilistic systems. CoRR abs/1807.06106, at NFM 2019 (2018, to appear)
46. Junges, S., Jansen, N., Dehnert, C., Topcu, U., Katoen, J.-P.: Safety-constrained reinforcement learning for MDPs. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 130–146. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49674-9_8
47. Junges, S., et al.: Finite-state controllers of POMDPs using parameter synthesis. In: UAI, pp. 519–529. AUAI Press (2018)
48. Kaelbling, L.P., Littman, M.L., Cassandra, A.R.: Planning and acting in partially observable stochastic domains. Artif. Intell. **101**(1–2), 99–134 (1998)
49. Křetínský, J.: 30 years of modal transition systems: survey of extensions and analysis. In: Aceto, L., Bacci, G., Bacci, G., Ingólfsdóttir, A., Legay, A., Mardare, R. (eds.) Models, Algorithms, Logics and Tools. LNCS, vol. 10460, pp. 36–74. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63121-9_3
50. 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
51. Kwiatkowska, M.Z., Norman, G., Parker, D., Vigliotti, M.G.: Probabilistic mobile ambients. Theor. Comput. Sci. **410**(12–13), 1272–1303 (2009)
52. Larsen, K.G.: Verification and performance analysis of embedded and cyber-physical systems using UPPAAL. In: MODELSWARD 2014, pp. IS-11 (2014)
53. Larsen, K.G., Thomsen, B.: A modal process logic. In: LICS, pp. 203–210. IEEE Computer Society (1988)
54. Lindemann, C.: Performance modelling with deterministic and stochastic Petri nets. Perf. Eval. Review **26**(2), 3 (1998)
55. Meuleau, N., Kim, K.E., Kaelbling, L.P., Cassandra, A.R.: Solving POMDPs by searching the space of finite policies. In: UAI, pp. 417–426. Morgan Kaufmann Publishers Inc. (1999)

56. de Moura, L., Bjørner, N.: Z3: an efficient SMT solver. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 337–340. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-78800-3_24
57. Nori, A.V., Ozair, S., Rajamani, S.K., Vijaykeerthy, D.: Efficient synthesis of probabilistic programs. In: PLDI, pp. 208–217. ACM (2015)
58. 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
59. Rodrigues, et al.: Modeling and verification for probabilistic properties in software product lines. In: HASE, pp. 173–180. IEEE (2015)
60. Rosenblum, D.S.: The power of probabilistic thinking. In: ASE, p. 3. ACM (2016)
61. Sharma, V.S., Trivedi, K.S.: Quantifying software performance, reliability and security: an architecture-based approach. J. Syst. Softw. **80**(4), 493–509 (2007)
62. Solar-Lezama, A., Jones, C.G., Bodik, R.: Sketching concurrent data structures. In: PLDI, pp. 136–148. ACM (2008)
63. Solar-Lezama, A., Rabbah, R.M., Bodík, R., Ebcioglu, K.: Programming by sketching for bit-streaming programs. In: PLDI, pp. 281–294. ACM (2005)
64. Solar-Lezama, A., Tancau, L., Bodik, R., Seshia, S., Saraswat, V.: Combinatorial sketching for finite programs. In: ASPLOS, pp. 404–415. ACM (2006)
65. Stewart, W.J.: Probability, Markov Chains, Queues, and Simulation: The Mathematical Basis of Performance Modeling. Princeton university press (2009)
66. Vandin, A., ter Beek, M.H., Legay, A., Lluch Lafuente, A.: QFLan: a tool for the quantitative analysis of highly reconfigurable systems. In: Havelund, K., Peleska, J., Roscoe, B., de Vink, E. (eds.) FM 2018. LNCS, vol. 10951, pp. 329–337. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95582-7_19
67. Varshosaz, M., Khosravi, R.: Discrete time Markov chain families: modeling and verification of probabilistic software product lines. In: SPLC Workshops, pp. 34–41. ACM (2013)
68. Češka, M., Hensel, C., Junges, S., Katoen, J.P.: Counterexample-driven synthesis for probabilistic program sketches. CoRR abs/1904.12371 (2019)
69. Vlassis, N., Littman, M.L., Barber, D.: On the computational complexity of stochastic controller optimization in POMDPs. ACM Trans. Comput. Theor. **4**(4), 12:1–12:8 (2012). https://doi.org/10.1145/2382559.2382563
70. Wimmer, R., Jansen, N., Ábrahám, E., Becker, B., Katoen, J.-P.: Minimal critical subsystems for discrete-time Markov models. In: Flanagan, C., König, B. (eds.) TACAS 2012. LNCS, vol. 7214, pp. 299–314. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28756-5_21
71. Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.-P., Becker, B.: High-Level Counterexamples for Probabilistic Automata. In: Joshi, K., Siegle, M., Stoelinga, M., D'Argenio, P.R. (eds.) QEST 2013. LNCS, vol. 8054, pp. 39–54. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40196-1_4
72. Woodside, M., Petriu, D., Merseguer, J., Petriu, D., Alhaj, M.: Transformation challenges: from software models to performance models. J. Softw. Syst. Model. **13**(4), 1529–1552 (2014)
73. Wu, S., Smolka, S.A., Stark, E.W.: Composition and behaviors of probabilistic I/O automata. Theor. Comput. Sci. **176**(1–2), 1–38 (1997)
74. Zhou, W., Li, W.: Safety-aware apprenticeship learning. In: Chockler, H., Weissenbacher, G. (eds.) CAV 2018. LNCS, vol. 10981, pp. 662–680. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96145-3_38