ANALYSIS OF BIO-PATHWAY MODELS USING PARALLEL ARCHITECTURES

R. RAMANATHAN

(B.E.)

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2017

SUPERVISORS:

ASSOCIATE PROFESSOR WONG WENG FAI

PROFESSOR P. S. THIAGARAJAN

EXAMINERS:

PROFESSOR DONG JIN SONG

ASSOCIATE PROFESSOR HE BINGSHENG

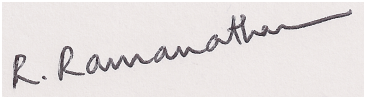
DR. ODED MALER, VERIMAG

DECLARATION

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Singapore, March 31st, 2017



R. Ramanathan

ACKNOWLEDGMENTS

First and foremost, I would like to thank Professor P. S. Thiagarajan and Professor Wong Weng Fai, my supervisors.

I am deeply indebted to Professor Thiagarajan for his patient mentorship and support throughout my time in NUS. Besides research, I also imbibed how to write and present to an audience from him. I am grateful to Professor Weng Fai for constantly motivating me and involving me in a number of projects right from my first year. I thank him for offering invaluable guidance and generous support during the later stages of my Ph.D. studies.

I want to thank Professors Dong Jin Song, Sung Wing Kin and Bingsheng He for serving on my thesis advisory committee and their feedback during the thesis proposal exam.

My sincere thanks to Soumya Paul for mentoring me and the many hours of discussions we had. I owe much gratitude to Yan Zhang for the encouragement and candid feedback on my ideas. Special thanks to the former members of the lab: Andrei, Sucheendra, Benjamin and Liu Bing. I learnt a lot from them. I would also like to thank Zhou Jun and wish him well for the future.

Life during my graduate education at School of Computing has been an enriching experience and I am thankful to all the people who made it possible.

Jing Quan, Charlie and Ratul have been great friends. I wish to thank Kaveh, Lavanya, Akshay and Muthu. They will always inspire me. I enjoyed the numerous meetings I have had with Yogesh, Vanchi, Elavarasi and Sumanan.

I consider myself very fortunate to have great people around in the lab: Peiyong, Haojun, Chern Han, Hoang, Narmada, Chandana, Bora, Kevin, Zhizhou, Mengyuan, Wilson, Yujing, Junqi, Ramesh, Luyu, Bingxin, Abha, Iana, Stefano, Meng Ge, Zhiqiang, Qiangwei and Bokui.

Special thanks to Enrico, Lahiru, Shalinda, Abhijeet, Prasanta, Pooja, Lakshminarasimhan, Anuja, Paramasiven, Parvathy, Yamilet, Malay Singh, Malai, Minh, Suhendry, Sergey, Shin Hwei, Ankit, Marcel, Inian, Pham Khanh, Zhang Peng and Rajendra Prasad.

Finally, none of this would have been possible without Shantha paati, Srimani chithi, Suba chithi and Ganesh mama, who make me who I am and offer their unconditional support and love.

CONTENTS

SUMMARY

We study models of bio-pathways that arise in systems biology.

Often a bio-pathway can be viewed as a network of bio-chemical reactions. One can then model the network as a dynamical system. In this thesis, we explore two classes of such models, namely, a single system of ordinary differential equations and hybrid dynamical systems.

Hybrid systems are multi-mode dynamical systems which evolve over continuous time. The dynamics in each mode is governed by a mode-specific system of differential equations and at discrete instances there can be instantaneous jumps between modes depending typically on the current continuous state.

Both these models —especially when used in systems biology context— are difficult to analyze and the analysis methods one develops are usually computationally intensive and hence difficult to scale. With this as motivation, we broadly explore the twin themes of

(i) Probabilistic approximations of ODEs systems and hybrid systems accompanied by a probabilistic verification technique known as statistical model checking,

(ii) GPU based implementations of the SMC procedures and the related analysis techniques.

In the first part of the thesis, we consider single systems of ODEs. We first recall a previously developed approximation technique in which a system of ODEs is first approximated as a dynamic Bayesian network (DBN). We show how the construction of the DBN can be parallelized via a GPU implementation.

Next we present a parallelized statistical model checking (SMC) based analysis method for ODEs systems. The core component of this technique is an online procedure for verifying whether a numerically generated trajectory of a model satisfies a dynamical property. We then show how this method can be applied to parameter estimation of biopathways to achieve significant performance improvement.

The next part of the thesis focuses on analysis of hybrid systems. We assume that the probability of making a mode transition is proportional to the measure of the set of pairs of time points and value states at which the mode transition is enabled. Based on this, we develop a probabilistic approximation scheme in which the hybrid system can be approximated as a discrete-time Markov chain. However, it is not computationally feasible to compute this Markov chain for high-dimensional systems. Hence we construct a simulation based method for sampling the paths of the Markov chain and carrying out SMC based verification. This probabilistic approximation scheme is then parallelized using a GPU implementation.

We have applied our methods to a number of realistic models. The results indicate that our approximation schemes scale well and can be applied in a number of different settings.

LIST OF TABLES

LIST OF FIGURES

1

INTRODUCTION

1.1 CONTEXT AND MOTIVATION

At the turn of the millennium, the field of *systems biology* emerged as a result of the need for a network level understanding of the cellular components such as genes and proteins [6]. The modelling and analysis of bio-pathways dynamics is a core activity in systems biology. Often, one views a bio-pathway as a network of bio-chemical reactions and then models the network as a dynamical system. Broadly speaking two fundamentally different approaches guide the choice of the system model. In one approach, the number of molecules of each kind is kept track of and stochastic simulations [7–13] are used to advance the system state one reaction at a time. In the second approach —assuming that all the relevant molecular species are present abundantly— one tracks the concentrations of the molecular species of each kind and ordinary differential equations (ODEs) are used to construct the models [1, 14–18]. Deterministic numerical simulations of the ODEs are then deployed to study the dynamics. Clearly, both approaches are needed to cover different contexts [19]. Here we pursue the second approach.

In general, for a well-defined system of ODEs, under suitable continuity assumptions, the differential equations will have a unique solution [20]. Therefore, the temporal evolution of the system behaviour can be obtained by solving the ODEs. However, bio-pathways usually involve a large number of bio-chemical reactions. Hence the correspond-

ing systems of ODEs will not admit closed-form solutions. Instead one will have to generate trajectories using numerical integration to study the dynamics. Further, the quantitative observations of the system will often have very limited precision. Specifically, the initial concentration levels of the various proteins and rate constants will often be available only as intervals of values. In addition, experimental data in terms of the concentration levels of a few proteins at a small number of time points will also be available only in terms of intervals of values. Moreover, the data will often be gathered using a population of cells. Consequently, when numerically simulating the trajectories of the ODEs model, one must resort to Monte Carlo methods to ensure that sufficiently many values from the relevant intervals are being sampled. As a result, standard analysis tasks such as model validation, parameter estimation and sensitivity analysis will require the generation of a large number of trajectories. Thus motivated *Liu et. al.* [21] developed a probabilistic approximation technique involving the following major steps:

(i) Sample many (of the order of a few million) times from a set of initial states,

(ii) Generate trajectories through numerical integration,

(iii) Store the statistical properties of this set of trajectories in the conditional probability tables (CPTs) of a dynamic Bayesian network (DBN) via a pre-specified discretization of the time and value domains.

Consequently one can carry out all analysis tasks —including parameter estimation and sensitivity analysis— using the DBN [21, 22] via standard Bayesian inferencing techniques. The large number of trajectories and the high dimensionality of the system makes the problem

of constructing the DBN approximation computationally intensive. Recently, graphics processing units (GPUs) have become a compelling platform [23] for a wide variety of computationally intensive tasks. Hence as our first contribution, we present a GPU based construction of the DBN approximation.

The above DBN approximation procedure is nonetheless rigid in that for the analysis of all properties, one must use the same DBN approximation. Further, due to the lack of closed-form solutions, it is not possible to estimate the error involved in the approximation. To get around this, a statistical model checking (SMC) procedure was developed in [24] to approximately and probabilistically analyze the dynamics of a system of ODEs. The basic idea is to assume a probability distribution —to cater for the dynamic variability across a population of cells— over a given set of initial states. Under a natural set of continuity restrictions, it then turns out the set of trajectories that satisfy a given bounded linear-time temporal formula constitutes a measurable set of trajectories to which a probability value can be assigned. This leads to an implicit approximation of the ODEs dynamics as a Markov chain. However one can sample paths through this chain by simply sampling from the initial states and generating numerical trajectories. This SMC procedure is also computationally intensive. As our second contribution, we present a GPU based implementation of this verification procedure.

Though the ODEs model is widely used to describe the dynamic behaviour of the bio-pathways in many contexts it is more natural to use the hybrid system model to capture the pathway dynamics [25–27]. Hybrid systems are dynamical systems which operate in multiple modes with both continuous and discrete dynamics. The continuous dynamics in each mode is governed by a system of ODEs. The discrete dynamics is represented by instantaneous jumps between different modes.

Nevertheless, due to their mixed dynamics, such systems are difficult to analyze. To get around this, we present as our third contribution a probabilistic approximation scheme of a hybrid dynamical system as a Markov chain. Though this Markov chain cannot be constructed explicitly —due to the lack of closed-from solutions— one can sample paths from this chain through sampling the dynamics of the hybrid system models. The underlying theory is much more involved and securing the mathematical basis for the corresponding statistical model checking procedure requires a lot more care. Further, as before, carrying out analysis tasks using this SMC procedure is computationally very intensive. To this end, we present as our final contribution a novel GPU based implementation of this much more sophisticated SMC procedure.

1.2 RESEARCH CONTRIBUTIONS

In summary the main contributions of this thesis are:

- A GPU based construction of the DBN approximation of a system of ODEs,
- A parallelized statistical model checking procedure for a system of ODEs, that exploits the massive parallelism offered by GPUs,
- A probabilistic approximation scheme by which a hybrid dynamical system is represented as a Markov chain accompanied by a SMC procedure,
- A GPU implementation of the above SMC procedure for hybrid dynamical systems.

The technical details concerning these various contributions are presented in the corresponding chapters that follow. We also mention rele-

vant related literature in the chapters. In each chapter we present experimental results using biologically relevant pathway models.

1.3 OUTLINE OF THE THESIS

The thesis is organized as follows:

Chapter 2 discusses the preliminaries on Graphics Processing Units, dynamic Bayesian networks, Markov chains, the logical background, probabilistic model checking, hybrid systems.

In Chapter 3, we first recall how an ODEs system can be approximated as a DBN. We then describe an automatic code generation scheme for GPU based implementation of the DBN approximation.

Chapter 4 describes how statistical model checking can be employed to verify properties of discrete-time Markov chains which represent the system dynamics induced by the discretization of the value and time domains of an ODEs system. We present this sketch of [24] as background material for the next chapter.

In Chapter 5, we develop an automaton-based BLTL path checking framework for the SMC based analysis of a single system of ODEs described in Chapter 4. We then show our technique can be implemented on GPUs to realize a parallelized parameter estimation method.

In Chapter 6, we build a probabilistic approximation of the hybrid system as a discrete-time Markov chain and show how one can use SMC to verify properties expressed in BLTL. We apply our approximation method to two case studies of cardiac cell model and circadian rhythm model and also present its GPU implementation.

Finally Chapter 7 summarizes the contributions of the thesis and points to possible directions of future work.

1.4 DECLARATION

Major portions of the thesis is based on the following works:

1. Hagiescu Andrei, Bing Liu, **R. Ramanathan**, Sucheendra K. Palaniappan, Zheng Cui, Bipasa Chattopadhyay, P. S. Thiagarajan, and Weng-Fai Wong. "GPU code generation for ODE-based applications with phased shared-data access patterns." ACM Transactions on Architecture and Code Optimization (TACO) 10, no. 4 (2013): 55.

2. **R. Ramanathan**, Yan Zhang, Jun Zhou, Benjamin M. Gyori, Weng-Fai Wong, and P. S. Thiagarajan. "Parallelized Parameter Estimation of Biological Pathway Models." Hybrid Systems Biology, pp. 37-57. Springer International Publishing, 2015.

3. Benjamin M. Gyori, Bing Liu, Soumya Paul, **R. Ramanathan**, and P. S. Thiagarajan. "Approximate probabilistic verification of hybrid systems." Hybrid Systems Biology, pp. 96-116. Springer International Publishing, 2015.

2

PRELIMINARIES

In this chapter, we briefly develop the required background material. We first introduce GPUs and their programming model. We then present ODEs models of bio-pathways. Next we describe probabilistic dynamical models, namely DBNs and Markov chains. We then present the temporal logic known as bounded linear-time temporal logic (BLTL). Finally, we present a brief description of hybrid systems.

2.1 GRAPHICS PROCESSING UNITS

A broad class of numerical applications which involve computationally intensive procedures, use specialized processors in order to improve their performance many fold when compared to a conventional implementation based on central processing unit (CPU). Two widely-used processors FPGA and GPU, naturally lend themselves for processing workload that map well to their parallel architecture. Field-programmable gate arrays (FPGAs) are programmable computing hardware which can be reconfigured to exploit instruction level parallelism in parallel applications. Though most of the vendors provide the common processing functions, programming in hardware description languages like VHDL or Verilog and creating the entire design from scratch is costly and requires intensive labour [28]. On the other hand, Graphics processing units (GPUs) are affordable, flexible to program using high-level languages, allow concurrent execution of a large number of

threads and are extensively used due to the high memory bandwidth they offer. In this thesis, we employ GPUs for accelerating computation-intensive simulations of high-dimensional systems in our applications.

2.1.1 *GPGPUs*

General purpose computing on graphics processing units or GPGPUs as it is called has been extensively used in the research community to speed-up computation-intensive parts of applications. In our setting, a large number of parallel simulations of dynamical systems can be accelerated in a GPU platform. A GPU platform is composed of a CPU host which offloads compute-intensive parallel sections of the program to one or more GPU devices containing massively parallel processors. By means of a standard programming model, the parallel sections are realized as computation kernels in the GPU which read in an input data stream, process it and produce an output data stream. A modern Nvidia Tesla GPU server [29] contains thousands of arithmetic processing cores and has a memory bandwidth of the order of hundreds of giga bytes per second. Furthermore, by virtue of their design choice to allocate more transistors to arithmetic logic units (ALUs) than CPUs do, GPUs offer peak floating point performance of the order of few tera floating point operations per second at inexpensive costs.

2.1.2 *GPU programming model*

In recent years, high level programming frameworks like CUDA and OpenCL have opened up GPU programming beyond conventional graphics-specific programming to a wide range of scientific and engineering applications. CUDA is a C-like programming language in which parallel

computations are executed as multi-threaded kernels on the GPU hardware. Specifically, GPU hardware consists of a number of Streaming Multiprocessors (SMs) which in turn contain a number of processing cores that work on data in a SIMD (Single Instruction Multiple Data) fashion. In each kernel, as shown in Figure 2, multiple coordinating threads are grouped into independent *"thread blocks"* such that each block runs on one SM. Blocks are in turn arranged into grids. Furthermore, instructions are issued to scheduling units of parallel threads which execute in lock-step called warps. Warps within the same thread block exchange data using a dedicated on-chip *"shared memory"*. All threads running on the GPU may exchange data using the off-chip *"global memory"*. Figure 1 is a simplified illustration of the GPU memory architecture.

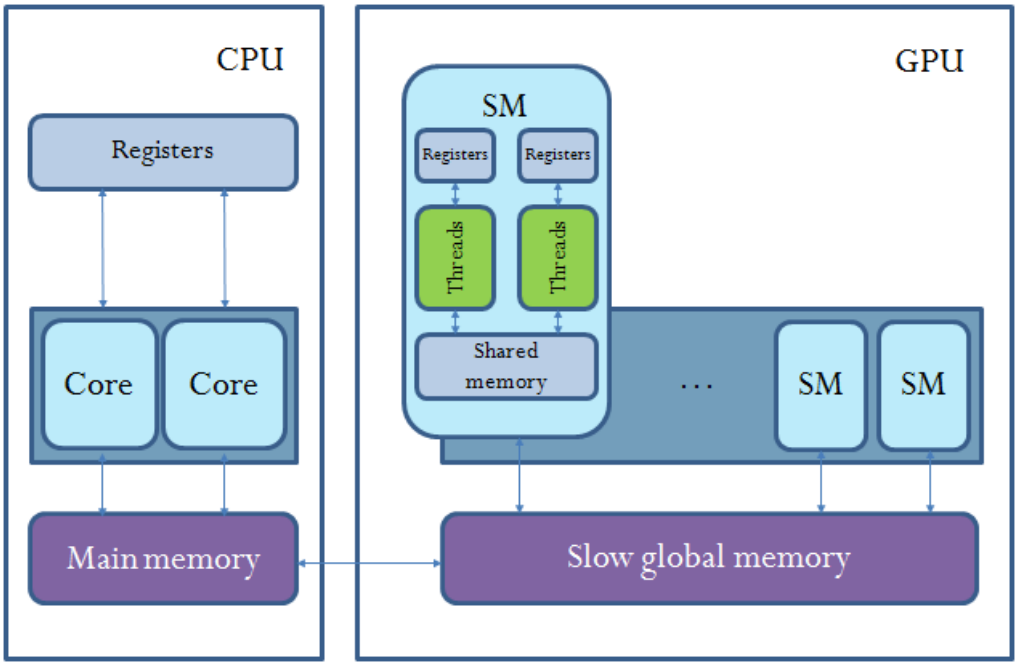


Figure 1: Simplified GPU memory architecture

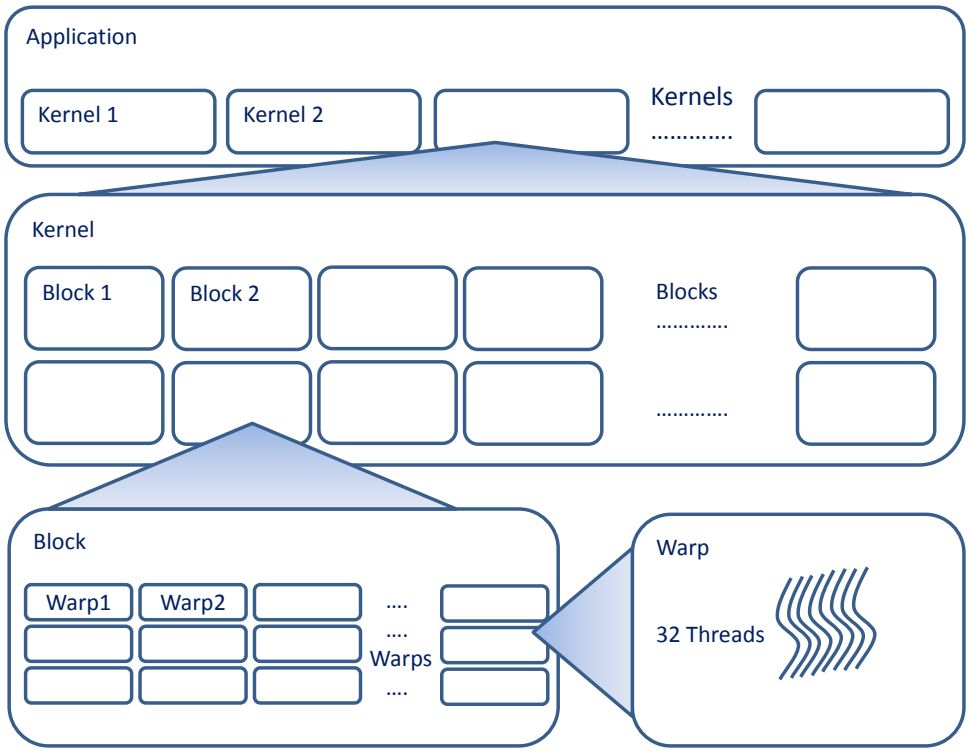


Figure 2: CUDA thread hierarchy

2.2 MODELLING OF BIO-PATHWAYS AS ODEs SYSTEMS

Bio-pathways consist of a network of bio-chemical reactions which govern a variety of fundamental cellular functions. These bio-chemical reactions typically involve molecules colliding with each other and as a result, they either bind together or transform into other types of molecules. Molecules of the same type are called as molecular species. The interactions between different molecular species bring about various complex cellular behaviours. In bio-pathway modelling —depending on the scope of the investigation— one often restricts the focus to a set of bio-chemical reactions that regulate a particular cellular behaviour of interest. Thus bio-pathways enable a systematic understanding of the biological processes. Based on the functions they perform, they can be classified into three categories, namely:

- *Gene regulatory networks* model the interactions between genes.

- *Metabolic networks* describe the mechanisms of energy production and storage within the cell which involve the synthesis and decomposition of complex molecules called metabolites.

- *Signal transduction networks* model the reactions in the cell which are set off in response to an external (or internal) stimuli.

In this thesis, we will mainly focus on signal transduction pathways. The methods we develop are general and applicable to gene regulatory pathways and metabolic pathways as well.

2.2.1 *Ordinary Differential Equations systems*

Traditional modelling approaches for biological systems provide a structural overview of the various molecular species in the system. Nevertheless, for models with a large number of species, quantitative models are required to study the dynamics of different reactions. Bio-pathways can be formalized using a variety of mathematical models, namely ordinary differential equations (ODEs), partial differential equations (PDEs), boolean networks, Petri nets, etc. Depending on the available experimental data, the analysis technique to be carried out and the nature of biological phenomenon under study, one chooses a suitable model that best captures behaviour of the biological system. One of the most widely used formalism for analyzing bio-pathway dynamics is a system of ODEs. The basic idea is to formulate the reactions in the bio-pathway as physicochemical equations [1].

Consider a bio-molecular network with n molecular species and n reactions. Its ODEs system represents the rates of production and consumption of molecular species x_i where $i \in [1, \ldots, n]$, in terms of the kinetic laws that govern each reaction y_j where $j \in [1, \ldots r]$. Typically

these kinetic laws are based on mass-action kinetics [1]. The choice of a kinetics law depends on the nature of the reaction. For instance, the enzyme catalyzed reactions are expressed in terms of Michaelis-Menten equations. We associate a kinetic function f_j to denote the rate of the reaction. As an example, consider the following biomolecular network consisting of 3 species.

$$S_1 + 2S_2 \xrightarrow{V} P$$

Here S_1 and S_2 are the reactants, P denotes the product formed from this reaction. Based on mass-action kinetics, the rate of the reaction V will be $k_1 \cdot [S_1] \cdot [S_2]^2$. Let the quantity k_1 be the kinetic rate constant.

The set of coupled ODEs for the system consists of one equation for each of the variable x_i of the form $\frac{d[x_i]}{dt} = \sum_{j=1}^{r}(p_{ij} \cdot f_{ij})$ where $p_{ij} = 0$ if x_i does not participate in reaction y_j, $p_{ij} = 1$ if x_i is a product in the reaction y_j and p_{ij} = -1 if x_i is a reactant in the reaction.

In our example, the corresponding system of ODEs will be

$$\begin{aligned}
\frac{d[S_1]}{dt} &= -k_1 \cdot [S_1] \cdot [S_2]^2 \\
\frac{d[S_2]}{dt} &= -k_1 \cdot [S_1] \cdot [S_2]^2 \\
\frac{d[P]}{dt} &= k_1 \cdot [S_1] \cdot [S_2]^2
\end{aligned}$$

For large bio-pathway systems, simplifying assumptions can be made in certain cases, as appropriate, to reduce the complexity or size of a model. One such approximation is the Michaelis-Menten approximation [18] to enzyme-substrate kinetics. Figure 3 [1] shows the various steps in the formulation of the ligand-receptor-kinase-substrate pathway as an ODEs system.

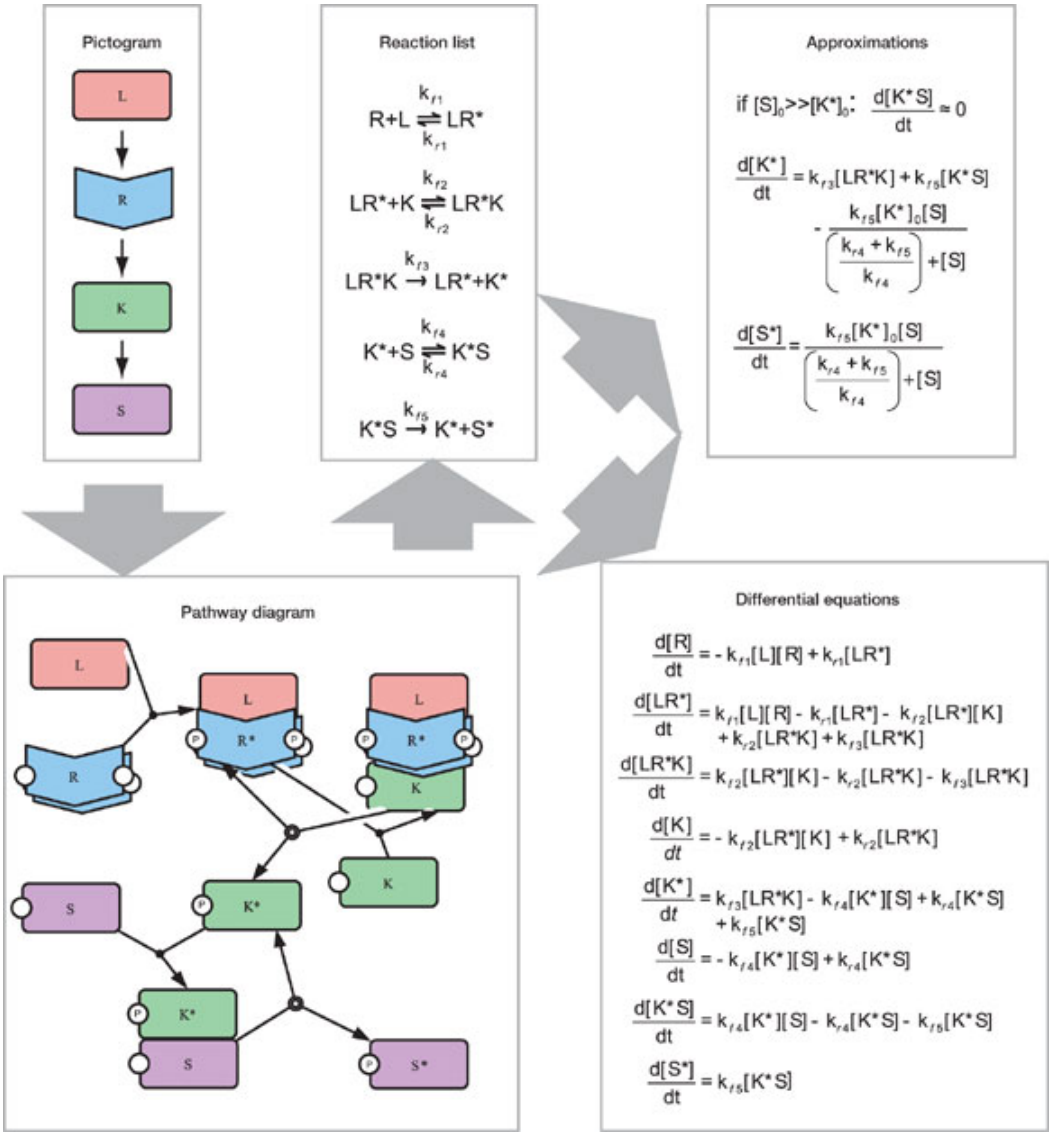


Figure 3: Encoding the ligand-receptor-kinase-substrate pathway as an ODEs system [1]

2.2.2 C^1 *continuity and measure theory*

To secure the mathematical basis for our approximation schemes we will often impose a C^1 continuity assumption. Further this will be shown to lead to a measurable set of trajectories.

Let $\mathbb{N}$ denote the set of non-negative integers. Assume that X and Y are metric spaces [30]. A function $f : X \to Y$ is said to be of class C^k, where $k \in \mathbb{N}$, if the derivatives $f', f'', \ldots, f^(k)$ exist and are continuous. Thus, the class C^0 consists of all *continuous functions* and the class C^1 consists of all *continuously differentiable* functions.

A *σ-algebra* over a set X is a nonempty collection of subsets of X that is closed under complementation and countable unions. The Borel σ-algebra on a topological space X, denoted as $\mathcal{B}_X$, is the minimal σ-algebra containing all the open sets of X.

A *probability space* is a triple $(\omega, \mathcal{F}, \mathbf{P})$ consisting of a set Ω, a σ-algebra $\mathcal{F}$ over Ω, and a function $\mathbf{P} : \mathcal{F} \to [0, 1]$ such that:

(i) $\mathbf{P}(\Omega) = 1$;

(ii) if $\{A_w\}_{w \in W}$ is a countable family of pairwise disjoint sets in $\mathcal{F}$, then $\mathbf{P}(\cup_w A_w) = \sum_w P(A_w)$

Let X and Y be nonempty sets and $\mathcal{M}$ and $\mathcal{N}$ be σ-algebras of subsets of X and Y respectively. A function $f : X \to Y$ is said to be $(\mathcal{M}, \mathcal{N})$-*measurable* if

$$E \in \mathcal{N} \implies f^{-1}(E) \in \mathcal{M} \equiv \{x \in X \mid f(x) \in E\} \in \mathcal{M} \tag{1}$$

Proposition 1 *If X and Y are metric spaces and $f : X \to Y$ is continuous, then f is* $(\mathcal{B}_X, \mathcal{B}_Y)$-measurable.

2.2.3 *ODEs and flows*

Let us assume that there are n molecular species $\{x_1, x_2, \ldots, x_n\}$ in a bio-molecular network. For each x_i, an equation of the form $\frac{dx_i}{dt} = f_i(\mathbf{x}, \Theta_i)$ describes the kinetics of the reactions that produce and consume x_i where $\mathbf{x}$ is the concentrations of the molecular species taking part in the reactions. Θ_i consists of the rate constants governing the reaction. Each x_i is a real-valued function of time $t \in \mathbb{R}$. We assume in this section that all rate constants are known. In what follows, we let $\mathbf{v}$ to range over $\mathbb{R}^n$.

We represent our system of ODEs in the vector form, $\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, \Theta)$ with $F_i(\mathbf{x}, \Theta) := f_i$. In the setting of bio-chemical networks, the expressions in f_i will model kinetic laws such as mass-action and Michaelis-Menten's [31]. Moreover, the concentration levels of the various species will be bounded and the behavior of the system will be of interest only up to a finite time horizon. Hence we assume that f_i is Lipschitz-continuous for each i . As a result, for each $\mathbf{v} \in \mathit{INIT}$ the system of ODEs will have a unique solution $\mathbf{X}_\mathbf{v}(t)$ [20]. We are also guaranteed that $\mathbf{X}_\mathbf{v}(t)$ is a C^0-function (i.e., continuous function) [20] and hence measurable.

For convenience, we define the flow $\Phi : \mathbb{R}_+ \times \mathbf{V} \to \mathbf{V}$ for arbitrary initial vectors $\mathbf{v}$ as $\mathbf{X}_\mathbf{v}(t)$. Intuitively, $\Phi(t, \mathbf{v})$ is the state reached under the ODE dynamics if the system starts at $\mathbf{v}$ at time 0. We work with $\Phi_t : \mathbf{V} \to \mathbf{V}$ where $\Phi_t(\mathbf{v}) = \Phi(t, \mathbf{v})$ for every t and every $\mathbf{v} \in \mathbf{V}$. Again, Φ_t is guaranteed to be a C^0-function (in fact $1 - to - 1$) and Φ_t^{-1} will also be a C^0-function.

2.3 PROBABILISTIC DYNAMICAL MODELS

2.3.1 *Markov chains*

Markov chains are a class of stochastic processes used to model dynamical systems. They are described by a finite set of states and probabilistic transitions where the probability of a transition from a current state to a next state does not depend on any of the previous states. A sequence of states is called as a *path* in the Markov chain.

Definition 1 Consider a stochastic process X_t which takes values from a finite domain $S = \{s_0, s_1, \ldots, s_{\hat{n}}\}$. It is called a *Markov chain* [32] if for all times $t \geq 0$ and all states $s_0, \ldots, s_{\hat{n}} \in S$,

$$P(X_{t+1} = s_j | X_t = s_i, X_{t-1} = s_{t-1}, \ldots, X_0 = s_0) = P(X_{t+1} = s_j | X_t = s_i) = p_{ij}$$

where $s_{t-1}, \ldots, s_0 \in S$; $i, j \in 0, 1, \ldots, \hat{n}$ and $t \geq 0$. p_{ij} denotes the transition probability that the markov chain, if it is in state s_i at time t, transitions in time $t+1$ to state s_j with $p_{ij} \in [0, 1]$ and $\sum_j^{\hat{n}} p_{ij} = 1$. We represent the transition probabilities using the matrix T of order $\hat{n} \times \hat{n}$, whose element $T_{ij} = p_{ij}$. An initial distribution λ^0 is specified over S at $t = 0$. The probability distribution λ^k over S at $t = k$ will be given by $(\lambda^0)T^k$.

2.3.2 *Dynamic Bayesian networks*

Dynamic Bayesian networks are a special class of probabilistic graphical models [33] that extend the notion of Bayesian networks to model dynamical systems. They are used in modelling the evolution of stochastic

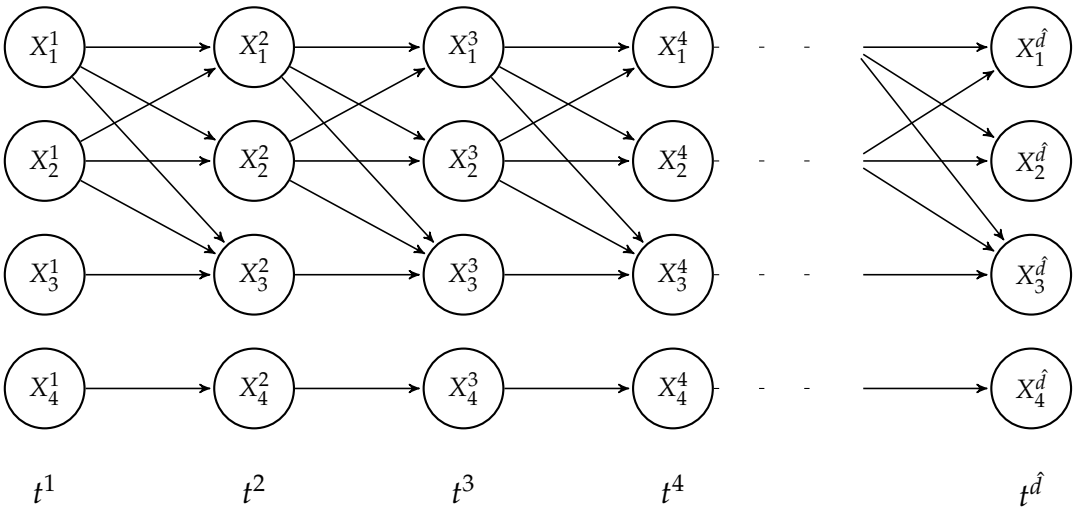
processes whose local states are modelled as random variables. Many varieties of DBNs exist. We deal with a restricted class which are time-variant two-slice dynamic Bayesian networks [21]. They will be of the form $(B_0, \{B^d_{\rightarrow}\}_{d=1}^{\hat{d}}, Pa)$, where B_0 defines the initial probability distributions $\{Pr(\mathbf{X}_i^0)\}$ of the random variables $\{X_i\}_{i=1}^{l}$. And $\{B^d_{\rightarrow}\}$ are two-slice temporal Bayesian networks for the time points $\{t^1, \ldots, t^{\hat{d}}\}$. The nodes of the Bayesian network $B^d_{\rightarrow}$ denoted V^d is given by $V^d = \{X_i^{d-1} | 1 \leq i \leq l\} \cup \{X_i^d | 1 \leq i \leq l\}$ (here we are identifying the nodes with the random variables associated with them). The edge relation E^d will be the subset of $\{X_i^{d-1} | 1 \leq i \leq l\} \times \{X_i^d | 1 \leq i \leq l\}$ satisfying $(X_j^{d-1}, X_i^d) \in E^d$ iff $X_j \in Pa(X_i)$.

As might be expected, $Pa : \mathbf{X} \rightarrow 2^{\mathbf{X}}$ with $\mathbf{X} = \{X_i | 1 \leq i \leq l\}$. Pa assigns a set of parents to each node and satisfies:

- $Pa(X_i^1) = \emptyset$,
- If $X_j^{d'} \in Pa(X_i^d)$ then $d' = d - 1$,
- If $X_j^{d-1} \in Pa(X_i^d)$ for some d then $X_j^{d'-1} \in Pa(X_i^{d'})$ for every $d' \in \{1, \ldots, \hat{d}\}$.

Each node X_i^d will also have a conditional probability table (CPT_i^d) associated with it to specify the local probabilistic dynamics. A typical entry in the CPT_i^d of X_i^d will be of the form $Pr(X_i^d = x | X_{i1}^{d-1} = x_{i1}, \ldots, X_{ij}^{d-1} = x_{ij})$ where $Pa(X_i) = \{X_{i1}, \ldots, X_{ij}\}$.

Thus the way the nodes of the $(d+1)^{th}$ layer are connected to the nodes of the d^{th} layer will remain invariant. However, CPT_i^d will be, in general, different from $CPT_i^{d'}$ if $d \neq d'$. An example of such a dynamic Bayesian network is shown in Figure 4.



$$
\begin{aligned}
Pa(X_1) &= \{X_1, X_2\} \\
Pa(X_2) &= \{X_1, X_2\} \\
Pa(X_3) &= \{X_1, X_2, X_3\} \\
Pa(X_4) &= \{X_4\}
\end{aligned}
$$

Figure 4: Example of a dynamic Bayesian network

2.4 LOGICAL BACKGROUND

Temporal logic [34] can be viewed as an extension of propositional logic with operators that refer to the behaviour of systems over time. A broad range of system properties such as functional correctness, reachability, safety, livenes, fairness, and real-time properties can be expressed using temporal logics. Using this formalism, one can then mathematically check whether the system description is a model of a property expressed in temporal logic. Depending on how time is perceived, temporal logics can be classified into either linear (time is viewed as a single path in which each moment in time has a single successor moment) or branching (time is viewed as a branching tree in which a system could take different paths). Linear temporal logic (LTL) and computation tree logic (CTL) are widely used temporal logic formalisms [35–40].

2.4.1 *Linear temporal logic*

Linear temporal logic was originally developed by Pnueli [35] for reasoning about reactive systems.

The syntax of LTL formulas over the set AP of atomic propositions are defined inductively:

$\varphi := true \mid false \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \mathbf{O}\varphi \mid \varphi_1 \mathbf{U} \varphi_2$

where $a \in AP$.

LTL formulas are interpreted over infinite sequences of sets of atomic propositions of the form $\pi : \omega \rightarrow 2^{AP}$. The semantics for LTL is defined as a language $Words(\varphi)$ that contains all infinite words over the alphabet 2^{AP} that satisfy φ.

The relation $\pi, k \models \varphi$ is defined as follows:

- $\pi, k \models true$, $\pi, k \not\models false$,
- $\pi, k \models$ a for $a \in AP$ iff $a \in \pi(k)$,
- $\pi, k \models \neg\varphi$ iff $\pi, k \not\models \varphi$,
- $\pi, k \models \varphi \vee \varphi'$ iff $\pi, k \models \varphi$ or $\pi, k \models \varphi'$,
- $\pi, k \models \mathbf{O}(\varphi)$ iff $\pi, k+1 \models \varphi$,
- $\pi, k \models \varphi \mathbf{U} \varphi'$ iff $\exists j$, $j \geq k$ such that $\pi, k \models \varphi'$ and $\forall k, k \leq i < j$, $\pi, k \models \varphi$.

The derived propositional operators such as $\wedge$, $\supset$, $\equiv$ and the temporal operators **G**, **F**, follow from the basic operators through the following relations: $\varphi \wedge \varphi' = \neg(\neg\varphi \vee \neg\varphi')$, $(\varphi \implies \varphi') = (\neg\varphi \vee \varphi')$, $(\varphi \equiv \varphi') = (\varphi \implies \varphi' \wedge \varphi' \implies \varphi)$, $\mathbf{F}(\varphi) = true \mathbf{U} \varphi$, $\mathbf{G}(\varphi) = \neg\mathbf{F}(\neg\varphi)$.

π is said to be a model of φ if $\pi, 0 \models \varphi$.

2.4.2 *Bounded linear-time temporal logic*

Bounded linear-time temporal logic (BLTL) is an extension of LTL with time bounds on temporal operators. We introduce the syntax and then the semantics of BLTL formulas.

The syntax of the BLTL formulas over the set AP of atomic propositions is defined as:

- Every atomic proposition as well as the constants $true$, $false$ are BLTL formulas,
- If ψ and $\psi^{'}$ are BLTL formulas then $\neg\psi$ and $\psi \vee \psi^{'}$ are BLTL formulas,
- If ψ is a BLTL formula then $\mathbf{O}(\psi)$ is a BLTL formula,
- If ψ and $\psi^{'}$ are BLTL formulas and $t \leq T$ is a positive integer then $\psi \mathbf{U}^{\leq t} \psi'$ and $\psi \mathbf{U}^{t} \psi'$ are BLTL formulas.

The derived propositional operators such as $\wedge$, $\supset$, $\equiv$ and the temporal operators $\mathbf{G}^{\leq t}$, $\mathbf{F}^{\leq t}$ are defined as before.

The semantics of BLTL is defined with respect to execution traces of the system. In our setting, the semantics of BLTL is defined in terms of the relation $\sigma, t \models \psi$ where σ is a trajectory of the model and $t \in \mathcal{T}$, a finite set of time points $\{0, 1, \ldots, T\}$ and ψ, the property of interest:

- $\sigma, t \models a$ iff $a \in AP$,
- $\neg$ and $\vee$ are interpreted in the usual way,
- $\sigma, t \models \psi \mathbf{U}^{\leq k} \psi'$ iff there exists k' such that $k' \leq k$, $t + k' \leq T$, and $\sigma, t + k' \models \psi'$. Further, $\sigma, t + k'' \models \psi$ for every $0 \leq k'' < k'$,
- $\sigma, t \models \psi \mathbf{U}^{k} \psi'$ iff $t + k \leq T$ and $\sigma, t + k \models \psi'$. Further, $\sigma, t + k' \models \psi$ for every $0 \leq k' < k$.

We will use BLTL as our logic of choice in our analysis techniques. The rationale of choosing BLTL instead of a more sophisticated logic is two-fold. First, relevant properties of bio-pathway models, especially in the context of parameter estimation are linear-time properties defined over a bounded time horizon. Second, BLTL has enough expressive power to characterize properties relating to bio-pathway models while being a very simple temporal logic to work with. Hence we choose BLTL over other commonly-used formalisms, such as continuous stochastic logic and metric temporal logic.

2.4.3 *Probabilistic model checking*

Probabilistic model checkers extend traditional model checking techniques for verifying properties in probabilistic systems. The probabilistic model checking problem can be expressed as: given a probabilistic model M over the set of states S, starting state s_0, temporal logic specification ψ, probability threshold $\theta \in [0,1]$, to decide whether M, $s_0 \models P_{\geq\theta}\psi$. Essentially, in addition to conventional model checking where we check whether a model satisfies a specification of interest, probabilistic model checking verifies whether a property is satisfied with at least a given probability θ. Markov chains and Markov decision processes are widely used models of probabilistic systems while a number of probabilistic flavours of temporal logics: PBLTL [41], PCTL [42] to name a few, have been developed.

In biological and engineering systems, state space explosion renders exact probabilistic model checking methods infeasible for large models. So approximate model checking methods are called for. In approximate probabilistic model checking, a set of execution traces is sampled and then the traces are verified against a property of interest. If the specifi-

cation is evaluated to be true for sufficiently large number of traces, the model checking algorithm decides "yes", otherwise "no".

Numerical solution techniques and statistical analysis methods are two standard approaches employed in probabilistic model checking of stochastic systems. However, as numerical methods [43] are memory intensive, they do not scale well for large systems. On the other hand, statistical analysis methods [44–46] rely on continuous sampling of independent trajectories of the system dynamics. After generating every sample trajectory, one checks whether the given property is satisfied by the sampled trajectory. This process of repeatedly sampling and checking a trajectory continues until a reliable estimate on the probability that the property holds (or does not hold) can be obtained based on statistics of the samples. As a result, such approximate methods obviate the construction of a large probabilistic model and also have low time complexity.

2.5 HYBRID SYSTEMS

Hybrid systems are dynamical systems which involve the interaction of discrete event states and continuous dynamics. For example, a thermostat switching between two discrete states *on* and *off* can be modelled as a hybrid system. The thermostat regulates the temperature in the room according to the continuous dynamics defined by a set of ODEs associated with each of the *on* and *off* discrete states. In systems biology, understanding the dynamics of bio-chemical interactions in large, complex, multi-cellular networks is difficult. To this end, multi-mode hybrid systems can be conveniently used to model a rich class of bio-molecular networks.

2.5.1 *Modelling of hybrid systems*

In order to carry out rigorous analysis on hybrid systems, a hybrid automaton is used as a formal model. As many of the problems in computer science, analysis of hybrid automata is hard due to the high expressive power of the mixed dynamics. The emptiness problem ("Does a given hybrid automaton have a run?") and the reachability problem ("Given a hybrid automaton, does it reach a particular region of the state space?") are undecidable [47] for hybrid automata.

In what follows, let us see how a thermostat system can be modelled as a hybrid automaton as shown in Figure 5. In this thermostat hybrid automaton, we have one continuous variable (room temperature, denoted by $\mathbf{x}$ and taking values in $\mathbb{R}$) and two modes *off* and *on*. Let's assume the the initial mode to be *off*. When the temperature in the room goes below 19 ° C, the thermostat switches to *on* mode as per the transition that is enabled and evolves according to the continuous dynamics associated with the *on* mode. Now when the temperature in the room goes above a certain threshold of 20 ° C the hybrid automaton takes the transition to the *off* mode and continues to evolve in this new mode.

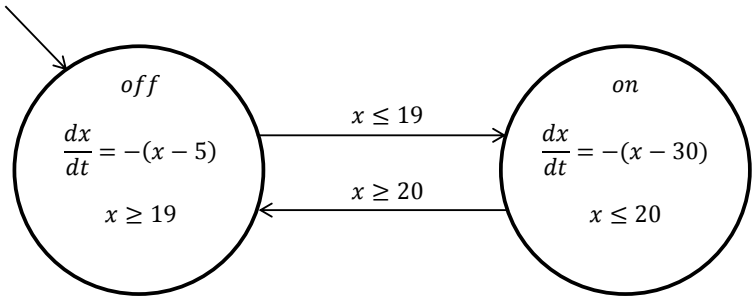


Figure 5: A two-state thermostat hybrid system

3

DBN APPROXIMATION BASED VERIFICATION OF ODEs

In [22], it was shown how an ODEs system can be approximated as a dynamic Bayesian network. Here we show how this approximation based on generating a large number of trajectories can be parallelized via a GPU implementation.

We first recall how an ODEs system can be approximated as a DBN. We then describe how the construction of this DBN can be parallelized for implementation on GPUs by exploiting the fine-grained parallelism in the computation of a trajectory of the ODEs system.

3.1 DBN APPROXIMATION OF A SYSTEM OF ODEs

The dynamics of a bio-pathway is often modelled as a system of ODEs with one equation of the form $\frac{dx}{dt} = f_i(\mathbf{x}, \mathbf{p})$ for each molecular species x in the pathway. Here f describes the kinetics of the reactions that produce and consume x and x are the molecular species taking part in these reactions whereas p are the rate constants governing these reactions. For large pathways, this ODE system which will typically have many unknown parameters will be difficult to calibrate and analyze. To get around this an approximation scheme was developed in [48] through which a system of ODEs can be reduced to a DBN.

1. First, we assume the states of the system are observed only at a finite number of time points, $\{0, 1, ..., T\}$. Next, the range of each

variable x_i (rate constant r_j) is partitioned into a set of intervals I_i (I_j). Both these discretizations are motivated by the fact that experimental data will be available only for a finite set of time points and this data will be of limited precision.

2. Next, the initial values of the variables as well as the rate constants are assumed to be distributions (usually uniform) over certain of these intervals.

3. We then sample the initial states of the system according to this distribution sufficiently many times, and generate a large number of trajectories by numerical integration for each sampled initial state.

4. The resulting set of trajectories is then treated as an approximation of the dynamics of the ODE system.

To handle unknown rate constants we assume that the minimum and maximum values of these constants are known. We then partition these ranges of values also into a finite numbers of intervals, and fix a uniform distribution over all the intervals. After building the DBN, we use a Bayesian inference-based technique to perform parameter estimation to complete the construction of the model. However, unlike the variables, once the initial value of an unknown rate constant has been sampled, this value will not change during the generation of a trajectory. Naturally different trajectories can have different initial values for an unknown rate constant.

A key idea is to compactly store the generated set of sequences as a DBN. This is achieved by means of a simple counting procedure that exploits the network structure. In order to keep the focus on the approximation procedure we give only an informal description of DBNs here.

3.1.1 *The DBN structure*

A DBN consists of a directed acyclic graph where the nodes are grouped into layers with each layer representing a time point [33]. The nodes in layer $t-1$ will be connected to the nodes in the layer t in the same way as t ranges from 1 to T. Each node will have a random variable associated with it. In our setting, there will be one random variable ${x^t}_i(r_j^t)$ corresponding to each variable x_i (unknown rate constant r_j) to capture in which interval the value of x_i (r_j) falls at time t. Further, for each unknown rate constant k, we add the equation $\frac{dk}{dt} = 0$ to capture the fact that once the value of k has been sampled, this value will not change during the numerical integration of a trajectory.

$Pa(x_i^t)$, the set of parent nodes of x_i^t is determined as follows. The node $x_k^{t-1}(r_j^{t-1})$ will be in $Pa(x_i^t)$ iff $x_k(r_j)$ appears in the equation for x_i or $x_k = x_i$. On the other hand, r_j^{t-1} will be the only parent of the node ${r^t}_j$ in case r_j is an unknown rate constant. In Figure 6, we show a simple enzymatic reaction network, its ODE model and the structure of its DBN approximation. In this example, we have assumed that k_3 is the only unknown parameter.

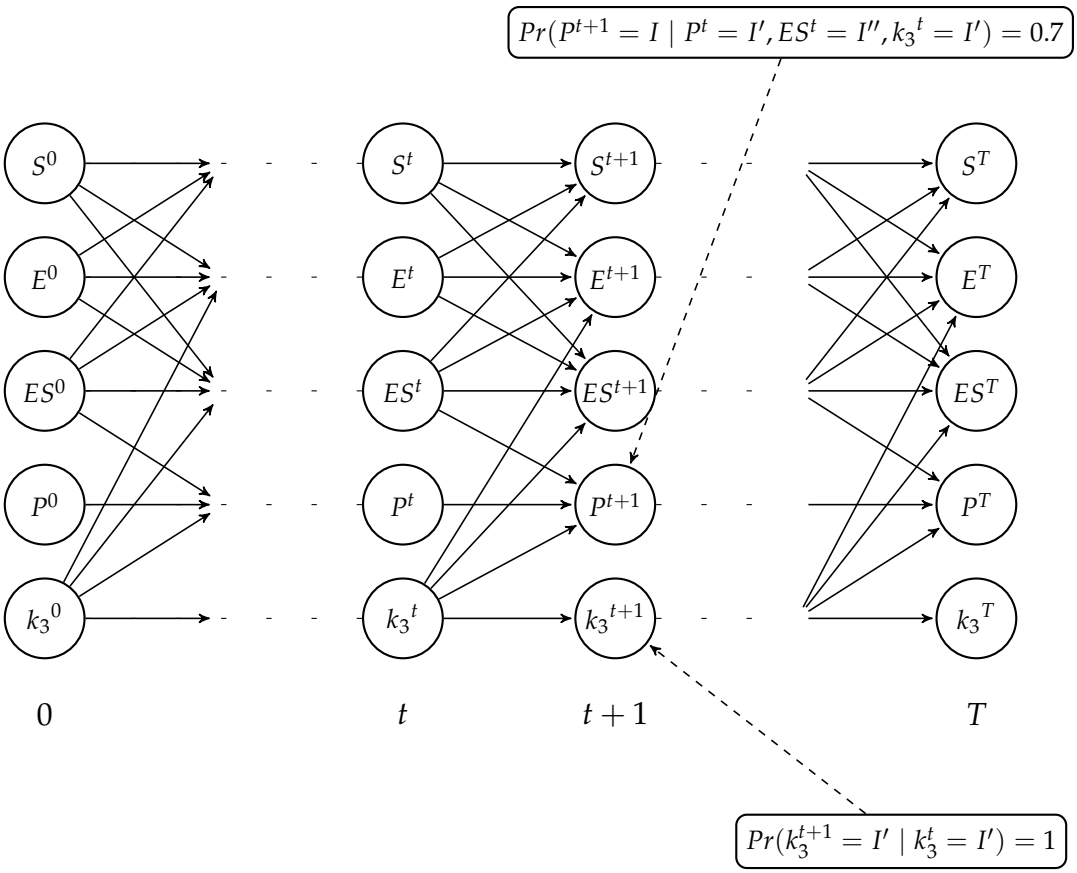
As indicated in Figure 6(c), each node will also have a conditional probability table (CPT) associated with it to specify the local probabilistic dynamics. A typical entry in the CPT of x_i^t will be of the form $Pr(x_i^t = I | z_1^{t-1} = I_1, z_2^{t-1} = I_2, \ldots, z_l^{t-1} = I_l) = p$ with $Pa(x_i^t) = \{z_1^{t-1}, z_2^{t-1}, \ldots, z_l^{t-1}\}$. Such an entry means that p is the probability that the value of x_i falls in the interval I at time t, given that the value of z_u was in I_u at time $t-1$ for each z_u^{t-1} in $Pa(x_i^t)$. The probability p is calculated through simple counting in we call a binning step of the approximation. Suppose N is the number of generated trajectories. We record the number of the trajectories from this collection for which their

$$S + E \underset{k_2=0.2}{\overset{k_1=0.1}{\rightleftharpoons}} ES \xrightarrow{k_3} E + P$$

(a)

$$\begin{aligned}
\frac{dS}{dt} &= -k_1.S.E + k_2.ES \\
\frac{dE}{dt} &= -k_1.S.E + (k_2 + k_3).ES \\
\frac{dES}{dt} &= k_1.S.E - (k_2 + k_3).ES \\
\frac{dP}{dt} &= k_3.ES \\
\frac{dk_3}{dt} &= 0
\end{aligned}$$

(b)



(c)

Figure 6: (a) Enzyme catalytic reaction network (b) ODEs model (c) Dynamic Bayesian network

value of z_u fell in the interval I_u for each z_u in $\{z_1, z_2, \ldots, z_l\}$ at time $t-1$. Suppose this number is J. We then determine for how many of these J trajectories, the value of x_i fell in the interval I at time t. If this number is J', then p is set to be J'/J.

If k is unknown, in the CPT of k^t we will have $Pr(k^t = I|k^{t-1} = I') = 1$ if $I = I'$ and $Pr(k^t = I|k^{t-1} = I') = 0$ otherwise. This is because the sampled initial value of k does not change during numerical integration. Suppose k appears on the right-hand side of the equation for x and $Pa(x_i^t) = \{z_1^{t-1}, z_2^{t-1}, \ldots, z_l^{t-1}\}$ with $z_l^{t-1} = k^{t-1}$. Then for each choice of interval values for nodes other than k in $Pa(x_i^t)$ and for each choice of interval value $\hat{I}$ for k there will be an entry in the CPT of x^t of the form $Pr(x_i^t = I|z_1^{t-1} = I_1, z_2^{t-1} = I_2, \ldots, k = \hat{I}) = p$. This is so since we will sample for all possible initial interval values for k and $k^0 = k^{t-1}$. In this sense, the CPTs record the approximated dynamics for all possible combinations of interval values for the unknown rate constants. These features are illustrated in Figure 6(c) for the unknown rate constant k_3.

Based on the constructed DBN, one can efficiently analyze the dynamics of the pathway under study using standard probabilistic formal verification methods [22]. In what follows, we present the GPU implementation of the DBN construction.

3.1.2 *Related work*

A survey of hardware accelerators, including GPUs, for systems biology applications is presented in [49]. In [50], a Python language based package called *cuda-sim* enables accelerated simulations of bio-chemical network models on GPUs. Further, a variety of previous schemes have been devised to improve the performance of GPU implementations. Of particular relevance to our work are the data prefetching and memory latency

hiding techniques [51–55]. However, these techniques are not applicable in our context, as they rely on a large ratio between computation and the size of the dataset prefetched into the on-chip shared memory. Another problem often affecting performance is the relationship between the kernel geometry and the layout of the data to be processed. In general, the selection of the number of parallel threads is correlated with data placement, and identifying a solution is not trivial [56]. In contrast, our framework goes beyond traditional data tiling [57] and introduces an additional level of flexibility in thread scheduling that allows for changes in the kernel computation without affecting data placement. Our approach extracts fine-grained parallel code from the bio-pathway model and distributes it across a number of concurrent threads [58]. Other GPU code generation schemes utilize heterogeneous collaborative threads [59, 60]. However, these schemes have only been directed to segregate slow global memory accesses into separate threads, thereby freeing dedicated computation threads from such accesses. Our method goes beyond these schemes and introduces multiple classes of dedicated compute threads.

3.2 GPU IMPLEMENTATION OF THE APPROXIMATION

We now describe how our approximation algorithm is implemented in GPUs. Recall each GPU unit consists of a number of streaming multiprocessors (SMs). Each SM in turn consists of a large set of registers, a number of execution cores, and a scratchpad memory that is shared by all warps allocated to the SM. Threads are grouped into scheduling units called warps, consisting of threads executing in lockstep. However, the threads belonging to warp must execute the same instruction. If not, they will be serialized. Each SM computes a set of trajectories

and records the number of times these trajectories hit the intervals of values of the variables at different time points. This binning information is stored in a specific area of the global memory which will be summed up to produce the CPTs of the DBN. We now describe how the computation within a single SM is orchestrated according to the scheme.

3.2.1 *The GPU computation pipeline*

The GPU computation steps are shown in Figure 7. Starting from an initial state at $t = 0$, for each time interval Δt, the new value of a variable x is determined by applying numerical integration using the current values of the variables and the values of the rate constants appearing in the ODE for x as well as the current value of x. Since trajectories are generated through numerical integration, to ensure numerical accuracy, each interval $[0, \Delta t]$ is uniformly subdivided into r sub-intervals for a suitable choice of r. We compute an updated value of the variables every $\tau = \frac{\Delta t}{r}$. Each variable may appear in multiple equations, leading to a large amount of read-sharing. To ensure consistency, all variables are updated together in an atomic transaction. We use a fourth-order Runge-Kutta algorithm to compute the next value of a variable for each time step. Overall, each trajectory is numerically simulated for $r \cdot T$ steps. Finally, the current values of the variables sampled at each of the time points $\{0, \Delta t, \ldots, T.\Delta t\}$ are used to count how many of the trajectories hit a particular interval of values for each variable at that time point. These counts are then used to derive the entries in the CPTs of the DBN. As described earlier, there will be one CPT for each variable and each time point of interest. Each CPT will have $|I_i|^{|Pa(x_i^t)|+1}$ entries where I_i is the set of intervals associated to the variable and $Pa(x_i^t)$ is the set

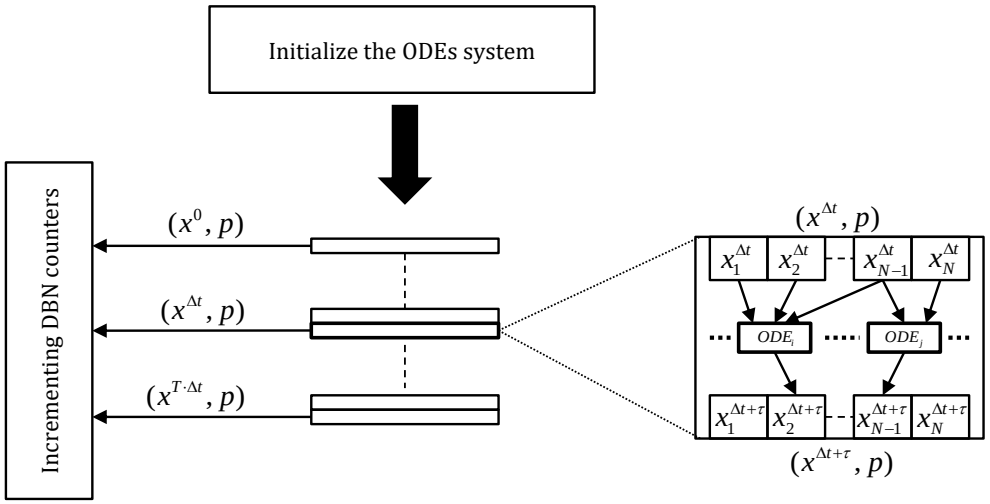


Figure 7: Computation steps of a trajectory showing the Runge-Kutta integration step

of parent nodes of x_i. It is important to note that $|Pa(x_i^t)|$ will almost always be much smaller than the number of variables in the system.

Due to the coupling between the variables, the entire front of the new values of all the variables must be computed at each time step per trajectory. If we naïvely allocate as many threads as possible to each SM with each thread computing a trajectory then their memory requirements will exceed the size of the (fast) local memory of the SM. Thus, for high-dimensional systems, the global memory has to be used to store the intermediate data. However, this leads to a vicious cycle in which more parallel threads have to be launched to hide the memory latency that in turn creates more accesses to the global memory, leading to memory bandwidth saturation and eventually to performance degradation. It is also important to note that since the ODEs are not identical, the resulting threads will be heterogeneous.

To get around this we devise an execution strategy based on fine-grained parallelism and heterogeneous threads tuned to the GPU architecture. Briefly, we partition the set of equations into blocks and allocate each block to a thread. Thus a single trajectory will be computed by a set of threads $\mathcal{C}$. Each member of $\mathcal{C}$ will handle a different block of equations and compute the new values of the variables appearing on the left-hand sides of these equations. The binning process executes in parallel, during the subsequent Δt iteration, using the memory access threads ($\mathcal{M}$), which will store the results in a large table located in global memory.

Many copies of the $\mathcal{C}$ and $\mathcal{M}$ groups of threads will be assigned to an SM. How they are scheduled is guided by the hardware organization of the SM. In what follows, we first describe the generic code generation scheme that extracts fine-grained parallel code from the ODEs model and distributes it across $\mathcal{C}$ threads. Then we give details of its application to the DBN approximation method.

3.2.2 *The heterogeneous code generation framework*

The code generation scheme described in this section forms the basis of the trajectory simulation procedure applied to different systems throughout this thesis.

We first present a review of the GPU architecture and its impact on performance. Essentially, in a GPU:

1. A large number of threads must be instantiated to obtain the maximum performance,

2. There is a warp-level affinity for lock-step execution (a more relaxed form of SIMD),

3. The amount of fast(local) shared memory is limited.

It is the programmer's responsibility to exploit the parallelism in the application through the programming model in order to satisfy the first requirement. However, this will often conflict with the other requirements. With a large number of threads instantiated, the shared memory quota for each thread is a small number of bytes, and often the user has to identify opportunities for data sharing across threads to achieve efficient execution.

Serialization occurs, with the accompanying penalty, when there is control flow divergence within a warp. Therefore, the programming model calls for as little divergence as possible. This leads, in general, to a particular type of data processing that we call homogeneous computing, in which loops are unrolled and distributed over the entire thread grid. Algorithm 2 describes this approach for a standard CPU pseudocode shown in Algorithm 1.

Algorithm 1 CPU computing model

```
1: for (i = 0; i < N_i; i++) do
2:     for (j = 0; j < N_j; j++) do
3:         code_0(i, j);
4:         ...
5:         code_{C-1}(i, j);
6:     end for
7: end for
```

Algorithm 2 Conventional GPU computing model

1: **for** `(i = 0; i <` N_i/Θ`; i++)` **do**
2: **for** `(j = 0; j <` N_j/Π`; j++)` **do**
3: **for** $(\theta \leq \Theta, \pi \leq \Pi)$ **do** in **parallel**: {
4: $code_0(i \cdot \Theta + \theta, j \cdot \Pi + \pi)$;
5: ...
6: $code_{\mathcal{C}-1}(i \cdot \Theta + \theta, j \cdot \Pi + \pi)$;
7: }
8: **end for**
9: **end for**
10: **end for**

In this code, N_i, N_j, Θ and Π allow for arbitrary geometric shapes of the loop structure. The loop body is formed of $\mathcal{C}$ code segments. We will discuss the significance of this in our context in Subsection 3.2.3. The execution does not diverge, as all threads execute the same homogeneous computation for different datasets. It is important to ensure that the product $\Pi \cdot \Theta$ is high enough so that enough GPU threads are utilized. However, we need to consider other details of the GPU architecture. Often as the loops are unrolled and launched on the GPU, in addition to the divergence problem, strict memory usage limitations imposed by the GPU architecture would mean one has to carefully tackle the memory management problem. In particular, it is desirable that all data accessed during the parallel execution is located in the SM memory.

In contrast to the homogeneous approach above, our code generation scheme is built on the insight that there is no penalty when threads in different warps diverge —as long as those in the same warp do not. Therefore, the key concept behind our code generation scheme is to look for fine-grained parallelism, within the loop body, and identify in-

dependent code segments that can be executed in parallel. Assuming that $code_0, code_1, \ldots, code_{\mathcal{C}-1}$ are independent, we place these segments in threads that belong to different warps in a heterogeneous computing model. Obviously, some amount of loop-level parallelism is still necessary to fill each warp with similar threads. Therefore, we choose to partially unroll only the outer loop in Algorithm 3.

Algorithm 3 Heterogeneous GPU computing model

1: **for** `(i = 0; i <` N_i/Θ`; i++)` **do**
2: **for** `(j = 0; j <` N_j`; j++)` **do**
3: **for** $(c \leq \mathcal{C}, \theta \leq \Theta)$ **do** in **parallel**:
4: $code_c(i \cdot \Theta + \theta, j)$;
5: **end for**
6: **end for**
7: **end for**

In this implementation, the number of threads is determined by $\mathcal{C} \cdot \Theta$. In addition, to ensure that threads with similar control flow can be grouped in each warp of size, W_{size}, $\exists w \in \mathbb{N}, \Theta = w \cdot W_{size}$.

When compared to the homogeneous approach, the main advantage is derived from the lower amount of unrolling, which for certain applications may significantly decrease the memory requirements. It is important to observe that the proposed code transformations do not affect the inner loop. This allows us to optimize even for the case where the iterations of the inner loop are not independent.

We have described a scheme where data resides only in the shared memory. However, the input and output of the application must be transferred from/to global memory. Due to the long latency of global memory, any such transfer suffers a large delay of up to 400 cycles, during which the requesting thread (and its associated warp) must stall. By default, the GPU architecture replaces the stalled warp with another

available warp. This approach relies on a high enough computation to memory transfer ratio such that alternative warps are available. If the global memory transfers are scattered across all warps, the memory access delay will impact all threads. Instead, our code generation scheme prefetches from the global memory within a few specialized memory access warps $\mathcal{M}$, handling these transfers in parallel and without interfering with the execution of the other $\mathcal{C}$ warps [60].

Our scheme attains the optimal GPU performance only if the amount of computation in each code segment is balanced such that the GPU pipeline is always full. Otherwise, some of the warps will finish processing early, whereas the remaining warps are not capable of ensuring sufficient GPU occupancy to fill the GPU pipeline. Our code generation scheme distributes fine-grained computation blocks extracted from the loop body among code segments located in different warps and obtains feedback regarding the quality of the computational balance and pipeline occupancy by analyzing the PTX assembly generated by the CUDA compiler.

The loop body consists of a list of instructions corresponding to an integration step for each variable. We can cluster these instructions into groups that exhibit only inter-iteration dependencies, because each integration step is independent of the others. These clusters are $(eq_0, eq_1, \ldots eq_n)$. We initially compile the entire loop body as a single thread and analyze its PTX assembly, obtaining the number of PTX instructions in each cluster i as $\text{PTX}(eq_i)$. We use this information to determine how to place these code clusters across threads in order to balance the pipeline occupancy.

Given the throughput stated in the documentation of the GPU for each arithmetic operation, we model the number of cycles required to issue each PTX instruction in the GPU pipeline. The pipeline has a la-

tency of 22 cycles, and multiple warps are multiplexed by the GPU hardware to issue continuous instructions on the pipeline. The Tesla 2.0 architecture supports the simultaneous execution of two half-warps, each of them utilizing half the number of compute cores available. For single-precision floating point instructions, the pipeline occupancy analysis is equivalent to the assumption that a single full warp is processed at a time. By compiling the code for *"fast math"*, we also ensure that the PTX instructions in the compiled code directly match the operations supported by the architecture.

Using the earlier assumptions, we model, for example, floating-point add instructions across one warp as being issued in a single cycle, whereas *div* instructions are issued within eight cycles. We use the notation $issue(div) = 8$. We also model the timing of the *ld* and *st* instructions that access the shared memory. With proper data alignment, all shared memory banks are utilized. Because of the inherent architectural two-way conflict on shared memory banks, a memory access is issued every two cycles. The pipeline occupancy represents the fraction of the execution cycles where a new operation is issued. For a code segment of size $PTX(code)$ instructions, this occupancy is calculated as:

$$o(code) = \frac{\sum_{i \in PTX(code)} issue(i)}{22 \cdot |PTX(code)|} \tag{2}$$

Our code generation scheme has two objectives:

- to ensure that all $\mathcal{C}$ warps have a balanced number of instructions, and
- to ensure that the pipeline occupancy achieved by summing the occupancy induced by each warp exceeds (but is close to) 1.

Because the GPU has a fixed latency pipeline and we avoid global memory accesses, when the occupancy is 1 or below, the number of

instructions corresponds to the latency of their execution. Additional threads beyond an occupancy of 1 will queue for execution and lead only to additional register pressure and subsequent performance degradation. We estimate how many threads $\mathcal{C}$ are required to occupy the pipeline by analyzing the average occupancy across all code segments: $\mathcal{C} = \lceil \frac{1}{o(eq_0 \cup eq_1 \cup ... \cup eq_n)} \rceil$. This is a reasonable approximation because distributing the code over $\mathcal{C}$ threads increases the occupancy $\mathcal{C}$ times. We chose which clusters to allocate to each code segment $code_i$ such that $|PTX(code_i)| = \frac{|PTX(eq_0 \cup eq_1 \cup ... \cup eq_n)|}{\mathcal{C}}$. We employ a greedy allocation, where instruction clusters are allocated in sequence to each code segment.

3.2.3 *Mapping to the GPU architecture*

We now apply our heterogeneous code generation scheme for the DBN approximation of a system of ODEs. For generating each trajectory, the dependencies between the variables in the system of ODEs require the entire front of variables belonging to each trajectory to be computed together. Hence, we have to store the data in the shared memory. By doing so, we prevent saturating the global memory bandwidth. For each time interval, the value of each variable x at the end of the interval is determined by applying a Runge-Kutta numerical integration using the current value of x and the current values of other variables (and parameters) appearing in the ODE for x.

The computational pattern for each trajectory matches the loop body of the heterogeneous computation scheme. We show the data movement during one computation step in Fig 8. The equations are valid instruction clusters and are distributed into compute threads $\mathcal{C}$ that will collaborate to generate a single trajectory. This entails sharing of the variables and parameters within a group. The number of threads in such

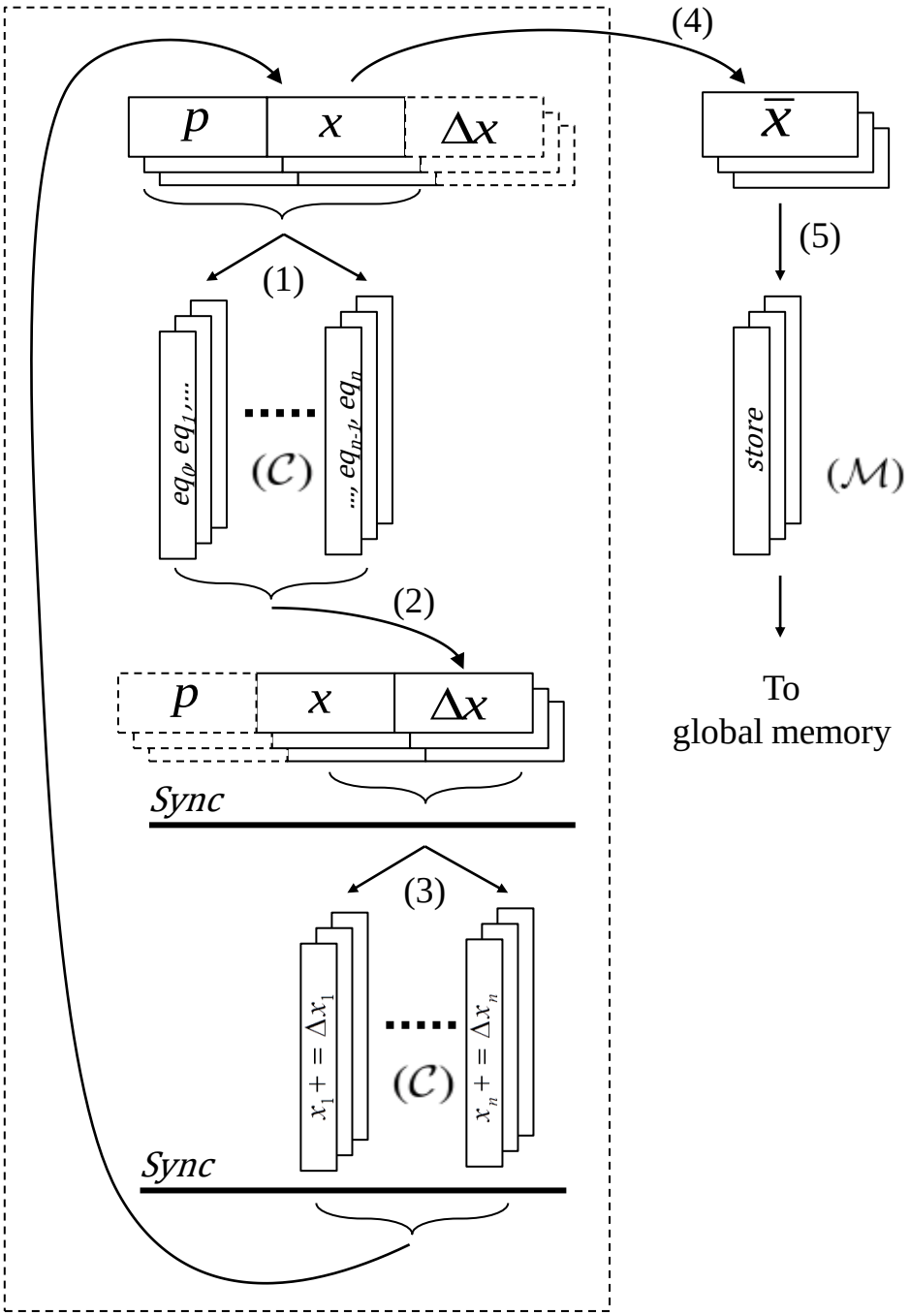


Figure 8: Data movement in a single simulation step

a group is $\mathcal{C}$. These threads read the current values of the variables (x) and parameters (p) appearing in the equations allotted to it from the local memory of the SM (step (1)). The Δx changes during a time step are computed in parallel and are stored back to local memory (step (2)). The vector of variables x is then updated (step (3)). This process is applied iteratively for each time step (of duration $\tau = \frac{\Delta t}{r}$). The Θ trajectories are computed in parallel to satisfy the lock-step requirements of the GPU architecture. Each trajectory requires $N_j = T \cdot r$ integration steps. The trajectory computation process is repeated for all N trajectories, hence $N_i = \frac{N}{\Theta}$.

In the binning steps of the approximation method, the number of times the threads hit the various intervals of values of the variables are counted. To do this, the vector x is replicated as $\bar{x}$ (step (4)). The binning process executes in parallel, during the next Δt iteration, using the memory access threads $\mathcal{M}$, which will store the results in a large table located in global memory (step (5)). This will ensure that the numerical integration can continue during the binning process, which has long latency memory operations.

As a departure from the code generation scheme described previously, we require additional synchronization among the $\mathcal{C}$ threads after each integration step. These $\mathcal{C}$ threads belong to several warps; hence, they are scheduled independently, and their execution may not be synchronized. Synchronization is achieved by a partial synchronization primitive available since the Tesla 2.0 architecture. The *bar.sync PTX* instruction allows for an explicit number of threads to be waited for at the barrier. The number of threads may be smaller than the total number of threads executing on the GPU. Once all of the $\mathcal{C}$ threads arrive at the barrier, they proceed to the next integration step.

The vectors x, p, Δx, and $\bar{x}$ together form the workset of the trajectory. All threads of each SM have access to the dedicated SM memory, which is similar to a scratchpad [61]. To ensure that enough parallel threads can be instantiated, the computation of each trajectory is unfolded onto the $\mathcal{C}$ threads. This enables a reduction of the number of trajectories being processed concurrently. In this way, the total memory footprint, consisting of the worksets of all Θ trajectories being computed in a SM, can be kept within the limit of the available SM memory.

Finally, the GPU architecture requires all threads belonging to a warp to have matching control flow in order to achieve the highest performance. Otherwise, the threads' execution will be serialized. Accordingly, we organize the $\mathcal{C}$ threads belonging to each trajectory so that threads executed together in the same warp process the same subset of model equations from different trajectories. Given a warp size of 32 threads, this eliminates the control flow divergence in each warp if $\exists w, \Theta = w \cdot 32$.

However, Θ is constrained by the SM memory capacity to $\Theta = \frac{SM_{size}}{workset_{size}}$. Therefore, it is not always feasible to instantiate a sufficient number of parallel trajectories in order to completely fill each warp with $\mathcal{C}$ threads having similar control flow. In this case, we have chosen to fill the rest of the warp with threads that belong to the next equation group. This ensures the best utilization of the GPU register pool. However, to maintain warp boundaries, we decrease the number of trajectories to the immediately lower number that matches the equation $\exists \delta \geq -\log_2(W_{size}), \Theta = W_{size} \cdot 2^{\delta}$. If a warp contains multiple sets of threads, their execution is serialized, and we can model the combined warp as if several warps were executed. Ideally, the total number of issue cycles for all $\mathcal{C}$ warps in the CUDA code has to match the pipeline length to ensure full GPU occupancy. In contrast, when $\Theta > W_{size}$, mul-

tiple warps may encapsulate the same code segment, and we determine $\mathcal{C}$ as follows:

$$\mathcal{C} = \frac{22 \cdot PTX(eq_0 \cup eq_1 \cup \ldots)}{\sum_{i \in PTX(eq_0, eq_1, \ldots)} issue(i) \cdot \lceil \Theta / W_{size} \rceil}$$

The overall orchestration of the application on each SM is shown in Figure 9. Instructions belonging to $\mathcal{C}$ warps are multiplexed onto the GPU pipelines. All of the GPU pipelines execute in lock-step. $\mathcal{M}$ threads are scheduled from time to time to transfer data to the global memory. The specialized warps accessing the global memory (GM) are subject to delays of up to 400 cycles. $\mathcal{M}$ threads are grouped together into specialized memory access warps such that they will not interfere with the $\mathcal{C}$ threads' executions. The same orchestration is replicated on all SMs of the GPU. This can be easily implemented by computing a fraction of the total number of trajectories on each SM.

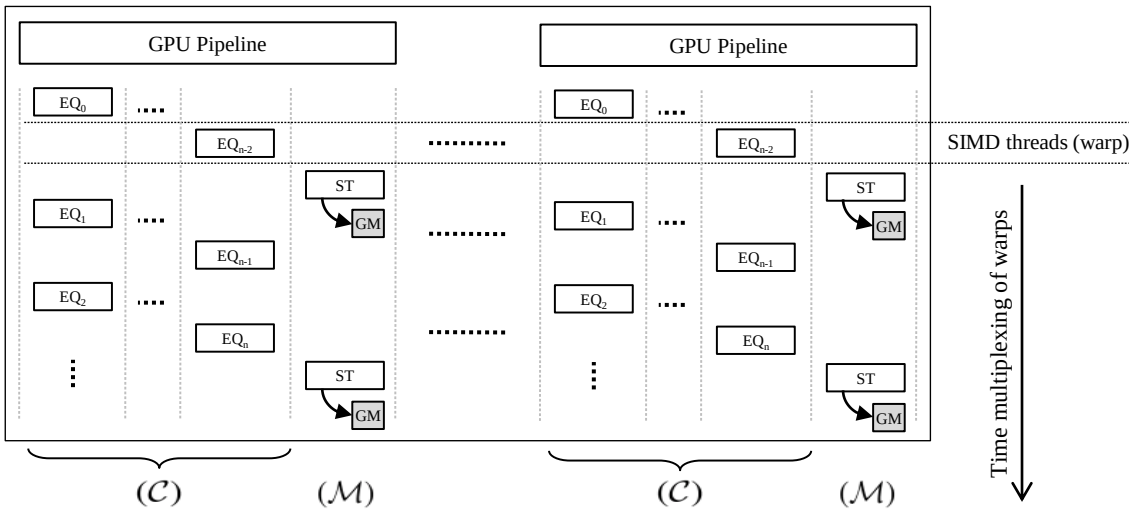


Figure 9: Concurrent execution of trajectories inside an SM

For each trajectory, we generate the initial states using a Mersenne twister algorithm based on the MT 19937 random number generator [62]

running in each of the $\mathcal{C}$ threads. This algorithm utilizes a large table stored in the global memory. Considering that this initialization step is done only once during the generation of a trajectory, the overhead due to storing this table in global memory is minimal.

The repetitive Runge-Kutta numerical integration process is at the heart of the trajectory simulation algorithm. We used a fourth-order Runge-Kutta algorithm that requires each equation to be applied four times as part of the integration step.

The code generation scheme produces the corresponding code for each equation and passes it to the CUDA compiler. The PTX assembly is analyzed using the previously described model to extract timing information for each equation. Our algorithm then distributes the equations so that the corresponding timing is balanced among the $\mathcal{C}$ threads. Because we utilize a small number of threads, register pressure is low and there are no spills to local memory, hence avoiding any additional delay.

By carefully considering the balance between computation load, data supply needs, and local resources available, we show in the following section, that using our code generation scheme for GPUs, one can obtain a 3.9× speed-up compared to a conventional GPU implementation.

3.3 RESULTS

We have implemented the scheme described above and have used it to generate CUDA code that was compiled for NVIDIA Tesla 2.0 ('Fermi') platforms using the CUDA 4.0 runtime. The target GPU is a S2050 at 1.15GHz with 2GB of memory. To evaluate the performance of our GPU-based implementation, we utilized three realistic pathway models that tested various features of our scheme as shown in Table 1. We chose the

number of trajectories such that the resulting DBN approximation was of sufficient good quality and that runtimes were sufficiently long.

Figure 10 shows the reaction network for the EGF-NGF model. The values for the parameters of this model taken from [2] are known. For our experiments, we have set a subset of the parameters as "unknown" in each model and constructed the DBN approximation accordingly. The same was done to two other pathway models, namely thrombin dependent MLC phosphorylation pathway [63] and segmentation clock network [64].

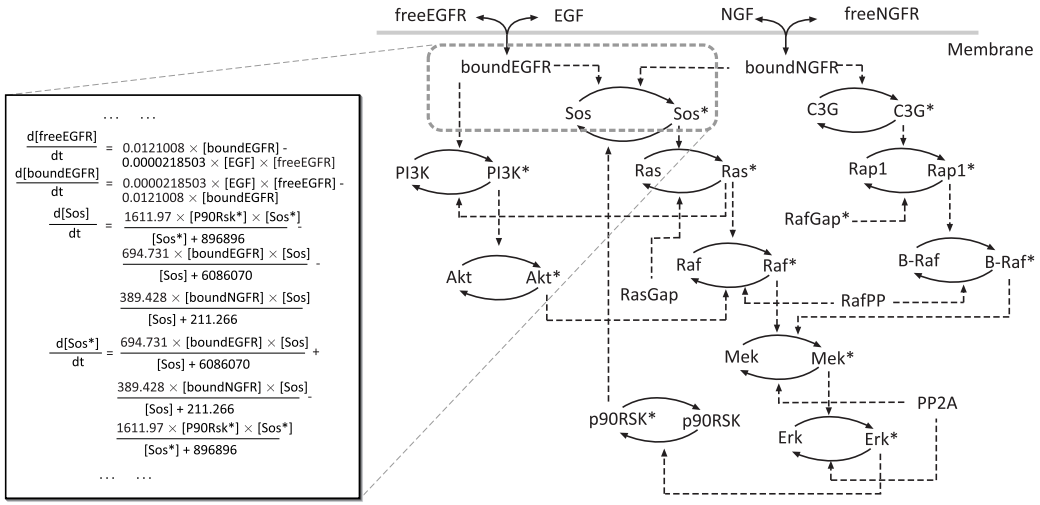


Figure 10: The reaction network diagram of the EGF-NGF pathway [2]

For each model, we listed the number of variables ($|x|$), the number of unknown parameters ($|p|$), the simulation time step (Δt), the number of time intervals (T), the number of integration sub-intervals (r), and the total number of trajectories (N) as shown in Table 1. We also listed the average number of operators within each model equation, as well as the distribution of each operator's type. For all models, the range of each variable and unknown parameter was discretized into five intervals of equal size. A smaller number of trajectories were computed for the larger models to keep the execution times within reasonable limits.

Model	$\|x\|$	$\|p\|$	Δt	T	r	N	Avg. Ops	+/-	$\times$	$\div$
EGF-NGF	32	20	6	100	100	10^6	7.4	87	106	44
Segmentation clock	22	40	300	100	500	10^6	11.9	67	91	33
Thrombin	105	164	2	100	2×10^4	3×10^4	13	419	942	2

Table 1: Characteristics of the models

The following evaluation strategy was used. We implemented the target application using both a homogeneous computation approach (where the workset is stored in global memory, as the datasets do not fit the shared memory) and our proposed heterogeneous approach. To emphasize the efficiency of the proposed flow, we characterized a broad design space by varying the number of threads of both homogeneous and heterogeneous schemes, producing a large spectrum of kernel geometries. For the homogeneous implementation, we varied the thread block size, whereas for the heterogeneous implementation, we varied $\mathcal{C}$, the number of threads collaborating to generate a trajectory.

In addition to an overall performance evaluation of our framework, we will show the contribution of each component of the framework: the proposed heterogeneous thread execution scheme, the separation of the GM accesses, and the load balancing.

Figure 11 shows a comparative design-space exploration for the three models we considered. We compare the performance of both the homogeneous and the heterogeneous implementations. For the graphs depicting the homogeneous scheme, the x-axis represents the total number of warps in a thread block, whereas for the graphs illustrating the heterogeneous approach, it represents the number of $\mathcal{C}$ threads. The performance is measured in trajectories computed per second, along the y-axis. The performance of the homogeneous implementation ends up always being lower, as it is bound by the GPU memory bandwidth. In addition, this performance cannot be trivially estimated, as it depends

on many factors such as the global memory bandwidth, GPU occupancy, and register pressure. Large performance variations are observed when the number of threads (warps) is varied.

In contrast, our heterogeneous scheme has a predictable as well as significantly higher performance. For all benchmarks, performance increases steadily as more parallel code segments are created.

A single code segment, containing all the ODEs (the first point in each graph for the heterogeneous implementation in Figure 11), is equivalent to a homogeneous implementation where the data have been moved from the global memory to the shared memory. The performance is low, as having a single code segment prevents data reuse across threads, leading to a higher ratio of data/thread. Only Θ threads can be run concurrently due to the limited size of the SM memory. This indicates that simply changing the location of the workset without refactoring the computation pattern does not provide any performance boost.

Initially, splitting the code leads to a nearly linear performance increase with respect to the number of resulting code segments $\mathcal{C}$. This shows that the resulting code segments can be well balanced and that the required synchronization has negligible overhead. Eventually, as more code segments are added, the performance reaches a plateau. This corresponds to reaching full pipeline occupancy. From this point onward, there is no benefit from creating additional code segments. Instead, the performance experiences a small degradation due to the granularity of the load balancing and also due to the additional register pressure. For the smaller benchmarks, performance degrades significantly more when too many $\mathcal{C}$ threads are created. In this case, the load balancer handles fewer equations, and their granularity prevents adequate balancing.

We have included the overall results in Table 2. The speed-up achieved by the heterogeneous scheme indicates the suitability of the proposed approach.

Table 3 includes additional details about the number of threads in each thread block of the kernel, the number of registers used, and SM memory occupancy.

	Setup			Runtime(s)		
Model	$\|x\|$	N	$T \cdot r$	Homogeneous	Our scheme	Speed-up
EGF-NGF	32	3×10^6	10^4	280.29	157.14	1.8×
Segmentation clock	16	3×10^6	5×10^4	1563.6	403.5	3.9×
Thrombin	105	3×10^4	2×10^6	8190	4596	1.8×

Table 2: Performance of the proposed approach compared to a homogeneous GPU implementation

Model	Block threads	Registers used	SM used (KB)
EGF-NGF	64×6	32	36.25 (75.5%)
Segmentation clock	128×4	49	46.50 (97.0%)
Thrombin	16×23	63	37.69 (78.5%)

Table 3: Execution configuration, register, and SM usage of the models

We also evaluated the impact of the memory thread specialization by comparing the speed-up achieved by the models

- when heterogeneous threads are used but computation and memory accesses are mixed within the same threads, and
- when compute and memory threads are distinct.

Table 4 underlines the benefit of this separation. The additional speed-up introduced by specialized memory access threads reaches up to 13%. The specialized threads provide better opportunity for data coalescing. In addition, because computation threads never stall, the $\mathcal{C}$ threads can more quickly reuse the small amount of shared memory.

Model	Heterogeneous approach		Specialized memory access threads	
	$\mathcal{C}$	Speed-up	$\mathcal{C}+\mathcal{M}$	Additional speed-up
EGF-NGF	7	1.65×	6 + 1	1.09×
Segmentation clock	5	3.45×	4 + 1	1.13×
Thrombin	24	1.78×	22 + 2	1.01×

Table 4: Benefit of heterogeneous groups and specialized memory threads

Model	Naïve balancing speed-up	Additional speed-up
EGF-NGF	1.62×	1.11×
Segmentation clock	3.80×	1.03×
Thrombin	1.50×	1.20×
Average	2.3×	1.11×

Table 5: Overall speed-up due to thread balancing

We also compare the performance of our thread balancer to naïve load balancing, where the same number of equations is allocated to each compute thread. Unless the equations have the same complexity, some of the threads finish processing earlier, and the GPU is not fully utilized, leading to a significant performance degradation as shown in Table 5. The proposed thread balancer can improve performance up to 1.5× for the set of benchmarks explored.

The results indicate that the heterogeneous scheme alone provides most of the performance improvement. Using heterogeneous threads not only exposes more parallel computation but also enables data reuse in the shared memory; hence, the global memory traffic is significantly reduced, whereas the level of parallelism increases. Furthermore, our method shows one can handle GPUs simulation of ODEs systems that are significantly larger than what conventional multi-processor implementation schemes can accommodate.

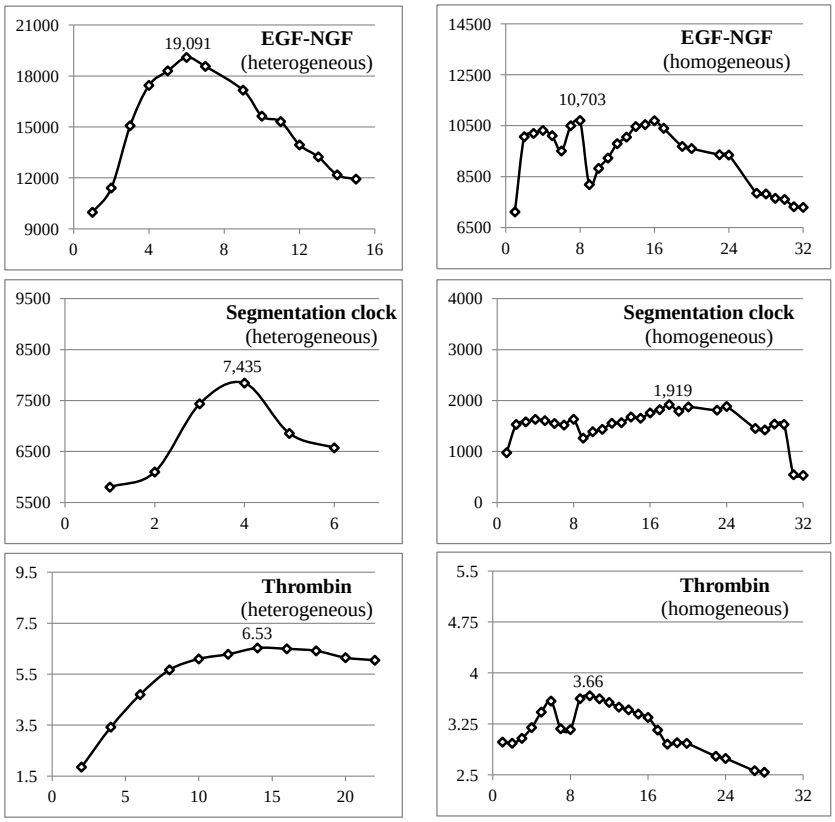


Figure 11: Performance characterization of the proposed heterogeneous scheme (left-side graph for each model) versus the homogeneous approach (right-side graph) on Tesla 2.0 S2050

3.4 SUMMARY

In this chapter, we have presented a GPUs based code-generation scheme for simulations of ODEs dynamics. Specifically, we recalled how an ODEs system can be approximated as a dynamic Bayesian network. Once we construct the DBN, it can be directly used for multiple analysis tasks by repeatedly computing the probability of a random variable assuming a specific value at a particular time point. For large models, approximate inferencing techniques are employed to this effect. However, one must pay a high one time cost of constructing the DBN. There

are other analysis methods to study ODEs dynamics. In the next chapter we introduce one such approach based on statistical model checking for ODEs systems. The automatic GPU code generation scheme which was described in this chapter forms the basis for the computational model of the parallelized implementation of the statistical model checking procedure.

4

STATISTICAL MODEL CHECKING BASED ANALYSIS OF ODEs SYSTEMS

In the previous chapter, we described how an ODEs system can be approximated as a DBN. As pointed out in the introduction, this method is rigid and it is not possible to estimate the error involved in the approximation. Here we provide —as background material for the next chapter— an analysis framework based on a statistical model checking procedure [24], which can provide error guarantees using the machinery of sequential hypothesis testing.

We consider ODEs systems that arise in systems biology. In bio-chemical networks, variability in a population of cells has at least two major causes. First, as discussed in [65], differences in the initial concentrations of proteins are the primary source of variability in response to external stimuli. Second, due to differing internal and external conditions among cells, the values of kinetic rate constants also vary across cells [66, 67]. In our ODEs setting, the variables will represent the concentrations of the bio-chemical species (typically proteins) in the pathway, and hence the initial concentrations of these species will constitute the initial values of the variables. Further, the parameters appearing in the equations will consist of the kinetic rate constants governing the reactions. Thus we can capture cell-to-cell variability in the behaviour of the bio-pathway by studying the ODEs dynamics across a range of values for the initial concentrations and kinetic rate constant values. We do this in a probabilistic setting by assuming initial probability distributions (usually uniform) over an interval of values for the initial con-

centrations and rate constants. We then show that the resulting space of trajectories can be used to construct a natural probability measure space if the vector field defined by the ODEs system is continuously differentiable. In our setting this requirement is easily met.

4.1 OVERVIEW

To analyze the ODEs system, we first formalize properties using our specification logic and decide a corresponding confidence level (probability) with which we wish to assess them. Consequently, an SMC procedure —which poses the problem as a hypothesis test— is used to decide approximately, but with statistical guarantees, whether the properties are satisfied with the desired probability. SMC continues to sample and verify trajectories from the ODEs system until a decision can be made. It is well-established that SMC is efficient since its complexity does not depend on the size of the system. Moreover, posing the problem as a sequential hypothesis test reduces the overall number of samples needed to make a decision [43]. These components form a principled method for analyzing the dynamics of a bio-pathway in the presence of dynamic variability across a population of cells.

To demonstrate the applicability of the approach, we describe a SMC based parameter estimation method. The unknown model parameters usually consist of initial concentrations and kinetic rate constants. Here, for convenience, we shall assume that all the initial concentrations are known but that their nominal values can vary over a cell population. The parameter estimation procedure searches through the value space of the unknown parameters to determine the "best" combination of values that can explain the given data and predict new behaviours [68]. The key step in this procedure is to determine the fit-to-data of the cur-

rent set of parameter values. We use our specification logic to encode both experimental time series data and known qualitative trends concerning the dynamics of the pathway. We then use our statistical model checking procedure (SMC) to determine the goodness of the given set of parameter values, while taking into account that these values can fluctuate across the population of cells that the data is based on. Subsequently, we use a global optimization strategy known as SRES [69] to choose a new set of candidate parameter values according to the SMC based score assigned to the current set. In Chapter 5, we will see how this procedure can be parallelized using GPUs to numerically generate trajectories in parallel and use our online model checking method to determine if the current trajectory satisfies the given specification.

4.2 ODEs AND TRAJECTORIES

We first recall the notations developed for describing the dynamics of a bio-chemical network as a system of ODEs. Assume that there are n molecular species $\{x_1, x_2, \ldots, x_n\}$ involved in the network. For each molecular species x_i taking part in the pathway there will be an equation of the form $\frac{dx_i}{dt} = f_i(\mathbf{x}, \Theta_i)$. Here f_i describes the kinetics of the reactions that produce and consume x_i, $\mathbf{x}$ are the molecular species taking part in these reactions while the vector Θ_i gives the rate constants governing these reactions.

Each x_i is real-valued function of t with $t \in \mathbb{R}_+$, where $\mathbb{R}_+$ denotes the set of non-negative reals. We shall realistically assume that $x_i(t)$ takes values in the interval $[L_i, U_i]$ where L_i and U_i are non-negative rationals with $L_i < U_i$. Hence the state space of the system will be $\mathbf{V} = [L_1, U_1] \times [L_2, U_2] \ldots \times [L_n, U_n] \subseteq \mathbb{R}_+^n$. Let $\Theta = \bigcup_i \Theta_i = \{\theta_1, \theta_2, \ldots, \theta_m\}$ be the set of all rate constants. We again assume that the range of values

for each θ_j is $[L_j, U_j]$ for $1 \leq j \leq m$. We shall present the SMC procedure while assuming that all the rate constants are known. In the next chapter, we shall explain how we handle the unknown rate constants in more detail and our approach to solving these problems. Here, in order to develop the basic material and notions, we shall assume all the rate constants are known rational values.

An implicit assumption in what follows is that the value of a rate constant, when fixed initially, does not change during the time evolution of the dynamics, although this value can be different for different cells. To capture the cell-to-cell variability and uncertainties regarding the initial states we define for each variable x_i an interval $[L_i^{init}, U_i^{init}]$ with $L_i \leq L_i^{init} < U_i^{init} \leq U_i$. The actual value of the initial concentration of x_i is assumed to fall in this interval. Similarly, we shall assume that the nominal value of the rate constant θ_j falls in the interval $[L_{init}^j, U_{init}^j]$ with $L^j \leq L_{init}^j < U_{init}^j \leq U^j$. We set $INIT = (\Pi_i[L_i^{init}, U_i^{init}]) \times \Pi_j([L_{init}^j, U_{init}^j])$. . Thus $INIT$ captures the cell-to-cell variability in the initial concentration and the rate constant values. In what follows we let $\mathbf{v}$ to range over $\Pi_i[L_i^{init}, U_i^{init}]$ and $\mathbf{w}$ to range over $\Pi_j([L_{init}^j, U_{init}^j])$.

In what follows, it will be convenient to represent our system of ODEs in vector form as $\frac{d\mathbf{x}}{dt} = F(\mathbf{x}, \Theta)$ with $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $F_i(\mathbf{x}, \Theta) := f_i$. Recall that a function $f_i : \mathbf{V} \to \mathbf{V}$ is a C^1 function if f_i', the derivative of f, exists at all $\mathbf{v} \in \mathbf{V}$ and is a continuous function. In the setting of bio-chemical networks, the expressions in f_i will model kinetic laws such as mass law and Michaelis-Menten's [31]. Moreover, the concentration levels of the various species will be bounded and the behaviour of the system will be of interest only up to a finite time horizon. Hence we assume that f_i is Lipschitz-continuous for each i. As a result, for each $(\mathbf{v},\mathbf{w}) \in INIT$, the system of ODEs will have a unique solution $\mathbf{X}_{\mathbf{v},\mathbf{w}}(t)$ [20]. Further, it will satisfy: $\mathbf{X}_{\mathbf{v},\mathbf{w}}(0) = \mathbf{v}$ and $\mathbf{X}'_{\mathbf{v},\mathbf{w}}(t) = F(\mathbf{X}_{\mathbf{v},\mathbf{w}}(t))$.

We are also guaranteed that $\mathbf{X}_{\mathbf{v},\mathbf{w}}(t)$ is a C^0-function (i.e. continuous function) [20] and hence measurable. This fact will be crucial when we later turn to probabilistic verification.

It will be convenient to define the flow $\Phi_{\mathbf{w}} : \mathbb{R}_+ \times \mathbf{V} \to \mathbf{V}$ for arbitrary initial vectors $\mathbf{v}$ as $X_{\mathbf{v},\mathbf{w}}(t)$. Intuitively, $\Phi_{\mathbf{w}}(t, \mathbf{v})$ is the state reached under the ODEs dynamics if the system starts at $\mathbf{v}$ at time 0. The flow will be the C^0-function given by: $\Phi_{\mathbf{w}}(t, \mathbf{v}) = \mathbf{X}_{\mathbf{v},\mathbf{w}}(t)$. Thus $\Phi_{\mathbf{w}}(0, \mathbf{v}) = \mathbf{X}_{\mathbf{v},\mathbf{w}}(0) = \mathbf{v}$ and $\partial(\Phi_{\mathbf{w}}(t, \mathbf{v}))/\partial t = F(\Phi_{\mathbf{w}}(t, \mathbf{v}))$ for all t [20]. We will, in fact, work with $\Phi_{\mathbf{w},t} : \mathbf{V} \to \mathbf{V}$ instead of $\Phi_{\mathbf{w}}$, where $\Phi_{\mathbf{w},t}(\mathbf{v}) = \Phi_{\mathbf{w}}(t, \mathbf{v})$ for every t and every $\mathbf{v} \in \mathbf{V}$. Again, $\Phi_{\mathbf{w},t}$ is guaranteed to be a C^0-function (in fact 1-to-1) and Φ^{-1} will also be a C^0-function.).

In our application, the dynamics will be of interest only up to a maximal time point T. Fixing such a T, a *trajectory* starting from $\mathbf{v} \in \mathbf{V}$ at time 0 and with $\mathbf{w}$ as parameter values is denoted $\sigma_{\mathbf{v},\mathbf{w}}$ to be the (continuous) function $\sigma_{\mathbf{v},\mathbf{w}} : [0, T] \to \mathbf{V}$ satisfying: $\sigma_{\mathbf{v},\mathbf{w}}(t) = X_{\mathbf{v},\mathbf{w}}(t)$. The behaviour of our dynamical system is the set of trajectories given by $BEH = \{\sigma_{\mathbf{v},\mathbf{w}} \mid (\mathbf{v},\mathbf{w}) \in INIT\}$. Our first goal is to probabilistically verify the dynamical properties of BEH.

4.3 STATISTICAL MODEL CHECKING OF ODEs DYNAMICS

In order to formally express dynamical properties of BEH, we will use formulas in bounded linear-time temporal logic (BLTL) since our trajectories will be of finite duration.

4.3.1 *Bounded linear-time temporal logic*

An atomic proposition in our logic will be of the form (i, ℓ, u) with $L_i \leq \ell < u \leq U_i$. Such a proposition will be interpreted as "the current

concentration level of x_i is in the interval $[\ell, u]$", and we fix a finite set of such atomic propositions.

We recall again —to fit the present context— the syntax and then the semantics of BLTL formulas. The BLTL formulas are defined as:

- Every atomic proposition as well as the constants *true*, $false$ are BLTL formulas.
- If ψ and $\psi^{'}$ are BLTL formulas then $\sim \psi$ and $\psi \vee \psi^{'}$ are BLTL formulas.
- If ψ is a BLTL formula then $\mathbf{O}(\psi)$ is a BLTL formula.
- If ψ and $\psi^{'}$ are BLTL formulas and $t \leq T$ is a positive integer then $\psi \mathbf{U}^{\leq t} \psi'$ and $\psi \mathbf{U}^{t} \psi'$ are BLTL formulas.

We have mildly strengthened BLTL to be able to express that a certain property will hold exactly at t time units from now. This will enable us to encode experimental data in the specification. The derived propositional operators such as $\wedge$, $\supset$, $\equiv$ and the temporal operators $\mathbf{G}^{\leq t}$, $\mathbf{F}^{\leq t}$, $\mathbf{F}^{t}$ are defined in the usual way.

We will interpret the formulas of our logic at the finite set of time points $\mathcal{T} = \{0, 1, \ldots, T\}$. Such a discretization is reasonable since experimental data will be available only at a finite number of discrete time points. Further, qualitative properties of interest are expressible in discrete time. We assume that $\mathcal{T}$ has been chosen appropriately and it includes all the relevant time points with respect to the specified properties.

Further the corresponding semantics of the logic is defined in terms of the relation $\sigma, t \models \phi$ where σ is a trajectory in BEH and $t \in \mathcal{T}$, a finite set of time points {0, 1, . . . , T} and ϕ, the property of interest:

- $\sigma, t \models (i, \ell, u)$ iff $\ell \leq \sigma(t)(i) \leq u$ where $\sigma(t)(i)$ is the i^{th} component of the n-dimensional vector $\sigma(t)$.

- $\neg$ and $\vee$ are interpreted in the usual way.
- $\sigma, t \models \psi \mathbf{U}^{\leq k} \psi'$ iff there exists k' such that $k' \leq k$, $t + k' \leq T$ and $\sigma, t + k' \models \psi'$. Further, $\sigma, t + k'' \models \psi$ for every $0 \leq k'' < k'$.
- $\sigma, t \models \psi \mathbf{U}^{k} \psi'$ iff $t + k \leq T$ and $\sigma, t + k \models \psi'$. Further, $\sigma, t + k' \models \psi$ for every $0 \leq k' < k$.

Now one can define $Models(\psi) = \{\sigma \mid \sigma, 0 \models \psi,\ \sigma \in BEH\}$.

Next, we wish to make statements of the form $P_{\geq r}(\psi)$, where the intended meaning is that the probability that a trajectory in BEH belongs to $models(\psi)$ is at least r. To assign meaning to such statements, we need to define a probability measure over sets of trajectories. Note, however, that the trajectory $\sigma \in BEH$ is completely determined by $\sigma(0)$, the (vector) value it assumes at t = o. Hence we will identify BEH with $INIT$, the set of initial states. To make this explicit, we define the set $Models(\psi) \subseteq INIT$ as:

$(\mathbf{v,w}) \in Models(\psi)$ iff $\sigma_{\mathbf{v,w}} \in models(\psi)$. We define the formulas of PBLTL as $P_{\geq r}(\psi)$ and $P_{\leq r'}(\psi)$ provided $r \in [0, 1), r' \in (0, 1]$ and ψ is a BLTL formula. We shall say that $\mathcal{S}$, the system of ODEs, meets the specification$P_{\geq r}(\psi)$ —and this is denoted $\mathcal{S} \models P_{\geq r}(\psi)$— iff $P(Models(\psi)) \geq r$, while $\mathcal{S} \models P_{\leq r'}(\psi)$ iff $P(Models(\psi)) \leq r'$.Here, and in what follows, P is the standard probability measure assigned to members of the σ-algebra generated by the open intervals contained in $INIT$. It is easy to show that $Models(\psi)$ is a member of this σ-algebra for every ψ. The only case that requires an argument is the one for atomic propositions, and here the measurability of the solution functions $X_{v,w}(t)$ is crucial.

4.3.2 *Statistical model checking of PBLTL formulas*

We now introduce a statistical framework for deciding approximately, but with statistical guarantees, whether the model satisfies a property of the form $P_{\geq r}(\psi)$. Instead of directly approximating the probability of ψ being satisfied [44], we formulate the model checking problem whether the ODEs system $\mathcal{S} \models P_{\geq r}(\psi)$, as a sequential hypothesis test. According to [70], the test is posed between the null hypothesis $\mathrm{H}_0 : p \geq r + \delta$ and the alternative hypothesis $\mathrm{H}_1 : p \leq r - \delta$, where $p = P(Models(\psi))$. Here, δ is supplied by the user and signifies the indifference region. The *strength* of the test is decided by parameters α and β which bound the Type-I (false positive) and Type-II (false negative) errors respectively. Thus the verification is carried out approximately but with guaranteed confidence levels and error bounds. The test proceeds by generating a sequence of sample trajectories $\sigma_1, \sigma_2, \ldots$ by randomly sampling an initial state from the initial distribution. One assumes a corresponding sequence of Bernoulli random variables $y_1, y_2 \ldots$, where each y_k is assigned the value 1 if $\sigma_k, 0 \models \psi$; otherwise y_k is assigned the value 0. A sequential test is constructed that helps to decide if the number of samples taken are sufficient or whether more samples need to be taken to guarantee the chosen test *strength*. For each $m \geq 1$, after drawing m samples, we compute a quantity q_m as:

$$q_m = \frac{[r-\delta]^{(\sum_{i=1}^{m} y_i)}[1-[r-\delta]]^{(m-\sum_{i=1}^{m} y_i)}}{[r+\delta]^{(\sum_{i=1}^{m} y_i)}[1-[r+\delta]]^{(m-\sum_{i=1}^{m} y_i)}} \tag{3}$$

When sufficient samples are drawn, the test terminates. Otherwise, the test proceeds to draw more samples until the statistical guarantee defined by the error bounds and the indifference region are met. The ratio q_m serves as a stopping criterion for the sampling process. Hypothesis

H_1 is accepted if $q_m \geq \widehat{A}$, and Hypothesis H_0 is accepted if $q_m \leq \widehat{B}$. If neither is the case then another sample is drawn. The constants $\widehat{A}$ and $\widehat{B}$ are chosen such that it results in a test of strength (α, β). In practice, a good approximation is $\widehat{A} = \frac{1-\beta}{\alpha}$ and $\widehat{B} = \frac{\beta}{1-\alpha}$.

4.4 PARAMETER ESTIMATION

Here we present our parameter estimation method. In doing so, we assume the terminology and notations developed in the previous sections. As a first step, we describe how experimental data can be encoded as BLTL formulas.

Assume, without loss of generality, that $O \subseteq \{x_1, x_2, \ldots, x_k\}$ is the set of variables for which experimental data is available and which has been alloted as training data to be used for parameter estimation. Assume $\mathcal{T}_i = \{\tau_1^i, \tau_2^i, \ldots, \tau_{T_i}^i\}$ are the time points at which the concentration level of x_i has been measured and reported as $[\ell_t^i, u_t^i]$ for each $t \in \mathcal{T}_i$. The interval $[\ell_t^i, u_t^i]$ is chosen to reflect the noisiness, the limited precision and the cell-population-based nature of the experimental data. For each $t \in \mathcal{T}_i$ we define the formula $\psi_i^t = \mathcal{F}^t(i, \ell_t^i, u_t^i)$. Then $\psi_{exp}^i = \bigwedge_{t \in \mathcal{T}_i} \psi_i^t$. We then set $\psi_{exp} = \bigwedge_{i \in O} \psi_{exp}^i$. In case the species x_i has been measured under multiple experimental conditions, then the above encoding scheme is extended in the obvious way.

Often qualitative dynamic trends will be available —typically from the literature— for some of the molecular species in the pathway. For instance, we may know that a species shows transient activation in which its level rises in the early time points and later falls back to initial levels. Similarly, a species may be known to show oscillatory behaviour with certain characteristics. Such information can be described as BLTL

formulas that we term to be *trend* formulas. We let ψ_{qlty} to be the conjunction of all the trend formulas.

Finally we fix the PBLTL formula $P_{\geq r}(\psi_{exp} \wedge \psi_{qlty})$, where r will capture the confidence level with which we wish to assess the goodness of the fit of the current set of parameters to experimental data and qualitative trends. We also fix an indifference region δ and the strength of the test (α, β). The constants r, δ, α and β are to be fixed by the user. In our application it will be useful to exploit the fact that both ψ_{exp} and ψ_{qlty} are conjunctions and hence can be evaluated separately. As shown in [70], one can choose the strength of each of these tests to be $(\frac{\alpha}{J}, \beta)$, where J is the total number of conjuncts in the specification. This will ensure that the overall strength of the test is (α, β). Further, the results for the individual statistical tests can be used to compute the objective function associated with the global search strategy, as detailed below.

4.4.1 *Parameter estimation based on PBLTL specification*

We assume $\Theta_u = \{\theta_1, \theta_2, \ldots, \theta_K\}$ as the set of unknown parameters. For convenience we will assume that the other parameter values are known and that their nominal values do not fluctuate across the cell population. We will also assume nominal values for the initial concentrations and the range of their fluctuations of the form $[L_i^{init}, U_i^{init}]$ for each variable x_i. Again, for convenience, we fix a constant δ'' so that if the current estimate of the values of the unknown parameters is $\mathbf{w} \in \prod_{1 \leq j \leq K}[L^j, U^j]$ then this value will fluctuate in the range $[\mathbf{w}(j) - \delta'', \mathbf{w}(j) + \delta'']$. Setting $L^j_{init,\mathbf{w}} = \mathbf{w}(j) - \delta''$ and $U^j_{init,\mathbf{w}} = \mathbf{w}(j) + \delta''$ we define $INIT_{\mathbf{w}} = (\prod_i [L_i^{init}, U_i^{init}]) \times (\prod_j [L^j_{init,\mathbf{w}} U^j_{init,\mathbf{w}}])$. The set of trajectories $BEH_{\mathbf{w}}$ is defined accordingly.

To estimate the quality of $\mathbf{w}$, we run our parallel SMC procedure — using $INIT_{\mathbf{w}}$— to verify $P_{\geq r}(\psi_{exp} \wedge \psi_{qlty})$. Depending on the outcome of the test for the various conjuncts in the specification, we assign a score to $\mathbf{w}$ using an objective function detailed below. We then iterate this scheme for various values of $\mathbf{w}$ generated using a suitable search strategy. The objective function consists of two components, evaluating the contribution from the qualitative properties and the experimental data respectively. It evaluates how many statistical tests carried out with $\mathbf{w}$ resulted in acceptance of the null hypothesis (desired outcome). For the second component, the tests are evaluated species-wise. The corresponding objective value is then composed as a summation of normalized contribution from each species.

The objective function is formed as follows. Let J^i_{exp} $(= T_i)$ be the number of conjuncts in ψ^i_{exp}, and J_{qlty} the number of conjuncts in ψ_{qlty}. Let $J^{i,+}_{exp}(\mathbf{w})$ be the number of formulas of the form ψ^t_i (a conjunct in ψ^i_{exp}) such that the statistical test for $P_{\geq r}(\psi^t_i)$ accepts the null hypothesis (that is, $P_{\geq r}(\psi^t_i)$ holds) with the strength $(\frac{\alpha}{J}, \beta)$, where $J = \sum_{i \in O} J^i_{exp} + J_{qlty}$. Similarly, let $J^+_{qlty}(\mathbf{w})$ be the number of conjuncts in ψ_{qlty} of the form $\psi_{\ell,qlty}$ that pass the statistical test $P_{\geq r}(\psi_{\ell,qlty})$ with the strength $(\frac{\alpha}{J}, \beta)$. Then $\mathcal{G}(\mathbf{w})$ is computed via:

$$\mathcal{G}(\mathbf{w}) = J^+_{qlty}(\mathbf{w}) + \sum_{i \in O} \frac{J^{i,+}_{exp}}{J^i_{exp}} \tag{4}$$

Thus the goodness to fit of $\mathbf{w}$ is measured by how well it agrees with the qualitative properties as well as the number of experimental data points with which there is acceptable agreement. To avoid over-training the model, we do not insist that every qualitative property and every data point must fit well with the dynamics predicted by $\mathbf{w}$.

The search strategy to evolve candidate parameters will use the values $\mathcal{G}(\mathbf{w})$ to traverse the parameter value space. Global search methods such as Genetic Algorithms (GA) [71], and Stochastic Ranking Evolutionary Strategy (SRES) [69] are computationally more intensive than local methods, but are much better at avoiding local minima. In practice, one usually maintains a *population* of parameter value vectors in each round, and a round is usually called a *generation*. We use the SRES strategy in our work since it is known to perform well in the context of pathway models [68]. The particular choice of search algorithm, however, is orthogonal to our proposed method.

4.5 SUMMARY

Here we described an SMC based approach for studying ODEs systems as background material based on the results presented in [24]. We have used the temporal logic BLTL to encode both quantitative experimental data and qualitative properties of pathway dynamics. To cater for variability among cells, we assume a uniform distribution over a set of initial states and kinetic rate constants —and impose a reasonable continuity restriction— and show how the probability of the property being met by the behaviour of the model can be assessed using an SMC procedure. In the next chapter, we develop a GPU-based implementation of our SMC algorithm to exploit the inherent massive parallelism in generating trajectories through numerical integration. By combining this method with a global search strategy, we arrive at an efficient parameter estimation procedure.

5

A GPU BASED IMPLEMENTATION OF THE SMC PROCEDURE FOR ODEs SYSTEMS

In the previous chapter, we showed how a system of ODEs together with a (initial) set of values for the initial concentrations and the rate constant values can be formulated as a model of a bio-chemical network that takes into account cell-cell variability in a population. These ODEs systems will be high dimensional with no closed-form solutions. To get around this, a probabilistic approximation technique accompanied by a statistical model checking (SMC) procedure —as sketched in the previous chapter and whose underlying theory was developed in detail in [24]— is used to carry out parameter estimation as follows. A conjunction of BLTL formulas describe the available experimental time-course data as well as known qualitative properties. One then deploys the statistical model checking procedure to evaluate the goodness of the current estimates for unknown parameter values. With the help of an evolutionary search strategy one then searches through the parameter space to obtain a good set of parameter values. The estimated values are then validated using test data that was not made available to the estimation procedure.

For high dimensional ODEs systems with many unknown parameters, one will have to call upon the SMC procedure many times and for each such call one will have to generate sufficiently many trajectories of the ODEs system to ensure the termination of the SMC procedure. Consequently, the computational cost induced by the repeated executions of the SMC procedure can be quite high. In this chapter, we develop a

GPU based implementation of the above mentioned parameter estimation procedure.

5.1 OVERVIEW

Obviously, one can numerically generate trajectories in parallel on a GPU. Thus it is tempting to take for granted an easy parallel implementation and a corresponding increase in performance. This is, however, not the case. The memory hierarchy of a GPU and its single-instruction multiple-thread (SIMT) organization of its arithmetic units constitute severe constraints. A naïve implementation will often perform no better than (and in some cases worse than!) a sequential implementation. GPUs are, however, an attractive candidate since they are available off-the-shelf and can offer performance that is comparable to the more-expensive and less-available multi-core platforms. Furthermore, it is possible to form large pools of GPUs in a scalable and cost effective way using cloud services. Therefore, the effort required to overcome the architectural constraints of GPUs may well be worth it and this is the hypothesis we pursue here.

In simplified terms, the iterative parameter estimation procedure based on SMC consists of:

(i) Encode the experimental data and known qualitative trends as a BLTL formula φ (as detailed in Section 5.2).

(ii) Fix the required confidence level and the false positives and negatives rates w.r.t. which one wishes to verify φ.

(iii) Guess a current value for each unknown parameter.

(iv) Evaluate the goodness of these estimated parameters by repeatedly generating trajectories till the statistical test associated with the SMC procedure terminates.

(v) If the outcome is yes then the current estimate is a good one. If not, guess a new set of values using the evolutionary search strategy and iterate.

Thus it is step (iv) which is ripe for parallelization. However just generating a numerical trajectory is not enough. One must evaluate if it satisfies φ which is of course easy to do. However only a small amount of memory will be available in the vicinity of a GPU core. Hence the generated trajectories need to be sent up through a number of levels in the memory hierarchy, each of which is significantly slower than the previous one. This will all but eliminate the performance gains obtained by generating the trajectories in parallel. Hence one must verify whether a generated trajectory satisfies φ on the fly without having to store the whole trajectory. Again this is not difficult to do though one must minimize the amount of intermediate data (typically Boolean combinations of the subformulas of φ that still need to be satisfied) to be kept track of. However the obvious online procedures will involve branching that is based on the current requirements and this will clash with the hardware parallelism available in GPUs. At the level of a single core, groups of parallel threads called *warps* are scheduled to run the compiled code, which at each step, execute the same machine instruction in a lock-step fashion. This is the heart of GPU's execution model. If two threads in a warp take different branches, the warp will have to be executed twice, once for each branch. This so called *branch divergence* causes severe performance degradation [72]. To avoid this, we construct a *deterministic* automaton-based online model checking technique. It turns out that it is better to store the automaton (as a look-up table) in the intermediate

storage shared by the cores and hence we also implement a standard latency hiding technique to mitigate the data transfer delays between this shared store and the global store (using which the rest of the analysis is carried out) during model checking.

5.1.1 *Related work*

Efficient methods for model checking probabilistic systems have been studied [43, 73–76]. The statistical model checking (SMC) approach initiated by Younes and Simmons [70] based on the sequential probability ratio test proposed by Wald [77] has turned out to be a fruitful one and is adopted here. SMC usually involves checking whether an individual trace satisfies a given temporal specification. When the specification is a BLTL formula, this is known as *BLTL path checking*. Kuhtz and Finkbeiner show that the path checking problem can be parallelized by unrolling the BLTL formulas into Boolean circuits [78]. Barre *et al.* adopt the MapReduce framework [79] to verify a single large trace using distributed computing [80]. However, it is not clear how these methods can be implemented on a GPU-based platform.

On the other hand, Barnat *et al.* take an automata-theoretic approach to parallel model checking of a restricted class of multi-affine ODEs systems [81, 82]. The ODEs model dynamics is first approximated as a rectangular abstraction automaton and a given LTL property is translated into a Büchi automaton that represents its negation. A parallel model checker then looks for an accepting cycle in the product automaton by symbolically exploring the state space. But this approach tends to over-approximate the model dynamics. Oshima *et al.* present a FPGA-based framework for the checking of BLTL specifications with applications on partial differential equations [83]. Their method also involves a Büchi au-

tomaton construction but requires a large set of trajectories to be stored in the hardware before a property can be verified. In contrast our online method is based on GPUs, which we believe are more accessible and scalable. Further our focus is on ODEs systems.

In recent years, statistical model checking has become a building block to solve complex problems. David *et al.* apply SMC using analysis of variance (ANOVA) to find the optimal set of parameters of a network of stochastic hybrid automata [84]. Jha *et al.* show how the parameter synthesis problem for stochastic systems can be approached using statistical model checking [85]. Here, we focus on efficient parallelization techniques for traditional analysis tasks based on SMC, especially parameter estimation [86].

5.2 ONLINE STATISTICAL MODEL CHECKING PROCEDURE

Recall from the previous chapter, we use formulas in bounded linear-time temporal logic (BLTL). The problem of BLTL path checking involves determining whether a BLTL formula is satisfied by a trajectory. According to the BLTL semantics, it is easy to see that the truth value of a BLTL formula can be decided by trajectories with finite length. Online BLTL path checking requires only the current valuation of the atomic propositions as input. At each step, it evaluates the BLTL formula under the current valuation and generates a new formula that represents the "obligation" in the following step. The procedure terminates when the formula under consideration becomes either true or false, indicating a satisfaction or falsification of the original formula.

Such an algorithm can be easily implemented on CPUs. On the other hand, to achieve good performance on GPUs one must address the problem of branch divergence, which occurs when two GPU threads choose

different code segments under the evaluation of a condition as illustrated in the following example.

Example 1 *Branch Divergence: Consider BLTL formula* $\phi = F^{\leq 8}G^{\leq 5}p$*, where* p *is an atomic proposition. Expanding* ϕ*, we get* $\phi = \left(p \wedge XG^{\leq 4}p\right) \vee XF^{\leq 7}G^{\leq 5}p$*. Notice that if the current valuation is* $\sigma_1 = \{p \mapsto false\}$*,* ϕ *is reduced to* $\phi_1 = F^{\leq 7}G^{\leq 5}p$*; if it is* $\sigma_2 = \{p \mapsto true\}$*,* ϕ *is reduced to* $\phi_2 = G^{\leq 4}p \vee F^{\leq 7}G^{\leq 5}p$*.*

Now we initiate two GPU threads to check whether ϕ *is satisfied for two different trajectories. Naively, we implement each thread as if* σ_1 *then check* ϕ_1 *else check* ϕ_2*. Branch divergence happens when the two trajectories take different valuations. Since GPU stream processors require that each GPU thread executes identical instructions, the two threads will process both* ϕ_1 *and* ϕ_2 *and simply discard the unrelated part, resulting in a 50% loss of performance.* □

5.2.1 *Automaton-based BLTL path checking*

To better utilize the parallelism of GPUs, we introduce an automaton-based BLTL path checking algorithm. Given a BLTL formula ψ, it is well-known that there exists a positive integer K that depends only on ψ such that for any trajectory τ whose length is greater than K, one needs to examine only a prefix of length K to determine whether τ is a model of ψ [87]. The online procedure we shall construct examines τ as it is being generated (through numerical simulation) in a lock-step fashion. Instead of generating a trajectory of length K at once, it incrementally simulates the ODEs model and checks whether the current trajectory satisfies the formula ψ.

It is convenient to focus on the sequence of truth values of the atomic propositions induced by a trajectory. Let us call such a sequence *AP-sequence*. Given a trajectory $\tau = \mathbf{v}_0\mathbf{v}_1 \ldots \mathbf{v}_k$, its induced AP-sequence is denoted as τ_{ap}, which is the sequence $P_0P_1 \ldots P_k$ where for $0 \leq i \leq k$:

$(x_j \bowtie v) \in P_i$ iff $\mathbf{v}_i(j) \bowtie v$, $\bowtie \in \{\leq, \geq\}$.

We now wish to construct a deterministic automaton for ψ that accepts (rejects) an AP-sequence iff it is (not) a model of ψ.

As the first step, we replace the time constants mentioned in ψ by symbolic variables and manipulate these variables separately. To this end, we define the formula $sym(\psi)$ inductively as follows.

- $sym(\psi) = \psi$ if ψ is an atomic proposition;
- $sym(\neg\psi) = \neg sym(\psi)$ and $sym(\psi_1 \vee \psi_2) = sym(\psi_1) \vee sym(\psi_2)$;
- $sym(X\psi) = Xsym(\psi)$;
- $sym(\psi_1 U^{\leq t} \psi_2) = sym(\psi_1) U^{\leq x_\alpha} sym(\psi_2)$ where $\alpha = \psi_1 U^{\leq t} \psi_2$.

Thus the subscript assigned to the symbolic variable is the sub-formula in which the time constant appears. Often for convenience we will index these variables by integers rather than concrete formulas. Thus $sym(F^{\leq 8} p \vee G^{\leq 3} q)$ will be typically represented as $F^{\leq x_1} p \vee G^{\leq x_2} q$. We refer to $sym(\psi)$ as a *symbolic* BLTL formula.

For a BLTL formula ψ, we now define the automaton $\mathcal{A}_\psi = \langle S_\psi, 2^{AP_\psi}, \rightarrow, s_{in}, \mathcal{F} \rangle$, where S_ψ is the set of states, AP_ψ is the set of atomic propositions that appear in ψ, $\rightarrow \subseteq S_\psi \times 2^{AP_\psi} \times S_\psi$ is the transition relation (to be defined below), $s_{in} \in S_\psi$ is the initial state and $\mathcal{F} \subseteq S_\psi$ are the final states.

Let $\phi_{in} = sym(\psi)$ and CL be the least set of formulas that contains the sub-formulas of $sym(\psi)$ and satisfies:

If $\psi_1 U^{\leq x} \psi_2$ is in CL then $X\psi_1 U^{\leq x} \psi_2$ is also in CL.

We let BC denote the Boolean combinations of formulas in CL. A state of the automaton is a triple of the form (ϕ, Y, V), where $\phi \in BC$, Y is the set of variables that appear in ϕ, and V is a valuation that assigns a *positive* integer to every variable in Y. We define $s_{in} = (\phi_{in}, Y_{in}, V_{in})$,

where Y_{in} is the set of the symbolic variables that appear in ϕ_{in}, and V_{in} assigns to each variable in Y_{in} the corresponding value in ψ. More precisely, if x_α is in Y_{in} and $\alpha = \psi_1 U^{\leq t} \psi_2$ then $V_{in}(x_\alpha) = t$. $\mathcal{F} = \{(true, \emptyset, \emptyset), (false, \emptyset, \emptyset)\}$.

Next we define the the transition relation $\rightarrow$ of $\mathcal{A}$. Let (ϕ, Y, V) and (ϕ', Y', V') be states and $P \subseteq AP_\psi$ be a set of atomic propositions. Then $(\phi, Y, V) \xrightarrow{P} (\phi', Y', V')$ is a transition iff the following conditions are satisfied.

- Suppose $\phi = p$ is an atomic proposition. If $p \in P$, then $\phi' = true$; otherwise, $\phi' = false$. In either case $Y' = V' = \emptyset$.
- Suppose $\phi = \neg\varphi$, and there exists a transition $(\varphi, Y, V) \xrightarrow{P} (\varphi', Y'', V'')$. Then $\phi' = \neg\varphi'$, $Y' = Y''$ and $V' = V''$.
- Suppose $\phi = \phi_1 \vee \phi_2$, and there exist transitions $(\phi_1, Y_1, V_1) \xrightarrow{P} (\phi_1', Y_1', V_1')$ and $(\phi_2, Y_2, V_2) \xrightarrow{P} (\phi_2', Y_2', V_2')$. Then $\phi' = \phi_1' \vee \phi_2'$, $Y' = Y_1' \cup Y_2'$, and $V'(x_i) = V_i'(x_i)$ for $x_i \in X_i$, $i \in \{1, 2\}$.
- Suppose $\phi = X\varphi$. Then $\phi' = \varphi$ and $Y' = Y$ and $V' = V$.
- Suppose $\phi = \phi_1 U^{\leq x_\alpha} \phi_2$, and there exist transitions $(\phi_1, Y_1, V_1) \xrightarrow{P} (\phi_1', Y_1', V_1')$ and $(\phi_2, X_2, V_2) \xrightarrow{P} (\phi_2', Y_2', V_2')$. Then $\phi' = \phi_2' \vee (\phi_1' \wedge X\varphi)$ where $\varphi = \phi_2$ if $V(x_\alpha) = 1$. Furthermore $Y' = Y_1' \cup Y_2'$ and V' restricted to Y_1' is V_1' and V' restricted to Y_2' is V_2'. If $V(x_\alpha) > 1$ then $\varphi = \phi_1 U^{\leq x_\alpha} \phi_2$. Furthermore $Y' = Y_1' \cup Y_2' \cup \{x_\alpha\}$ while V' restricted to Y_1' is V_1' and V' restricted to Y_2' is V_2'. In addition $V'(x_\alpha) = V(x_\alpha) - 1$.

The set of states S_ψ is given inductively:
$s_{in} \in S_\psi$. Suppose $s \in S_\psi$ and $s \xrightarrow{P} s'$. Then $s' \in S_\psi$.

It is easy to show that this automaton has the required properties. Moreover its number of states is bounded by $\ell + \Sigma_{x \in X_{in}} V_{in}(x)$ where ℓ is the number of appearances of the X operator in ψ.

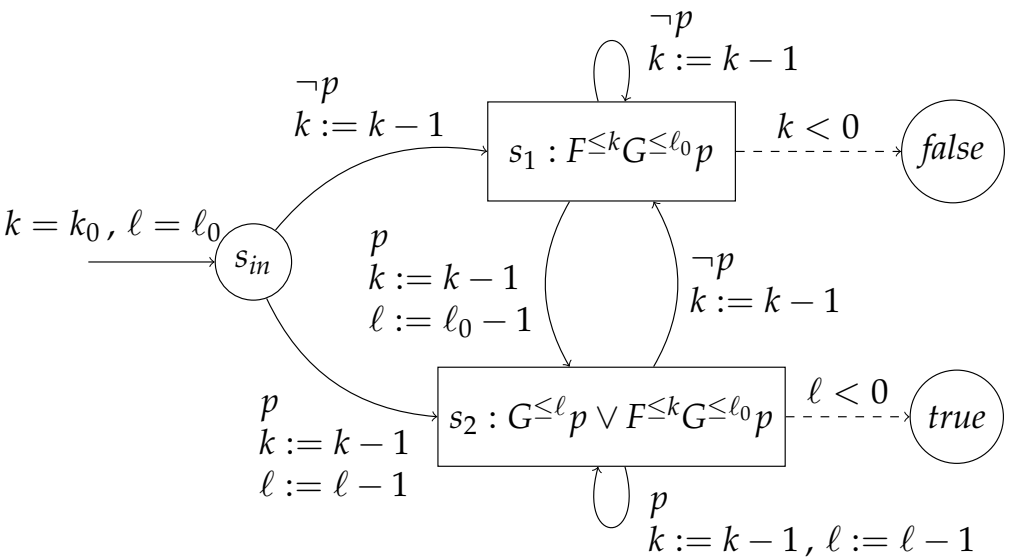


Figure 12: Automaton for the nested BLTL formula $F^{\leq k_0} G^{\leq \ell_0} p$

Example 2 *Consider the BLTL formula $\psi = F^{\leq k_0} G^{\leq \ell_0} p$, where k_0 and ℓ_0 are constants. Figure 12 shows a fragment of the automaton A_ψ. To avoid clutter we have not explicitly shown the symbolic variables and their valuations. The dashed arcs indicate that the input states will transit to the corresponding final states given proper valuations of atomic propositions.* □

5.3 MAPPING TO THE GPU PLATFORM

In this section, we first describe the design of our online method that overcomes the stringent memory restrictions imposed by the GPU platform to evaluate large number of trajectories as they are numerically generated. We then discuss how the SMC procedure described in Section 4.3 is implemented in our setting using latency hiding. This will lead to a GPU implementation of the parameter estimation problem first presented in Section 4.4.

Our online approach uses the automaton constructed in Section 5.2, which eliminates the need for handling different formulas explicitly. Recall that running an automaton $\mathcal{A}$ is equivalent to evaluating the corresponding BLTL formula under a series of valuations at different time points until a final state is reached. To efficiently implement this on GPU, branch divergence should be avoided as much as possible. Our solution is to index states, variables and the atomic propositions as defined in Section 5.2, and encode the transitions and the operations on the valuations into an array A_T. This array represents transitions and operations on the valuations, in which each row corresponds to an input state, and each column to an atomic proposition. Each element of the array consists of an output state and the operations on the valuations associated to the transition. Each GPU thread has access to A_T which is pre-computed and stored in the shared memory. A step in the run of the automaton is performed by all threads of a warp executing in lock-step updating the state and the variables according to A_T. Note that dummy self-loops for the terminal states are added so that once one of them is reached, the automaton stays there forever. This avoids explicit checking for termination, which induces branch divergence.

Example 3 *For the fragment of the automaton A_ψ defined in Figure 12, the array*

$$A_T = \begin{array}{c} \\ s_{in} \\ s_1 \\ s_2 \\ \top \\ \bot \end{array} \begin{array}{c} \begin{array}{cc} \sigma_1 & \sigma_2 \end{array} \\ \left[\begin{array}{cc} s_1, a_{01} & s_2, a_{02} \\ s_1, a_{11} & s_2, a_{12} \\ s_1, a_{21} & s_2, a_{22} \\ \top, \{\} & \top, \{\} \\ \bot, \{\} & \bot, \{\} \end{array}\right] \end{array}$$

encodes the automaton, where $\sigma_1 = \{p \mapsto \mathit{false}\}$ *and* $\sigma_2 = \{p \mapsto \mathit{true}\}$*, and* a_{ij} *updates the set of time variables for the jth transition out of the ith state.*

The GPU computation steps are shown in Figure 13. Starting from an initial state at $t = 0$, for each time interval Δt, the new value of a variable x is determined by applying numerical integration using the current values of the variables and the values of the rate constants appearing in the ODE for x as well as the current value of x. Since trajectories are generated through numerical integration, to ensure numerical accuracy, each interval $[0, \Delta t]$ is uniformly subdivided into q sub-intervals for a suitable choice of q. We compute an updated value of the variables every $\lambda = \frac{\Delta t}{q}$. Each variable may appear in multiple equations, leading to a large amount of read-sharing. To ensure consistency, all variables are updated together in an atomic transaction. We use a fourth-order Runge-Kutta algorithm to compute the next value of a variable for each timestep. Overall, each trajectory is numerically simulated for $q \cdot K$ steps. Finally, the current values of the variables sampled at each of the time points $\{0, \Delta t, \ldots, K.\Delta t\}$ are used to update the atomic propositions and in turn execute a step in the run of the set of automata.

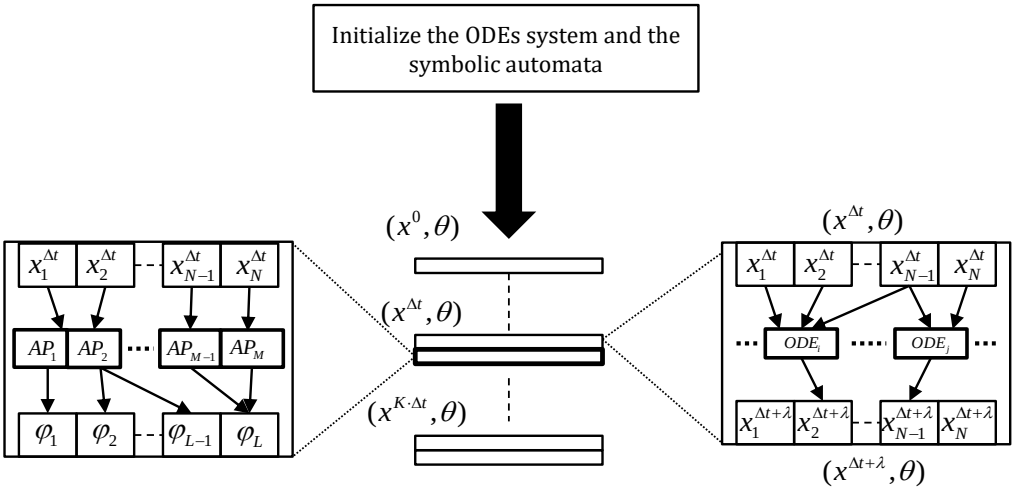


Figure 13: Lock step execution of the numerical integration and the symbolic BLTL automata

Our code generation scheme for the multi-thread based numerical simulation of an ODEs system distributes the fine-grained computation involved in the computation of a variable across different warps and obtains feedback regarding the quality of the computational balance by analyzing the PTX assembly generated at compile time. This information is then used to efficiently distribute the computation blocks across threads. The load balancing and latency hiding methods described here are based on the automatic code generation scheme described in Chapter 3. Moreover at a higher level, we generate a number of blocks of trajectories in parallel and the blocks are distributed across a number of GPU cores. At each time step, for each trajectory, we update the current state of the constructed deterministic automaton running in lockstep with the numerical integration. We also periodically check if the trajectories have hit a final state in the automaton. When this is the case, we update this state information for all the trajectories in the block to the global memory.

If threads from other warps are also scheduled for such long latency global memory accesses, the memory access delay due to control flow divergence will impact performance. To get around this, we use a latency hiding technique where by the global memory accesses are prefetched by threads in a separate warp at the same time as when the other threads carry out the numerical integration.

At the global memory level, we first pick the terminal state of a trajectory uniformly at random and use it to update the current SPRT score. When the SMC procedure reaches a decision we stop the concurrent numerical integration.

5.3.1 *Parallelized parameter estimation based on PBLTL formulas*

We refer to Section 4.4, where we described how experimental data can be encoded as BLTL formulas. To do so we mildly extend the syntax of BLTL with the formulas of type $\psi_1 U^t \psi_2$ with the semantics: ψ_1 will hold exactly up to t time units from now at which point ψ_2 will hold. The construction of the automaton presented in Section 5.2 can be easily extended to handle this case.

As described in Section 4.4, using a suitable search strategy, we generate estimates for the unknown parameters and iterate the scheme. In order to estimate the quality of $\mathbf{w}$ —the current estimate of the values of the unknown parameters— we launch a fresh instance of the parallel SMC procedure, using $INIT_{\mathbf{w}}$, to verify $P_{\geq r}(\psi_{exp} \wedge \psi_{qlty})$ on the GPU network. Depending on the outcome of the test, we assign a score to $\mathbf{w}$ using the objective function in Section 4.4. This evaluation is done at the global memory level. Using a cloud service, one can launch as many parallel sets of SMC procedures as there are GPU instances available.

5.4 EXPERIMENTAL EVALUATION

We applied our method to three ODEs based pathway models taken from the BioModels database [88]. We first verified properties of interest on each of the three pathway models. Using our parallelized SMC framework, we then performed parameter estimation on these models. The GPU implementation was based on CUDA 5.0 runtime and tested on four NVidia Tesla K20m GPUs with 4.8 GB global memory, clocked at 706 MHz each. We compared the performance of our algorithm with that of a CPU based implementation on a PC with 3.4 Ghz Intel Core i7 processor with 8 GB of memory. The model checker and the numer-

ical solver for the CPU implementation were written in C++. On the GPU, we implemented the fourth order Runge-Kutta method (used for the EGF-NGF and segmentation clock model) and the adaptive step-size Runge-Kutta-Fehlberg method [89] (used for the thrombin pathway model). For the cloud implementation, we ported our single node implementation to 25 Amazon Web Service (AWS) cloud `g2.8xlarge` GPU nodes. Each such node has two Intel Xeon E5-2670 CPUs of 8 cores each and four NVIDIA GK104 GPUs with 60 GB host memory and 4 GB global memory on each GPU device. The nodes are connected by AWS Enhanced Networking and communicate using CUDA-aware OpenMPI. The NVIDIA GK104 GPUs have 1536 cores clocked at 797 MHz each with 4 GB global memory and a memory bandwidth of 160 GB/s.

5.4.1 *Case studies: Property verification*

5.4.1.1 *Thrombin dependent MLC-phosphorylation pathway*

Thrombin plays an important role in the contraction of endothelial cells through multiple pathways leading to the phosphorylation of MLC [90]. The pathway model has 105 differential equations and 197 kinetic parameters. Simulation time was fixed at 1000 seconds divided into 20 equally spaced time points. We used the nominal model (all rate parameter values known) to verify if it conformed to a property with a high probability expressed in BLTL. It is known experimentally that the concentration of MLC* (phosphorylated MLC) starts at a low level, and then reaches a high steady state value. The corresponding formula is

$$P_{\geq 0.9}(([\text{MLC}^* \leq 1]) \wedge F^{\leq 5}(G^{\leq 20}([\text{MLC}^* \geq 3]))).$$

Our SMC analysis concluded that the nominal model does not satisfy this property, and we found that phosphorylated MLC shows a tran-

sient profile. This discrepancy has been studied in [91], where it was attributed to missing components in the proposed model.

Our online procedure for this case achieves significant speed-up (4.6× in a single GPU setting) compared to an offline GPU based model checker which first generates trajectories in parallel, stores them in the global memory and then carries out the model checking procedure on the CPU.

5.4.1.2 *EGF-NGF pathway*

The EGF-NGF signaling pathway captures the differential response to two growth factors, EGF and NGF in the PC12 neuro-endocrine cell line [2]. EGF induces cell proliferation while NGF promotes cell differentiation. The difference in cell fate is attributed to the duration of Erk activation. For studying this model, simulation time was set to 61 minutes divided into equally spaced intervals of 1 minute each. We checked whether starting from a low value, the concentration of Erk^* (active Erk) reaches a high value and then begins to fall. This property can be formalized as

$$P_{\geq 0.9}([0 \leq \text{Erk}^* \leq 2.2 \cdot 10^5] \wedge F^{\leq 10}([4.8 \cdot 10^5 \leq \text{Erk}^* \leq 5.6 \cdot 10^5]) \\ \wedge F^{\leq 20}(G^{\leq 30}([2.2 \cdot 10^5 \leq \text{Erk}^* \leq 4.8 \cdot 10^5]))).$$

The property was confirmed to be true by our SMC method suggesting that Erk shows sustained activation upon EGF stimulation.

5.4.1.3 *Segmentation clock pathway*

The segmentation pattern of the spine in developing embryos is controlled by oscillations in Notch, Wnt and FGF signaling due to coupled feedback loops [64]. The ODEs model representing this pathway was simulated up to 200 minutes with observations assumed to be available

every 5 minutes. We formulated the oscillations observed in the concentration profile of Dusp6-mRNA as a BLTL property as follows

$$P_{\geq 0.9}([\text{Dusp6 mRNA} \leq 1] \wedge (F^{\leq 10}([\text{Dusp6 mRNA} \geq 5.5] \wedge F^{\leq 10}([\text{Dusp6 mRNA} \leq 1] \wedge F^{\leq 10}([\text{Dusp6 mRNA} \geq 5.5])))))).$$

This property was verified to be true suggesting oscillations in Dusp6 mRNA with a period of approximately 100 minutes.

5.4.2 *Case studies: Parameter estimation*

We next evaluated our method for estimating unknown model parameters based on a combination of quantitative time series data and qualitative specifications of dynamical trends. Using the nominal model we generated training data to be used for parameter estimation and an independent set of test data not used for fitting. To generate time series data points, we simulated random trajectories on the GPU by sampling initial concentration from a ± 5 % range around the nominal values. We also encoded the dynamic trends of a few species as properties in BLTL. Later, for each BLTL property, its respective symbolic automaton was constructed. We allowed 0.5% parameter variability around the current estimate of parameters in each iteration of the search procedure. Table 6 summarizes the key features of the models including the number of variables (N_x), the number of parameters (N_Θ), the number of parameters assumed to be unknown (N_{Θ_u}), the number of equally spaced time points (T) and for SRES, the total number of individuals (λ) and the number of generations (G).

For the thrombin pathway, all training data and test data were quantitative time course data with one exception. Namely, for Thrombin R*, a dynamical trend formulated in BLTL was used as training data expressing that it reaches a high level within 200 seconds and then falls to a

Table 6: Parameter estimation setup and model specifications

Bio-pathway model	N_x	N_Θ	N_{Θ_u}	T	λ	G
EGF-NGF	32	48	20	61	200	100
Segmentation clock	16	75	39	40	200	300
Thrombin	105	197	100	20	100	500

low level (Figure 14). We used only quantitative data for the EGF-NGF pathway and found a good fit to both training and test data by the fitted model (Figure 15). For the segmentation clock model, only Axin2 mRNA was assumed to have quantitative time course data available, and dynamical trends were given as training and test data for the remaining species. For instance, the test data for Dusp6 protein expresses that at least two peaks and troughs are reached within 200 minutes — this test property was satisfied by the fitted model as seen in Figure 16. In each case, the simulated dynamics of the fitted model is plotted by sampling randomly from the initial conditions while using the fitted parameter values.

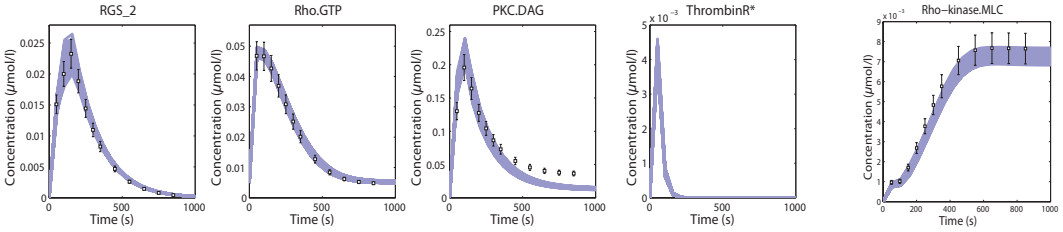


Figure 14: Parameter estimation of the thrombin pathway, showing model fit to (a) training data and (b) test data.

5.4.3 *Performance*

We measured the runtime of the parameter estimation procedure with different combinations of SPRT error bounds (α and β), indifference regions (δ), and threshold probability (r) used within the SMC procedure.

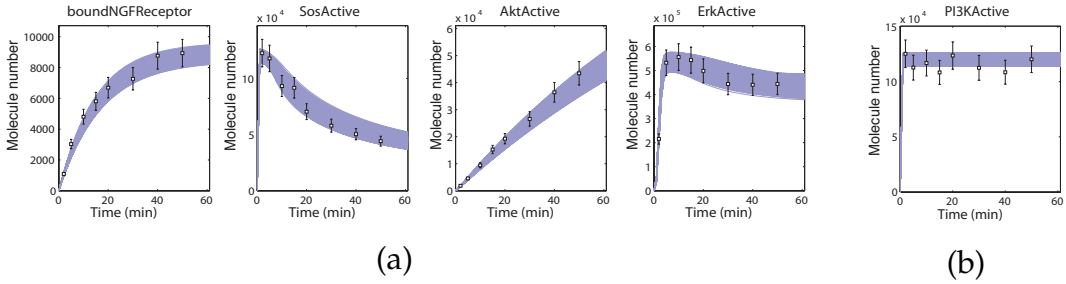


Figure 15: Parameter estimation of the EGF-NGF pathway, showing fit to (a) training data and (b) test data.

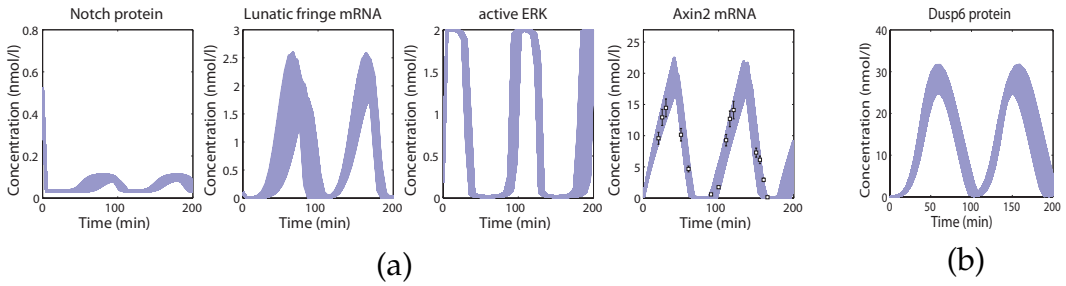


Figure 16: Parameter estimation of the segmentation clock pathway, showing fit to (a) training data and (b) test data.

We found that for all three models, while GPU runtimes stayed roughly constant across all SPRT parameter combinations, runtimes for the CPU based implementation increased significantly for more stringent statistical tests (see Figure 17). For instance with the most stringent statistical test, the GPU implementation took just 42 minutes for finding the best parameter set for the EGF-NGF model on a 4-GPU node, a 24.6× speed-up compared to the 17.2 hours taken by the CPU implementation.

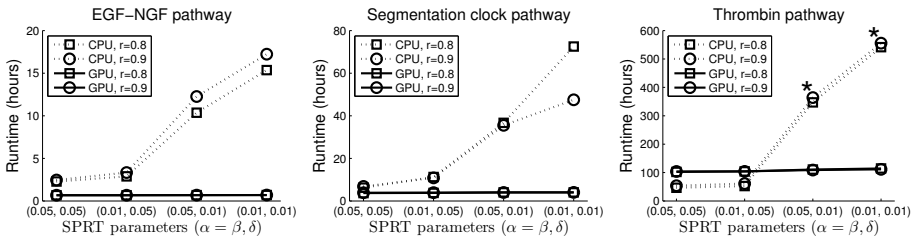


Figure 17: Comparison of CPU and GPU runtimes on parameter estimation with different combinations of SPRT parameters (error bounds $\alpha = \beta$, indifference region δ and probability threshold r). *Estimated values based on shorter runs

Next, Table 7 shows the performance of our parameter estimation method on a range of parallel architectures with the SPRT parameters

Table 7: Performance of our scheme across different architectures (*Estimated values based on shorter runs)

Model	CPU [hr]	4-GPU node [hr]	100-GPU cloud [hr]	4-GPU node over CPU
EGF-NGF	17.22	0.69	0.05	24.6×
Segmentation clock	47.5	4.01	0.45	11.9×
Thrombin	556.8*	111.1	5	5×*

Table 8: Strong scaling performance of the cloud based implementation

Bio-pathway model	40-GPUs Time[s]	80-GPUs over 40-GPUs	100-GPUs over 40-GPUs
EGF-NGF	445.28	1.62x	2.36x
Segmentation clock	3864.74	1.74x	2.35x

set to $\alpha = \beta = \delta = 0.01$ and $r = 0.9$. In the 4-GPU server setup, for every generation in our single node parallel implementation, we divided the total number of individuals across 4 GPUs equally. For the cloud based implementation, the set of individuals were divided across 100 GPU instances in 25 machines with 4 GPUs per node.

While the 4-GPU server implementation took 42 minutes to complete the EGF-NGF parameter estimation task, the same took only 3 minutes on the 100-GPU cloud. For the segmentation clock pathway, the 4-GPU implementation took 4 hours, a speed-up of approximately 11.9× over the CPU implementation. Finally, parameter estimation for the thrombin model would take an estimated 23.2 days using a CPU based implementation. (Note that this estimate was obtained by running an initial number of generations in the parameter search, calculating the average time taken for a generation, and then extrapolating the run time for the maximal generation number.) The cloud based implementation on the other hand is able to estimate the parameters in about 5 hours.

Finally, Table 8 presents the scaled performance of our parameter estimation method applied on the EGF-NGF and the segmentation clock pathway models on the cloud. As might be expected our method achieves near perfect linear scaling when all the individuals in each round of the SRES procedure are launched on unique instances on the cloud.

5.5 SUMMARY

In this chapter, we proposed a technique for studying the dynamics of ODEs systems that utilizes the power of commodity graphics processors. In particular, we developed a parallel, online procedure for checking if the trajectories of this model satisfy a bounded linear temporal logic formula. Our procedure works around various architectural constraints of the graphics processor execution model to achieve significant performance both on local systems as well as in the cloud. We believe that this opens the door for studying large pathway models in a scalable and cost-effective manner. We used the parameter estimation problem to illustrate the applicability of our method, which consists of a parallel SMC procedure whose core is a deterministic online model checking procedure that determines if the trajectory under construction satisfies a given BLTL formula. Many analysis questions can be tackled by assuming a distribution over the set of initial concentrations and parameter values, which will then induce a probability measure on the set of trajectories satisfying a given BLTL formula.

6

STATISTICAL MODEL CHECKING OF HYBRID SYSTEMS

In this chapter, we consider the analysis of hybrid systems. A rich class of biological, cyber-physical and engineering systems can be naturally modelled as hybrid systems [25, 92, 93]. Hybrid systems are multi-mode dynamical systems which evolve over continuous time and have instances where jumps between discrete states occur. Generally, the evolution "flows" in continuous time according to a set of ordinary differential equations (ODEs) associated with a given state. At some instance, the system jumps from a current state to a new state in which the system now evolves according to a different set of ODEs associated with the new state.

The interaction between the discrete and the continuous components makes analysis of hybrid systems intractable. Even very basic analysis questions for simple hybrid systems become undecidable [47, 94]. Various lines of work have explored ways to mitigate this problem with a common technique being to restrict the mode dynamics [95–100]. However, for many of the models arising in systems biology or engineering, the continuous dynamics will be governed by a system of non-linear ODEs. So one must look for approximate methods for their analysis. In this chapter, we first describe a probabilistic approximation method for analysis of a hybrid system in which such a system can be approximated as a discrete-time Markov chain. We then develop a statistical model checking procedure based on this approximation.

Once we relate the behaviours of the Markov chain and the hybrid system, we can randomly sample paths in the Markov chain according to the underlying transition probabilities and approximately verify time bounded properties of the Markov chain.

6.1 OVERVIEW

A key difficulty in analyzing a hybrid system's behaviour is that the time points and value states at which a trajectory meets a guard will depend on the solutions to the ODE systems associated with the modes. For high-dimensional systems these solutions will not be available in closed-form. To get around this, we approximate the mode transitions as stochastic events by fixing the probability of a mode transition to be proportional to the measure of the set of value state and time point pairs at which this transition is enabled. More sophisticated hypotheses could be considered. For instance one could tie the mode transition probability to how long the guard has been continuously enabled or how deeply within a guard region the current state is.

6.1.1 *Assumptions*

To secure a sound mathematical basis for our approximation, we make the following crucial assumptions.

- It is impossible, if not impractical to observe the system continuously. So we assume the states of the system are observed only at discrete time points. Fixing a suitable unit time interval Δ, we discretize the time domain as $t = 0, \Delta, 2\Delta, \ldots$.

- The vector fields associated with the ODEs are Lipschitz continuous functions.
- The set of initial states and the guard sets are bounded open sets.
- A hybrid system is said to contain Zeno executions if the system makes an infinite number of mode switches in a finite time interval. In reality, engineering or biological systems are non-Zeno but design flaws in modelling or over-abstraction may introduce Zeno behaviour. The hybrid systems we consider are strictly non-Zeno in the sense that there is a uniform upper bound on the number of transitions that can take place in a unit time interval. For technical convenience we in fact assume that time discretization is so chosen that at most one mode transition takes place between two successive discrete time points.

Under these assumptions, we show that the dynamics of the hybrid system H can be approximated as an infinite-state Markov chain M. Given the application domain we have in mind, namely, biological pathway dynamics, we focus on the behaviour of the hybrid system over a finite time horizon and BLTL (bounded linear-time temporal logic) [101] to specify dynamic properties of interest. The maximum discrete time point we fix will be determined by the BLTL specification. Our probabilistic approximation is such that the set of trajectories satisfying a BLTL formula will correspond to a measurable set of paths of the Markov chain and hence can be assigned a probability value. We then show that H meets the specification ψ–i.e. every trajectory of H is a model of ψ–iff M meets the specification ψ with probability 1. This allows us to approximately verify interesting properties of the hybrid system using its Markov chain approximation. However, even a bounded portion of M can not be constructed effectively. This is because the transition proba-

bilities of the Markov chain will depend on the solutions to the ODEs associated with the modes, which will not be available in a closed-form. In addition, the structure of M itself will be unknown since the states of the chain will be those that can reached with non-zero probability from the initial mode and we can not determine which transitions have non-zero probabilities. To cope with this, we design a statistical model checking procedure to approximately verify that the chain (and hence the hybrid system) almost certainly meets the specification. One just needs to ensure that the dynamics of the Markov chain is being sampled according to underlying probabilities. We achieve this by randomly generating trajectories of H through numerical simulations in a way that corresponds to randomly sampling the paths of the Markov chain according to its underlying structure and transition probabilities. We note that a simple minded Monte Carlo simulations based strategy consisting of sampling an initial state (according to the given initial distribution) followed by a random generation of trajectory will flounder on the issue of how one "randomly" picks a mode transition during the generation of a trajectory in the presence of the non-linear dynamics captured by the ODEs systems. Our approximation technique instead establishes a principled way of achieving this.

In establishing these results, we assume that the atomic propositions in the specification are interpreted over the modes of the hybrid system. Consequently one can specify patterns of mode visitations while quantitative properties can be inferred only indirectly and in a limited fashion. Our results however can be extended to handle quantitative atomic propositions ("the current concentration of protein X is greater than 2 μM"). The details of this extension can be found in the Appendix A.1.

6.1.2 *Related work*

A well studied subclass of hybrid automata is timed automata. In this formalism, the continuous dynamics which model time are defined by variables having derivative 1. Though the continuous dynamics is restricted, timed automata are interesting since a rich class of real-time systems can be modelled in this formalism. Moreover, the reachability problem for timed automata being decidable, model checking problem can be solved exactly [102]. But in a more general setting, analyzing classical hybrid automata is hard for complex systems.

The continuous dynamics associated with a mode often makes it difficult to pin down mode transitions. On the other hand, it is important to note that restricting the discrete mode dynamics or the continuous dynamics of hybrid automata is often unrealistic. So approximation techniques are called for. Mode transitions have been approximated as random events in the literature. In [103], the dynamics of a hybrid system is approximated by substituting the guards with probabilistic barrier functions. When a vector field approaches a guard, integration steps are dynamically adapted to precisely detect whether a mode switch occurs.In our approximation scheme, the transition probabilities are constructed using similar but simpler considerations. We have done so in order to be able to carry out temporal logic based verification based on simulations.

In [104], Julius and Pappas describe an approximation scheme for a restricted class of hybrid automata namely stochastic linear hybrid automata. The approximation is based on the assumption that there exists a stochastic bisimulation function which is quadratic in nature. A bisimulation function provides a sufficient condition for the existence of a simpler automaton with one state which approximates the dynamics

of the stochastic linear hybrid automaton. The authors also hint that though the approximation scheme is not restricted in theory, the computational implementation does not take non-determinism in the model into account.

Another relevant related work is [105], which studies a network of hybrid automata that communicate with each other through input/output actions. The main idea roughly boils down to: the time point in $(0, 1)$, at which the decision about what the mode should be up to the next discrete time point, is determined by the uniform distribution over $(0, 1)$. In our setting, however, this probability is determined by the intersection of the continuous mode dynamics with the guard sets.

An alternative approach to approximately verifying non-linear hybrid systems is one based on δ-reals [106]. Here one verifies bounded reachability properties that are robust under small perturbations of the numerical values mentioned in the specification. Since the approximation involved is of a very different kind, it is difficult to compare this line of work with ours. However, it may be fruitful to combine the two approaches to verify a richer set of reachability properties.

Ballarini et al. developed statistical model checking tool for stochastic processes based on an extension of continuous stochastic logic (CSL). In [107], the temporal logic for expressing properties of stochastic processes, namely Hybrid Automata Stochastic Logic (HASL) is based on acceptance of a path in the linear hybrid automaton synchronized with the probabilistic model. But the continuous dynamics in the framework does not consider ODEs based models.

The present work may be viewed as an extension of [24] where a *single* system of ODEs is considered. This method however, breaks down in the multi-mode hybrid setting and an entirely new machinery is required to tackle hybrid behaviours. Finally, a wealth of literature is

available on the analysis of stochastic automata [104, 107–109]. It will be interesting to explore if these methods can be transported to our setting.

6.2 HYBRID AUTOMATA

We fix n real-valued variables $\{x_i\}_{i=1}^n$ viewed as functions of time $x_i(t)$ with $t \in \mathbb{R}_+$, the set of non-negative reals. A valuation of $\{x_i\}_{i=1}^n$ is $\mathbf{v} \in \mathbb{R}^n$ with $\mathbf{v}(i) \in \mathbb{R}$ representing the value of x_i. The language of *guards* is given by: (i) $a < x_i$ and $x_i < b$ are guards where a, b are rationals and $i \in \{1, 2, \ldots, n\}$. (ii) If g and g' are guards then so are $g \wedge g'$ and $g \vee g'$.

$\mathcal{G}$ denotes the set of guards. We define $\mathbf{v} \models g$ (i.e. $\mathbf{v}$ satisfies the guard g) via: $\mathbf{v} \models a < x_i$ iff $a < \mathbf{v}(i)$ and similarly for $x_i < b$. The clauses for conjunction and disjunction are standard. We let $\|g\| = \{\mathbf{v} \mid \mathbf{v} \models g\}$. We note that $\|g\|$ is an open subset of $\Re^n$ for every guard g. We will abbreviate $\|g\|$ as g.

Definition 2 A hybrid automaton is a tuple $H = (Q, q_{in}, \{F_q(\mathbf{x})\}_{q \in Q}, \mathcal{G}, \rightarrow, \text{INIT})$, where

- Q is a finite set of *modes* and $q_{in} \in Q$ is the *initial* mode.
- For each $q \in Q$, $d\mathbf{x}/dt = F_q(\mathbf{x})$ is a system of ODEs, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $F_q = (f_q^1(\mathbf{x}), f_q^2(\mathbf{x}), \ldots, f_q^n(\mathbf{x}))$. Further, f_q^i is Lipschitz continuous for each i.
- $\rightarrow \subseteq (Q, \mathcal{G}, Q)$ is the mode transition relation. If $(q, g, q') \in \rightarrow$ we shall often write it as $q \xrightarrow{g} q'$.
- $\text{INIT} = (L_1, U_1) \times (L_2, U_2) \ldots \times (L_n, U_n)$ is the set of initial states where $L_i < U_i$ and L_i, U_i are rationals.

We have not associated invariant conditions with the modes or reset conditions with the mode transitions. They can be introduced with some additional work.

Fixing a suitable unit time interval Δ, we discretize the time domain as $t = 0, \Delta, 2\Delta, \ldots$. We assume the states of the system are observed only at these discrete time points. Furthermore, we shall assume that only a bounded number of mode changes can take place between successive discrete time points. Both in engineered and biological processes this is a reasonable assumption. Given this, we shall in fact assume that Δ is such that at most one mode change takes place within a Δ time interval. We note that there can be multiple choices for Δ that meet this requirement and in practice one must choose this parameter carefully. (Our method can be extended to handle a bounded number of mode transitions in a unit time interval but this will entail notational complications that will obscure the main ideas.) In what follows, for technical convenience we also assume the time scale has been normalized so that $\Delta = 1$. As a result, the discretized set of time points will be $\{0, 1, 2, \ldots\}$.

6.2.1 *Trajectories*

In what follows, we fix a hybrid automaton H as defined in 6.2. The behaviour of H will consist of its finite trajectories. To define this notion and for later use, we start with some preliminaries. We recall that a function $f : \mathbb{R} \to \mathbb{R}$ is C^1 if f', the derivative of f, exists everywhere and is continuous. This notion extends to $\mathbb{R}$ in the obvious way. Further, the function $F : \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz if there exists a $c \in \mathbb{R}$, $c > 0$, such that for all $\mathbf{v}_1, \mathbf{v}_2 \in \mathbb{R}^n$, $|F(\mathbf{v}_1) - F(\mathbf{v}_2)| \leq c|\mathbf{v}_1 - \mathbf{v}_2|$, where $|\cdot|$ is the standard Euclidean norm on $\mathbb{R}^n$.

We have assumed that for every mode q, the right hand side of the ODEs, $F_q(\mathbf{x})$, is Lipschitz continuous for each component. As a result, for each initial value $\mathbf{v} \in \mathbb{R}^n$ and in each mode q, the system of ODEs $d\mathbf{x}/dt = F_q(\mathbf{x})$ will have a unique solution $Z_{q,\mathbf{v}}(t)$ [20]. We are also guaranteed that $Z_{q,\mathbf{v}}(t)$ is Lipschitz and hence measurable [20]. It will be convenient to work with two sets of functions derived from solutions to the ODE systems.

The (unit interval) *flow* $\Phi_q : (0,1) \times \mathbb{R}^n \to \mathbb{R}^n$ is given by $\Phi_q(t, \mathbf{v}) = Z_{q,\mathbf{v}}(t)$. Φ_q will also be Lipschitz. Next we define the parametrized family of functions $\Phi_{q,t} : \mathbb{R}^n \to \mathbb{R}^n$ given by $\Phi_{q,t}(\mathbf{v}) = \Phi_q(t, \mathbf{v})$. Applying once again the fact that the RHS of the ODEs are Lipschitz continuous functions, we can conclude that these parametrized functions $\Phi_{q,t}$ (which will be $1-1$) as well as their inverses will be Lipschitz.

Definition 3 A (finite) *trajectory* is a sequence $\tau = (q_0, \mathbf{v}_0)\,(q_1, \mathbf{v}_1)\,\ldots\,(q_k, \mathbf{v}_k)$ such that for $0 \leq j < k$ the following conditions are satisfied:

(i) For $0 \leq j < k$, $q_j \xrightarrow{g_j} q_{j+1}$ for some guard g_j.

(ii) There exists $t \in (0,1)$ such that $\Phi_{q_j,t}(\mathbf{v}_j) \in g$. Furthermore $\mathbf{v}_{j+1} = \Phi_{q_{j+1},1-t}(\Phi_{q_j,t}(\mathbf{v}_j))$.

We say that the trajectory τ as defined above *starts* from q_0 and *ends* in q_k. Further, its initial value state is $\mathbf{v}_0$ and its final value state is $\mathbf{v}_k$. We let TRJ denote the set of all finite trajectories that start from the initial mode q_{in} with an initial value state in INIT.

6.3 THE MARKOV CHAIN APPROXIMATION

A (finite) path in H is a sequence $\rho = q_0 q_1 \ldots q_k$ such that for $0 \leq j < k$, there exists a guard g_j such that $q_j \xrightarrow{g_j} q_{j+1}$. We say that this path starts

from q_0, ends at q_k and is of length $k+1$. We let paths_H denote the set of all finite paths that start from q_{in}.

In what follows μ will denote the standard Lebesgue measure over finite dimensional Euclidean spaces. We will construct $M_H = (\mathrm{Y}, \Rightarrow)$, the Markov chain approximation of H inductively. Each state in Y will be of the form $(\rho, X, \mathbf{P}_X)$ with $\rho \in \mathsf{paths}_H$, X an open subset of $\mathbb{R}^n$ of non-zero, finite measure and $\mathbf{P}_X$ a probability distribution over $SA(X)$, the Borel σ-algebra generated by X.

We start with $(q_{in}, \mathrm{INIT}, \mathbf{P}_{\mathrm{INIT}}) \in \mathrm{Y}$. Clearly, INIT is an open set of non-zero, finite measure since $\mu(\mathrm{INIT}) = \prod_i (U_i - L_i)$. For technical convenience we shall assume $\mathbf{P}_{\mathrm{INIT}}$ to be the uniform probability distribution, but other probability distributions with respect to which the Lebesgue-measure is absolutely continuous could also be chosen. Assume inductively that $(\rho, X, \mathbf{P}_X)$ is in Y with X an open subset of $\mathbb{R}^n$ of non-zero, finite measure and $\mathbf{P}_X$ a probability distribution over $SA(X)$. Suppose ρ ends in q and there are m outgoing transitions $q \xrightarrow{g_1} q_1, \ldots, q \xrightarrow{g_m} q_m$ from q in H (Figure 18 illustrates this inductive step).

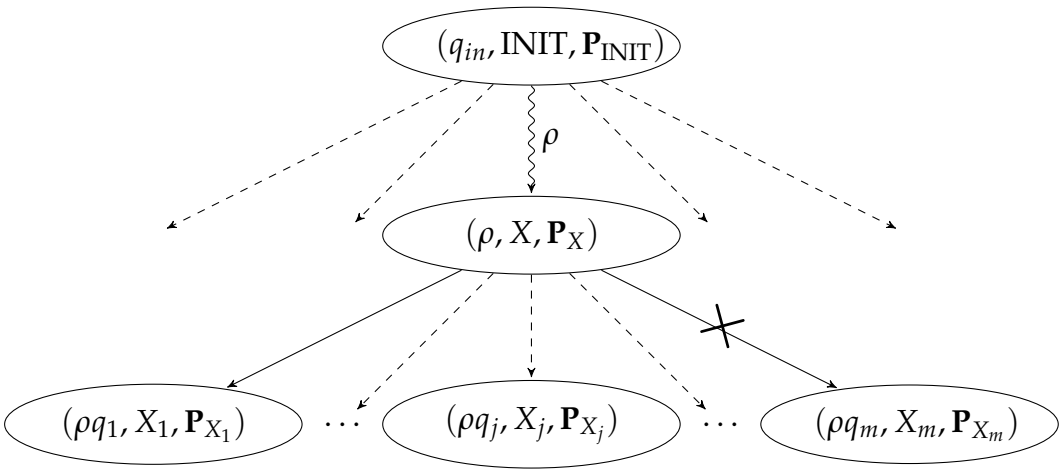


Figure 18: The Markov chain construction. The edge from the state $(\rho, X, \mathbf{P}_X)$ to the state $(\rho q_m, X_m, \mathbf{P}_{X_m})$ marked with a '$\times$' represents the case where X_m has measure 0, and hence the probability of this transition is 0. Thus, $(\rho q_m, X_m, \mathbf{P}_{X_m})$ will not be a state of the Markov chain.

Then for $1 \leq j \leq m$ we define the triples $(\rho q_j, X_j, \mathbf{P}_{X_j})$ as follows. In doing so we will assume the required properties of the objects involved in this construction. We will then establish these properties and thus the soundness of the construction. For convenience, through the remaining parts of this section j will range over $\{1, 2, \ldots, m\}$.

For each $\mathbf{v} \in X$ and each j we first define the set of time points $\mathbb{T}_j(\mathbf{v}) \subseteq (0, 1)$ via

$$\mathbb{T}_j(\mathbf{v}) = \{t \mid \Phi_q(t, \mathbf{v}) \in g_j\}. \tag{5}$$

Thus $\mathbb{T}_j(\mathbf{v})$ is the set of time points in $(0, 1)$ at which the guard g_j is satisfied if the system starts from $\mathbf{v}$ in mode q and evolves according to dynamics of mode q up to time t. We next define X_j for each j as

$$X_j = \bigcup_{\mathbf{v} \in X} \{\Phi_{q_j}(1 - t, \Phi_q(t, \mathbf{v})) \mid t \in \mathbb{T}_j(\mathbf{v})\}. \tag{6}$$

Thus X_j is the set of all value states obtained by starting from some $\mathbf{v} \in X$ at time k, evolving up to $k + t$ according to the dynamics q, making an instantaneous mode switch to q_j at this time point, and evolving up to time $k + 1$ according to dynamics of mode q_j.

To complete the definition of the triples $(\rho q_j, X_j, \mathbf{P}_{X_j})$, we first denote by $\mathbf{P}_{\mathbb{T}_j(\mathbf{v})}$ a probability distribution over $\mathbb{T}_j(\mathbf{v})$. We shall choose this distribution to be uniform but it could be any other non-uniform probability distribution with respect to which the Lebesgue-measure is absolutely continuous. We now define the probability distributions $\mathbf{P}_{X_j}$ over $SA(X_j)$ as follows. Suppose Y is a measurable subset of X_j. Then

$$\mathbf{P}_{X_j}(Y) = \int_{\mathbf{v} \in X} \int_{t \in \mathbb{T}_j(\mathbf{v})} \mathbf{1}_{(\Phi_{q_j}(1-t, \Phi_q(t, \mathbf{v})) \cap Y)} d\mathbf{P}_{\mathbb{T}_j(\mathbf{v})} d\mathbf{P}_X. \tag{7}$$

As usual $\mathbf{1}_Z$ is the indicator function of the set Z while $d\mathbf{P}_{\mathbb{T}_j(\mathbf{v})}$ indicates that the inner integration over $\mathbb{T}_j(\mathbf{v})$ is w.r.t. the probability measure $\mathbf{P}_{\mathbb{T}_j(\mathbf{v})}$ and $d\mathbf{P}_X$ indicates that the outer integration over X is w.r.t. the probability measure $\mathbf{P}_X$. Thus $\mathbf{P}_{X_j}(Y)$ captures the probability that the value state $\Phi_{q_j}(1-t, \Phi_q(t, \mathbf{v}))$ lands in $Y \subseteq X_j$ by taking the transition $q \xrightarrow{g_j} q_j$ at some time point in $\mathbb{T}_j(\mathbf{v})$ given that one started with some value state in X.

Next we define the triples $((\rho, X, \mathbf{P}_X), p_j, (\rho q_j, X_j, \mathbf{P}_{X_j}))$, where p_j is given by

$$p_j = \int_{\mathbf{v} \in X} \frac{\mu(\mathbb{T}_j(\mathbf{v}))}{\sum_{\ell=1}^{m} \mu(\mathbb{T}_\ell(\mathbf{v}))} d\mathbf{P}_X. \tag{8}$$

Thus p_j captures the probability of taking the mode transition $q \xrightarrow{g_j} q_j$ when starting from the value states in X and mode q. For every j we add the state $(\rho q_j, X_j, \mathbf{P}_{X_j})$ to Y and the triple $((\rho, X, \mathbf{P}_X), p_j, (\rho q_j, X_j, \mathbf{P}_{X_j}))$ to $\Rightarrow$ iff $\mu(X_j) > 0$.

Finally, $(q_{in}, \text{INIT}, \mathbf{P}_{\text{INIT}})$ is the initial state of M_H. We can summarize the key properties of our construction as follows (while assuming the associated terminology and notations).

Theorem 1 1. *$\mathbb{T}_j(\boldsymbol{v})$ is an open set of finite measure for each $\boldsymbol{v} \in X$ and each j.*

2. *X_j is open and is of finite measure for each j.*

3. *If $(\rho q_j, X_j, \boldsymbol{P}_{X_j}) \in \text{Y}$ then $\mu(X_j) > 0$.*

4. *$\boldsymbol{P}_{X_j}$ is a probability distribution for each j.*

5. *$M_H = (\text{Y}, \Rightarrow)$ is an infinite-state Markov chain whose underlying graph is a finitely branching tree.*

PROOF To prove the first part, suppose $t \in \mathbb{T}_j(\mathbf{v})$. Then $\Phi_q(t, \mathbf{v}) = \mathbf{v}' \in g_j$ and g_j is open. Hence $\mathbf{v}'$ will be contained in an open neighborhood U contained in g_j. Since Φ_q is Lipschitz we can pick U such that $Y' = \Phi_q^{-1}(U)$ is an open set containing $(\mathbf{v}, t)$ with $Y' \subseteq (0,1) \times X$. Thus every element of $\mathbb{T}_j(\mathbf{v})$ is contained in an open neighborhood in $(0,1)$ and hence $\mathbb{T}_j(\mathbf{v})$ is open.

Using the definition of X_j, the fact that X and $\mathbb{T}_j(\mathbf{v})$ are open, and the continuity of the inverses of the flow functions it is easy to observe that X_j is open. To see that it is of finite measure, by the induction hypothesis, X is open and $\mu(X)$ is finite. Hence $((0,1) \times X)$ is open as well and $\mu((0,1) \times X)$ is finite. Since $\mathbb{R}^{n+1}$ is second-countable [110], there exists a countable family of disjoint open-intervals $\{I_i\}_{i \geq 1}$ in $\mathbb{R}^{n+1}$ such that $((0,1) \times X) = \bigcup_i I_i$. Clearly each I_i has a finite measure. By the Lipschitz continuity of Φ_q we know that there exists a constant c such that $\mu(\Phi_q(I_i)) < c \cdot \mu(I_i)$ for all i. We thus have

$$\begin{aligned} \mu(\Phi_q((0,1), X)) &\leq \sum_i \mu(\Phi_q(I_i)) \\ &< c \sum_i \mu(I_i) = c\mu((0,1) \times X) < \infty. \end{aligned} \tag{9}$$

Therefore $\Phi_q((0,1), X)$ has a finite measure. By a similar argument we can show that $\Phi_{q_j}((0,1), \Phi_q((0,1), X))$ has a finite measure as well. Since $X_j = \bigcup_t \Phi_{q_j, 1-t}(\Phi_{q,t}(X) \cap g) \subseteq \Phi_{q_j}((0,1), \Phi_q((0,1), X))$, it must have a finite measure.

The remaining parts of the theorem follow easily from the definitions and basic measure theory. □

6.4 RELATING THE BEHAVIOURS OF H AND M_H

In order to give a framework of comparison between the hybrid automaton H and the constructed Markov chain M_H, we shall use bounded linear-time temporal logic (BLTL) [101] to specify time bounded properties and use it to relate the behaviours of H and M_H. For convenience we shall write M instead of M_H from now on.

We assume a finite set of atomic propositions AP and a valuation function $Kr : Q \to 2^{AP}$. Formulas of BLTL are defined as: (i) Every atomic proposition as well as the constants *true*, *false* are formulas. (ii) If ψ, ψ' are formulas then $\neg\psi$ and $\psi \vee \psi'$ are formulas. (iii) If ψ, ψ' are formulas and ℓ is a positive integer then $\psi \mathbf{U}^{\leq \ell} \psi'$ is a formula. The derived operators $\mathbf{F}^{\leq \ell}$ and $\mathbf{G}^{\leq \ell}$ are defined as usual: $\mathbf{F}^{\leq \ell}\psi \equiv \mathit{true}\mathbf{U}^{\leq \ell}\psi$ and $\mathbf{G}^{\leq \ell}\psi \equiv \neg\mathbf{F}^{\leq \ell}\neg\psi$.

We shall assume through the rest of the paper that the behaviour of the system is of interest only up to a maximum time point $K > 0$. This is guided by the fact that given a BLTL formula ψ there is a constant K_ψ that depends only on ψ so that it is enough to evaluate an execution trace of length at most K_ψ to determine whether ψ is satisfied [87]. Hence we assume that a sufficiently high K has been chosen to handle the specifications of interest. Having fixed K, we denote by TRJ^{K+1} the trajectories of length $K+1$, and view this set as representing the time bounded non-deterministic behaviour of H of interest.

To develop the corresponding notion for M, we first define a finite path in M to be a sequence $\eta_0\eta_1 \ldots \eta_k$ such that $\eta_j \in \Upsilon$ for $0 \leq j \leq k$. Furthermore for $0 \leq j < k$ there exists $p_j \in (0,1]$ such that $\eta_j \overset{p_j}{\Rightarrow} \eta_{j+1}$. Such a path is said to start from η_0 and its length is $k+1$. We define paths_M to be the set of finite paths that start from the initial state of M while paths_M^{K+1} is the set of paths in paths_M of length $K+1$.

THE TRAJECTORY SEMANTICS: Let $\tau = (q_0, \mathbf{v}_0)\ (q_1, \mathbf{v}_1)\ \ldots\ (q_k, \mathbf{v}_k)$ be a finite trajectory, ψ a BLTL formula and $0 \leq j \leq K$. Then $\tau, j \models_H \psi$ is defined via:

- $\tau, j \models_H A$ iff $A \in Kr(q_j)$, where A is an atomic proposition.
- $\neg$ and $\vee$ are interpreted in the usual way.
- $\tau, j \models_H \psi \mathbf{U}^{\leq \ell} \psi'$ iff there exists j' such that $j' \leq \ell$ and $j + j' \leq k$ and $\tau, (j + j') \models_H \psi'$. Further, $\tau, (j + j'') \models_H \psi$ for every $0 \leq j'' < j'$.

We now define $models_H(\psi) \subseteq TRJ^{K+1}$ via: $\tau \in models_H(\psi)$ iff $\tau, 0 \models_H \psi$. We say that H *meets the specification* ψ, denoted $H \models \psi$, iff $models_H(\psi) = TRJ^{K+1}$.

THE MARKOV CHAIN SEMANTICS: Let $\pi = \eta_0 \eta_1 \ldots \eta_k$ be a path in M with $\eta_j = (\rho q_j, X_j, \mathbf{P}_{X_j})$ for $0 \leq j \leq k$. Let ψ be a BLTL formula and $0 \leq j \leq k$. Then $\pi, j \models_M \psi$ is given by:

- $\pi, j \models_M A$ iff $A \in Kr(q_j)$, where A is an atomic proposition.
- The remaining clauses are defined just as in the case of $\models_H$.

Now we define $models_M(\psi) \subseteq \mathsf{paths}_M^{K+1}$ via: $\pi \in models_M(\psi)$ iff $\pi, 0 \models_M \psi$. We can now define the probability of satisfaction of a formula in M. Let $\pi = \eta_0 \eta_1 \ldots \eta_K$ be in paths_M^{K+1}. Then $\Pr(\pi) = \prod_{0 \leq \ell < K} p_\ell$, where $\eta_\ell \overset{p_\ell}{\Rightarrow} \eta_{\ell+1}$ for $0 \leq \ell < K$. This leads to

$$\Pr(models_M(\psi)) = \sum_{\pi \in models_M(\psi)} \Pr(\pi).$$

We write $M \models \psi$ to denote $\Pr(models_M(\psi)) = 1$. For $p \in [0, 1]$ we write as usual $\Pr_{\geq p}(\psi)$ instead of $\Pr(models_M(\psi)) \geq p$. We note that $Pr(\pi) > 0$ for every $\pi \in models_M(\psi)$. Furthermore $\sum_{\pi \in models_M(\psi)} \Pr(\pi) \leq 1$. Hence $\Pr_{\geq 1}(\psi)$ iff $models_M(\psi) = \mathsf{paths}_M^{K+1}$ iff $M \models \psi$.

6.4.1 *The correspondence result*

We wish to show that H meets the specification ψ iff $\Pr_{\geq 1}(\psi)$. To this end let $\pi = \eta_0\eta_1 \ldots \eta_k$ be a path in M with $\eta_j = (q_0q_1 \ldots q_j, X_j, \mathbf{P}_{X_j})$ for $0 \leq j \leq k$ and let $\tau = (q'_0, \mathbf{v}_0)\ (q'_1, \mathbf{v}_1) \ldots (q'_{k'}, \mathbf{v}_{k'})$ be a trajectory. Then we say that π and τ are *compatible* iff $k = k'$ and $q_j = q'_j$ and $\mathbf{v}_j \in X_j$ for $0 \leq j \leq k$. The following three observations based on this notion will easily lead to the main result.

Lemma 1 *1. Suppose the path $\pi = \eta_0\eta_1 \ldots \eta_k$ in M and the trajectory $\tau = (q_0, \boldsymbol{v}_0)\ (q_1, \boldsymbol{v}_1) \ldots (q_k, \boldsymbol{v}_k)$ are compatible. Let $0 \leq j \leq k$ and ψ be a BLTL formula. Then $\pi, j \models_M \psi$ iff $\tau, j \models_H \psi$.*

2. Suppose π is a path in M. Then there exists a trajectory τ such that π and τ are compatible. Furthermore if $\pi \in \mathsf{paths}_M$ then $\tau \in TRJ$.

3. Suppose τ is a trajectory. Then there exists a path π in M such that τ and π are compatible. Furthermore if $\tau \in TRJ$ then $\pi \in \mathsf{paths}_M$.

PROOF The proof follows via a systematic application of the definitions. To prove the first part we note that if A is an atomic proposition then $\pi, j \models_M A$ iff $A \in Kr(q_j)$ iff $\tau, j \models_H A$. We next note that the suffix of length m of π will be compatible with the suffix of length m of τ whenever π and τ are compatible. The result now follows at once by structural induction on ψ.

To show the second part let $\pi = \eta_0\eta_1 \ldots \eta_k$ be a path in M with $\eta_j = (q_0q_1 \ldots q_j, X_j, \mathbf{P}_{X_j})$ for $0 \leq j \leq k$. Clearly X_j is non-empty for $0 \leq j \leq k$ since $\eta_j \in \mathrm{Y}$ implies $\mu(X_j) > 0$. We proceed by induction on k. If $k = 0$ then we can pick $\mathbf{v}_0 \in X_0$ and the trajectory $(q_0, \mathbf{v}_0)$ will be compatible with τ. So assume $k > 0$. Then by the induction hypothesis there exists a trajectory $(q_1, \mathbf{v}_1)(q_2, \mathbf{v}_2) \ldots (q_k, \mathbf{v}_k)$ which is compatible

with the path $\eta_1\eta_2\ldots\eta_k$. Let $q_0 \xrightarrow{g} q_1$. Since $\mathbf{v}_1 \in X_1$ there must exist $\mathbf{v}_0$ in X_0 and $t \in (0,1)$ such that $\Phi_{q_0,t}(\mathbf{v}_0) \in g$ and $\mathbf{v}_1 = \Phi_{q_1,1-t}(\Phi_{q_0,t}(\mathbf{v}_0))$. Clearly $\mathbf{v}_0\mathbf{v}_1\ldots\mathbf{v}_k$ is a trajectory that is compatible with π. The fact that $\tau \in TRJ$ if $\pi \in \mathsf{paths}_M$ follows from the definition of compatibility.

To prove the third part let $\tau = (q_0,\mathbf{v}_0)\ (q_1,\mathbf{v}_1)\ldots(q_k,\mathbf{v}_k) \in TRJ$. Again we proceed by induction on k. Suppose $k = 0$. Then $(q_{in}, \text{INIT}, \mathbf{P}_{\text{INIT}})$ is in paths_M which is compatible with τ. So suppose $k > 0$. Then by the induction hypothesis there exits $\pi' = \eta_0\eta_1\ldots\eta_{k-1}$ such that π' is compatible with $\tau' = (q_0,\mathbf{v}_0)(q_1,\mathbf{v}_1)\ldots(q_{k-1},\mathbf{v}_{k-1})$. Let $q_{k-1} \xrightarrow{g} q_k$. Since X_{k-1} is open there exists an open neighborhood $Y \subseteq X_{k-1}$ that contains $\mathbf{v}_{k-1}$. But then both $\Phi^{-1}_{q_{k-1}}$ and $\Phi^{-1}_{q_k}$ are continuous. Thus $\Phi_{q_{k-1},t}(Y)$ is open and $\Phi_{q_{k-1},t}(Y) \cap g$ should be open and non-empty (since g is open and $(q_k,\mathbf{v}_k)$ is part of the trajectory). Hence $Y' = \bigcup_{t\in(0,1)} \Phi_{q_k,1-t}(\Phi_{q_{k-1},t}(Y) \cap g)$ is a non-empty open set with a positive measure. This means there will be a state of the form $\eta_k = (\rho_k, X_k, \mathbf{P}_{X_k})$ in Y with $Y' \subseteq X_k$ and $\eta_{k-1} \overset{p}{\Rightarrow} \eta_k$ for some $p \in (0,1]$. Clearly $\pi = \pi'\eta_k \in \mathsf{paths}_M$ and is compatible with τ. Again the fact that $\pi \in \mathsf{paths}_M$ if $\tau \in TRJ$ follows from the definition of compatibility. □

Theorem 2 *$H \models \psi$ iff $M \models \psi$.*

PROOF Suppose H does not meet the specification ψ. Then there exists $\tau \in TRJ^{K+1}$ such that $\tau, 0 \not\models_H \psi$. By the third part of Lemma 1 there exists $\pi \in \mathsf{paths}_M^{K+1}$ which is compatible with τ. By the first part of Lemma 1 we then have $\pi \notin \mathit{models}_M(\psi)$ which leads to $Pr_{<1}(\psi)$.

Next suppose that $Pr_{<1}(\psi)$. Then there exists $\pi \in \mathsf{paths}^{K+1}$ such that $\pi, 0 \not\models_M \psi$. By the second part of Lemma 1 there exists $\tau \in TRJ^{K+1}$ which is compatible with π. By the first part of Lemma 1 this implies $\tau, 0 \not\models_H \psi$ and this in turn implies that H does not meet the specification ψ. □

6.4.2 *Quantitative atomic propositions*

The above results can be extended to handle atomic propositions of the form $\langle x_i < c \rangle$ and $\langle x_i > c \rangle$ where c is a rational constant. We partition $\Re^n$ into hypercubes according to the constants appearing in the given set of quantitative atomic propositions in AP_{qt}. We then blow up the state space of the Markov chain to record which hypercube the current values of the variables fall in. We restrict our attention to robust trajectories and show that every robust trajectory of H meets a BLTL specification iff its Markov chain approximation meets the same specification with probability 1. Informally a robust trajectory is one which has an open neighborhood of trajectories under a natural topology over the space of $K+1$-length trajectories. Under an associated measure the set of non-robust trajectories will have measure 0. The details can be found in the Appendix A.1.

6.5 STATISTICAL MODEL CHECKING OF HYBRID SYSTEMS

To verify whether the hybrid system H meets the specification ψ, we solve the equivalent problem whether $\Pr_{\geq 1}(\psi)$ on its associated Markov chain M. This model checking problem can be solved using approximate probabilistic model checking algorithms. One such computationally effective verification technique increasingly employed in verifying large, complex engineering and biological models is statistical model checking (SMC). Using either hypothesis testing approach or confidence interval estimation approach, SMC is based on sampling independent traces of a system until enough statistical evidence for the satisfaction or violation of the specification has been found.

Following a SMC based approach, we don't need to explicitly represent M, which may be intractable. We generate random realizations of the branches of M and pose a hypothesis test to decide whether $\Pr_{\geq 1}(\psi)$ based on these realizations.

6.5.1 *The SMC procedure*

To verify whether H meets the specification ψ, we solve the equivalent problem whether $\Pr_{\geq 1}(\psi)$ on M. However, as discussed in the Section 6.1, M cannot be constructed explicitly since both its structure and transition probabilities, defined in terms of the solutions to the ODEs, will not be available. Therefore we shall use randomly generated trajectories to sample the paths of M and formulate a sequential hypothesis test to decide with bounded error rate whether $\Pr_{\geq 1}(\psi)$ holds. Algorithm 1 describes our trajectory sampling procedure.

Algorithm 1 Trajectory simulation

Input: Hybrid automaton $H = (Q, q_{in}, \{F_q(\mathbf{x})\}_{q\in Q}, \mathcal{G}, \to, \text{INIT})$, maximum time step K.

Output: Trajectory τ

1: Sample $\mathbf{v}_0$ from INIT uniformly, set $q_0 := q_{in}$ and $\tau := (q_0, \mathbf{v}_0)$.
2: **for** $k := 1 \dots K$ **do**
3: Generate time points $T := \{t_1, \dots, t_J\}$ uniformly in $(0, 1)$.
4: Simulate $\mathbf{v}^j := \Phi_{q_{k-1}}(t_j, \mathbf{v}_{k-1})$, for $j \in \{1, \dots, J\}$
5: Let $\widehat{\mathbb{T}}_j := \{t \in T : \mathbf{v}^j \in g_j\}$ be the time points where g_j is enabled.
6: Pick g_ℓ randomly according to probabilities $\{p_j := |\widehat{\mathbb{T}}_j| / \sum_{i=1}^m |\widehat{\mathbb{T}}_i|\}$.
7: Pick t_ℓ uniformly at random from $\widehat{\mathbb{T}}_\ell$.
8: Simulate $\mathbf{v}' := \Phi_{q'}(1 - t_\ell, \mathbf{v}^\ell)$, where q' is the target of g_ℓ.
9: Set $q_k := q'$, $\mathbf{v}_k := \mathbf{v}'$, and extend $\tau := (q_0, \mathbf{v}_0) \dots (q_k, \mathbf{v}_k)$.
10: **end for**
11: **return** τ

Clearly Algorithm 1 generates a trajectory in TRJ^{K+1}. We now relate these trajectories to paths in M.

The initial value $\mathbf{v}_0$ is sampled uniformly on INIT, and we start in mode q_{in}, consistent with the initial state (q_{in},INIT,$\mathbf{P}_{\text{INIT}}$) of M. Inductively, suppose $\eta = (\rho, X, \mathbf{P}_X)$ is a state of M with ρ ending in q. Suppose $\eta \overset{p_j}{\Rightarrow} \eta_j$ is a transition in M such that $\eta_j = (\rho q_j, X_j, \mathbf{P}_{X_j})$.

Proposition 2 *Suppose, we obtain a sample $\boldsymbol{v} \sim \boldsymbol{P}_X$. The probability of choosing guard g_j whose target mode is q_j in Algorithm 1 tends to p_j as $J \to \infty$.*

PROOF According to Algorithm 1, the probability of picking guard g_j for a trajectory starting at $\mathbf{v} \in X$ is defined as $|\widehat{\mathbb{T}}_j| / \sum_{i=1}^{m} |\widehat{\mathbb{T}}_i|$, which, by the law of large numbers tends to

$$p_j(\mathbf{v}) := \frac{\mu(\mathbb{T}_j(\mathbf{v}))}{\sum_{i=1}^{m} \mu(\mathbb{T}_i(\mathbf{v}))} \tag{10}$$

as J tends to ∞.

Now if $\mathbf{v}$ is randomly sampled according to $\mathbf{P}_X$, then the probability of picking guard j can be expressed as the expected value of $p_j(\mathbf{v})$ under $\mathbf{v} \sim \mathbf{P}_X$ as

$$\mathbb{E}_{\mathbf{v}\sim\mathbf{P}_X}[p_j(\mathbf{v})] = \int_{\mathbf{v}\in X} p_j(\mathbf{v}) d\mathbf{P}_X = \int_{\mathbf{v}\in X} \frac{\mu(\mathbb{T}_j(\mathbf{v}))}{\sum_{i=1}^{m} \mu(\mathbb{T}_i(\mathbf{v}))} d\mathbf{P}_X, \tag{11}$$

which by (8) is equal to p_j, the corresponding transition probability in the Markov chain. □

Similarly, picking the transition time t from $\widehat{\mathbb{T}}_j$ will approximate sampling $t \sim \mathbf{P}_{\mathbb{T}_j(\mathbf{v})}$, for sufficiently high J. Next, assume that we have picked $q \xrightarrow{g_j} q_j$ as the transition to take. We sample $t \sim \mathbf{P}_{\mathbb{T}_j(\mathbf{v})}$, and obtain $\mathbf{v}'$ by numerical simulation via:

$$\mathbf{v}' = \Phi_{q_j}(1 - t, \Phi_q(t, \mathbf{v})). \tag{12}$$

Proposition 3 *v' is distributed according to $\boldsymbol{P}_{X_j}$.*

PROOF Clearly it suffices to show that for a measurable subset $Y \subseteq X_j$, $\Pr(\mathbf{v}' \in Y) = \mathbf{P}_{X_j}(Y)$. We start with

$$\Pr(\mathbf{v}' \in Y \mid \mathbf{v}) = \int_{t\in\mathbb{T}_j(\mathbf{v})} \mathbf{1}_{(\Phi_{q_j}(1-t,\Phi_q(t,\mathbf{v}))\cap Y)} d\mathbf{P}_{\mathbb{T}_j(\mathbf{v})}.$$

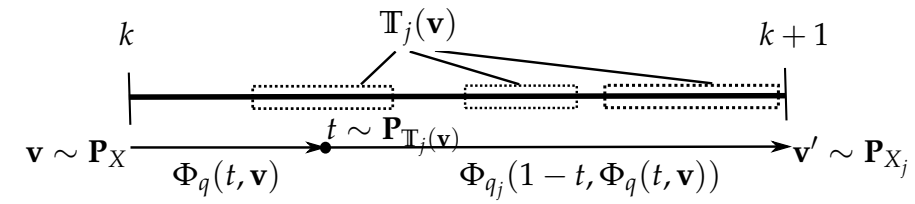


Figure 19: Propagating a single value $\mathbf{v} \in X$ to $\mathbf{v}' \in X_j$ when taking the transition $q \to q_j$ at time $t \in \mathbb{T}_j(\mathbf{v})$.

Integrating now over all possible choices of $\mathbf{v}$ with respect to $\mathbf{P}_X$ we have

$$\Pr(\mathbf{v}' \in Y) = \int_{\mathbf{v} \in X} \Pr(\mathbf{v}' \in Y \mid \mathbf{v}) d\mathbf{P}_X.$$

From (7) it follows that $\Pr(\mathbf{v}' \in Y) = \mathbf{P}_{X_j}(Y)$ with $\mathbf{v} \sim \mathbf{P}_X$ and $t \sim \mathbf{P}_{\mathbb{T}_j(\mathbf{v})}$.

□

Consequently, the trajectory being generated will now be in mode q_j with $\mathbf{v}' \in X_j$ and $\mathbf{v}'$ distributed according to $\mathbf{P}_{X_j}$, compatible with the state $\eta_j = (\rho q_j, X_j, \mathbf{P}_{X_j})$ of M. Inductively it is hence guaranteed that each subsequent iteration of Algorithm 1 will produce values compatible with a path of M.

Whether the generated trajectory of length $K + 1$ (and hence the corresponding path of M) is a model of ψ can be determined using a standard BLTL model checker [101]. In fact this can be done on the fly which will often avoid generating the whole trajectory. Based on this, we can test whether $\Pr_{\geq 1}(\psi)$ on M by testing the following alternative pair of hypotheses: $H_0 : \Pr_{\geq 1}(\psi)$ and $H_1 : \Pr_{< 1-\delta}(\psi)$, where $0 < \delta < 1$ is a parameter chosen by the user marking the interval $[1 - \delta, 1)$ as an indifference region in which accepting either hypothesis is fine. In our setting, whenever we encounter a sample (i.e. a randomly generated trajectory) that does not satisfy ψ, we can reject H_0 and accept H_1. Thus we only have to deal with false positives (when H_0 is accepted while H_1 happens to be true).

This leads to Algorithm 2 that repeatedly generates a random trajectory (using Algorithm 1), and decides after a finite number of tries between H_0 and H_1. For doing so we also fix a user-defined false positive rate α.

Algorithm 2 Sequential hypothesis test

Input: Markov chain M, BLTL property ψ, indifference parameter δ, false positive bound α.

Output: H_0 or H_1.

1: Set $N := \lceil \log \alpha / \log(1-\delta) \rceil$
2: **for** $i := 1 \ldots N$ **do**
3: Generate a random trajectory τ using Algorithm 1
4: **if** $\tau, 0 \models^H \psi$ **then** Continue
5: **else return** H_1
6: **end for**
7: **return** H_0

The accuracy of Algorithm 2 is captured by the next result.

Theorem 3 *1. The probability of choosing H_1 when H_0 is true (false negative) is 0.*

2. Further, suppose $N \geq \log \alpha / \log(1-\delta)$. Then the probability of choosing H_0 when H_1 is true (false positive) is no more than α.

PROOF As observed earlier the first part is obvious. To prove the second part, if H_1 is true, then we know that $\Pr_{<1-\delta}(\psi)$. The probability of N sampled trajectories all satisfying ψ (and thus returning H_0, a false positive) is at most $(1-\delta)^N$. Therefore we have $\alpha \leq (1-\delta)^N$, leading to $N \geq \log \alpha / \log(1-\delta)$. □

Hence we use $N := \lceil \log \alpha / \log(1-\delta) \rceil$ to set the sample size. For example for $\delta = 0.01$ and $\alpha = 0.01$ we get $N = 459$ while for $\delta = 0.001$ and $\alpha = 0.01$ we get $N = 4603$.

6.6 THE GPU IMPLEMENTATION

One general approach would be to map the computation of each trajectory of the hybrid system to a GPU thread. In this way a large number of such threads can be executed in parallel on GPUs. But conventional methods of realizing parallelism in computation of independent simulations have their fair share of challenges. As observed before in previous chapters, the GPU programming model has poor tolerance for control flow divergence and crude memory coalescence. The mode switchings in the evolution of a trajectory could translate to a large number of control-flow divergences and hence result in serious performance degradation. We developed a method based on the heterogeneous code generation scheme described in Chapter 3 which overcomes these challenges by generating a dedicated pool of compute threads that coordinate to compute a single trajectory.

We generate a number of blocks of trajectories that execute the sampling algorithm —Algorithm 1— in parallel. These blocks are distributed across a number of GPU cores. Starting from an initial state at $t = 0$, for each time interval Δt, the new value of a variable x and the mode q is determined by applying numerical integration using the current values of the variables and the set of ODEs corresponding to the current mode. This state information of the variables and its current operating mode of a trajectory is maintained in the shared memory. In each Δt, for the J sampled time points at which the guards are to be evaluated, each trajectory computes the set of guards that are enabled. This, along

with the current operating mode information, is stored in the fast on-chip memory. Each trajectory then picks g_ℓ according to the probability $\{p_j := |\widehat{\mathbb{T}}_j| / \sum_{i=1}^{m} |\widehat{\mathbb{T}}_i|\}$ and the time point t is picked uniformly at random from the set of timepoints at which g_ℓ is enabled. Based on the chosen g_ℓ and t, each trajectory then updates its state information of the variables x at t and the new operating mode q to the shared store, so that the simulation continues based on the new mode.

Further, to get around the stringent memory restriction imposed by the GPU kernels, one often has to manage latency hiding to move data between the small on-chip scratch pad memory and the slow global memory. Hence to achieve optimal performance, we spawn a dedicated set of memory access threads that carry out the high-latency memory transfers while the trajectory simulation continues in parallel.

6.7 CASE STUDIES

We first evaluated our method on a a model of the electrical dynamics of the cardiac cell [111] and a model of circadian rhythm network [112]. The Δ time step parameter for the cardiac cell model and the circadian rhythm model were both set to 0.1.The parameters used for the statistical model checking were $\delta = 0.01$ and $\alpha = 0.01$. We have implemented our method using MATLAB and C++. The experiments were carried out on a PC with a 3.4GHz Intel Core i7 processor with 8GB RAM. The GPU implementation was based on CUDA 5.0 runtime and the target GPU was NVidia Tesla K20m clocked at 706 MHz with 4.8 GB global memory.

We note that when checking quantitative properties, the trajectories that hit corner points such as $u = 1.4$ for the cardiac cell model will be non-robust and hence can be ignored.

6.7.1 *Cardiac cell model*

Heart rhythm depends on the organized opening and closing of gates–called ion channels–on the cell membrane, which govern the electrical activity of cardiac cells. Disordered electric wave propagation in heart muscle can cause cardiac abnormalities such as *tachycardia* and *fibrillation*. The dynamics of the electrical activity of a single human ventricular cell has been modelled as a hybrid automaton [3, 111] shown in Figure 20. The model contains 4 state variables and 26 parameters. Ventricular cells consist of three subtypes, namely epicardial, endocardial, and midmyocardial cells, which possess different dynamical characteristics. The cell-type-specific parameters of the model are summarized in Table 9.

Parameter	EPI	ENDO	MID	Parameter	EPI	ENDO	MID
θ_o	0.006	0.006	0.006	τ_{v1}^-	60	75	80
θ_w	0.13	0.13	0.13	τ_{v2}^-	1150	10	1.4506
θ_v	0.3	0.3	0.3	τ_{w1}^-	60	6	70
u_w^-	0.03	0.016	0.016	τ_{w2}^-	15	140	8
u_{so}	0.65	0.65	0.6	τ_{o1}	400	470	410
u_s	0.9087	0.9087	0.9087	τ_{o2}	6	6	7
u_u	1.55	1.56	1.61	τ_{so1}	30.0181	40	91
w_∞^*	0.94	0.78	0.5	τ_{so2}	0.9957	1.2	0.8
k_w^-	65	200	200	τ_{s1}	2.7342	2.7342	2.7342
k_{so}	2.0458	2	2.1	τ_{s2}	16	2	4
k_s	2.994	2.994	2.994	τ_{fi}	0.11	0.1	0.078
τ_v^+	1.4506	1.4506	1.4506	τ_{si}	1.8875	2.9013	3.3849
τ_w^+	200	280	280	$\tau_{w\infty}$	0.07	0.0273	0.01

Table 9: Parameter values of the cardiac model for epicardial (EPI), endocardial (ENDO), and midmyocardial (MID) cells under healthy condition

An action potential (AP) is a change in the cell's transmembrane potential u, as a response to an external stimulus (current) ϵ. The flow of

total currents is controlled by a fast channel gate v and two slow gates w and s.

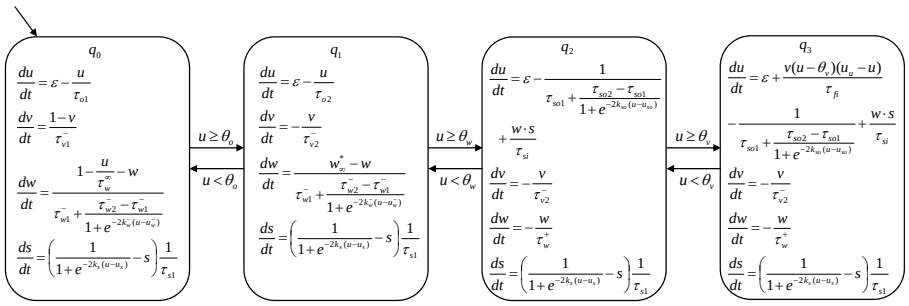


Figure 20: The hybrid automaton model for the cardiac cell system [3].

In mode q_0, the "Resting mode", the cell is waiting for stimulation. We assume an external stimulus ϵ equal to 1 mV lasting for 1 millisecond. The stimulation causes u to increase which may trigger a mode transition to mode q_1. In mode q_1, gate v starts closing and the decay rate of u changes. The system will jump to mode q_2 if $u > \theta_w$. In mode q_2, gate w is also closing. When $u > \theta_v$, mode q_3 can be reached, which means a successful "AP initiation". In mode q_3, u reaches its peak due to the fast opening of a sodium channel. The cardiac muscle then contracts and u starts decreasing.

Property C1 It is known that the cardiac cell can lose its excitability, which will lead to disorders such as ventricular tachycardia and fibrillation. We formulated the property for responding to stimulus by leaving the resting mode:

$$\mathbf{F}^{\leq 500}(\neg[\text{Resting mode}]).$$

The property was verified to be *true* for all three cell types under the healthy condition. However, under a disease condition (for example $\tau_{o1} = 0.004$ or $\tau_{o2} = 0.1$ [113]) the property was verified to be *false* no matter what stimulation value of ϵ was used. Consequently, a region of such unexcitable cells blocks the impulse conduction and can lead to

cardiac disorders such as fibrillation. This is consistent with experimental results reported in [114].

Property C2 After successfully generating an AP (that is, reaching the "AP mode", q_3), the cardiac cell should return to a low transmembrane potential and wait in "Resting mode" for the next stimulation. The corresponding formula is

$$\mathbf{F}^{\leq 500}([\text{AP mode}]) \wedge \mathbf{F}^{\leq 500}(\mathbf{G}^{\leq 100}([\text{Resting mode}])).$$

The above query was verified to be *true* for all three cell types under the healthy condition and transient stimulation. However, if we change the stimulation profile from transient to sustained, i.e. assuming ϵ lasts for 500 milliseconds, the property was verified to be *false*–the cell doesn't return to and settle at a low transmembrane potential resting state. In ventricular tissue the stimulus ϵ can be delivered from neighboring cells [111]. Thus, our results suggest that the transient activation of a single cardiac cell depends on the stimulation profile of its neighboring cells.

Property C3 It has been reported that epicardial, endocardial, and midmyocardial cells have different AP morphologies [4, 5]. In particular, a crucial "spike-and-dome" (i.e. a sharp peak followed by a blunt peak) AP morphology can only be observed in epicardial cells but not endocardial and midmyocardial cells (Figure 21).

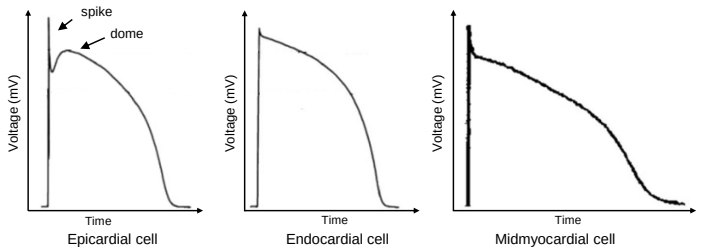


Figure 21: The AP morphologies of epicardial [4], endocardial [4] and midmyocardial [5] cells.

We formulated the property for a spike-and-dome AP morphology as a quantitative property,

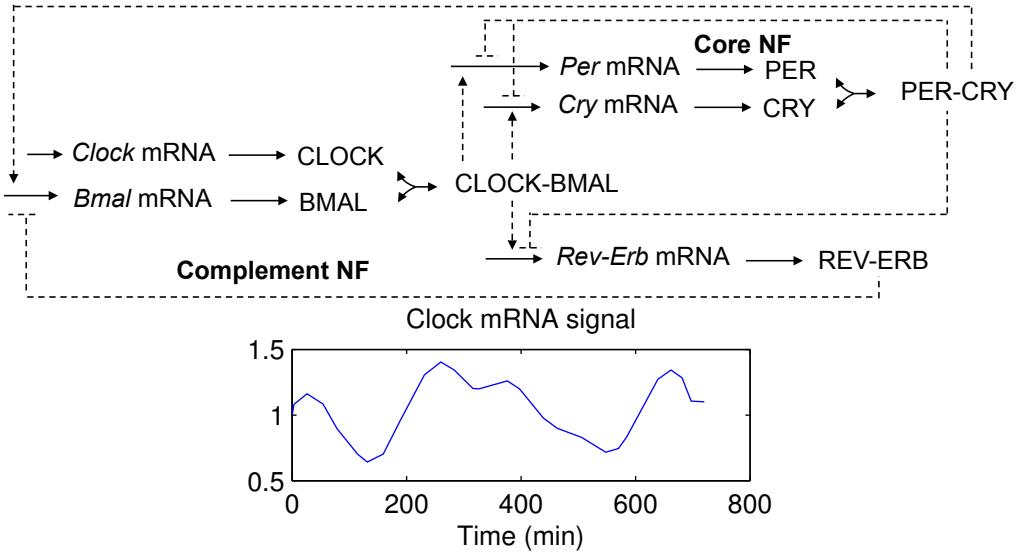
$$\mathbf{F}^{\leq 500}(\mathbf{G}^{\leq 1}([1.4 \leq u]) \wedge \mathbf{F}^{\leq 500}([0.8 \leq u] \wedge [u \leq 1.1] \wedge \mathbf{F}^{\leq 500}(\mathbf{G}^{\leq 50}([1.1 \leq u])))).$$

The property was verified to be *true* for epicardial cells and *false* for endocardial and midmyocardial cells under the healthy condition and transient stimulation. Among 26 model parameters, 20 of them have different values over different cell types. We then perturbed each epicardial parameter and checked if the above property still holds. Our results show that τ_{s2} is key to the AP morphology (i.e. the spike-and-dome AP morphology disappears when $\tau_{s2} = 2$), which highlights the importance of s gate to epicardial cells. This is consistent with [115] in that the model proposed in [116] (which does not include s gate) is unable to capture the dynamics of epicardial cells.

6.7.2 *Circadian rhythm model*

Mammalian cells follow a circadian rhythm with a 24h period, which is generated and governed by a highly coupled transcription-translation network. The model diagram and the corresponding hybrid system dynamics proposed in [112, 117] is described below.

The equations governing the dynamics of the circadian clock model are given in Figure 22. The equations contain rate constants which are denoted k_1 to k_28 and are set according to [117]. The combination of "mode indicator" binary variables θ_{CB} to θ_{RE}, θ_{PC1}, θ_{PC2} and θ_{PC3} define the mode of the dynamics, and each mode is defined by a unique value combination of the mode indicators. These value combinations are listed in Table 10. The guards associated with a source and target mode are constructed as follows. Each mode indicator corresponds to a guard component which is a threshold on a state variable. For instance, θ_{RE} has the corresponding guard component [REV-ERB]$<$ 1.1. The guard



$$
\begin{aligned}
\mathrm{d}/\mathrm{d}t[\text{Per}] &= -k_1 \cdot [\text{Per}] + k_{13} \cdot \theta_{PC2} \cdot \theta_{CB} + k_{14} \\
\mathrm{d}/\mathrm{d}t[\text{PER}] &= -k_2 \cdot [\text{PER}] + k_{15} \cdot [\text{Per}] - k_{16} \cdot [\text{PER}] \cdot [\text{CRY}] \\
\mathrm{d}/\mathrm{d}t[\text{Cry}] &= -k_3 \cdot [\text{Cry}] + k_{17} \cdot \theta_{PC2} \cdot \theta_{CB} + k_{18} \\
\mathrm{d}/\mathrm{d}t[\text{CRY}] &= -k_4 \cdot [\text{CRY}] + k_{19} \cdot [\text{Cry}] - k_{16} \cdot [\text{PER}] \cdot [\text{CRY}] \\
\mathrm{d}/\mathrm{d}t[\text{PER-CRY}] &= -k_5 \cdot [\text{PER-CRY}] + k_{16} \cdot [\text{PER}] \cdot [\text{CRY}] \\
\mathrm{d}/\mathrm{d}t[\text{Rev-Erb}] &= -k_6 \cdot [\text{Rev-Erb}] + k_{20} \cdot \theta_{PC1} \cdot \theta_{CB} + k_{21} \\
\mathrm{d}/\mathrm{d}t[\text{REV-ERB}] &= -k_7 \cdot [\text{REV-ERB}] + k_{22} \cdot [\text{Rev-Erb}] \\
\mathrm{d}/\mathrm{d}t[\text{CLOCK}] &= -k_9 \cdot [\text{CLOCK}] + k_{24} \cdot [\text{Clock}] - k_{25} \cdot [\text{CLOCK}] \cdot [\text{BMAL}] \\
\mathrm{d}/\mathrm{d}t[\text{Bmal}] &= -k_{10} \cdot [\text{Bmal}] + k_{26} \cdot \theta_{PC3} \cdot \theta_{RE} + k_{27} \\
\mathrm{d}/\mathrm{d}t[\text{BMAL}] &= -k_{11} \cdot [\text{BMAL}] + k_{28} \cdot [\text{Bmal}] - k_{25} \cdot [\text{CLOCK}] \cdot [\text{BMAL}] \\
\mathrm{d}/\mathrm{d}t[\text{CLOCK-BMAL}] &= -k_{12} \cdot [\text{CLOCK-BMAL}] + k_{25} \cdot [\text{CLOCK}] \cdot [\text{BMAL}]
\end{aligned}
$$

Figure 22: The model diagram, the Clock mRNA signal and the equations governing the circadian clock model.

to a target mode is enabled if all the mode indicators that are on in the mode are enabled according to their respective guard components. Finally, a transition between a source and a target mode only exists if there is only one difference in the combination of mode indicators. For instance, there is a transition from mode 1 to mode 2 but not from mode 1 to mode 9. The dynamics of the *Clock* mRNA is governed externally.

The system comprises 16 modes, each of which contains 12 state variables and 29 parameters. Each mode corresponds to a particular combination of ON or OFF transcriptional states of genes *Per*, *Cry*, *Rev-Erb*, *Clock*, and *Bmal*. The switches between modes are guarded by the thresh-

Mode indicator	Guard component
θ_{RE}	[REV-ERB]<1.1
θ_{CB}	[CLOCK-BMAL]>1.0
θ_{PC1}	[PER-CRY]<1.4
θ_{PC2}	$1.4<$[PER-CRY]<1.5
θ_{PC3}	$2.2<$[PER-CRY]

Mode	1	2	3	4
$(\theta_{PC1},\theta_{PC2},\theta_{PC3},\theta_{RE},\theta_{CB})$	(1,1,0,1,0)	(1,1,0,1,1)	(1,1,0,0,0)	(1,1,0,0,1)
Mode	5	6	7	8
$(\theta_{PC1},\theta_{PC2},\theta_{PC3},\theta_{RE},\theta_{CB})$	(0,1,0,1,0)	(0,1,0,1,1)	(0,1,0,0,0)	(0,1,0,0,1)
Mode	9	10	11	12
$(\theta_{PC1},\theta_{PC2},\theta_{PC3},\theta_{RE},\theta_{CB})$	(0,0,0,1,0)	(0,0,0,1,1)	(0,0,0,0,0)	(0,0,0,0,1)
Mode	13	14	15	16
$(\theta_{PC1},\theta_{PC2},\theta_{PC3},\theta_{RE},\theta_{CB})$	(0,0,1,1,0)	(0,0,1,1,1)	(0,0,1,0,0)	(0,0,1,0,1)

Table 10: The 5 *mode indicator* variables and their associated guard components (top). The 16 modes of the circadian clock model with the corresponding combination of binary mode indicator variables (bottom).

old levels of protein complexes PER-CRY, CLOCK-BMAL and REV-REB. The mRNA levels of *Per* and *Cry* are known to be oscillating due to the negative feedback loops in the network. Specifically, there are two major negative feedback (NF) loops: (i) the core NF formed by PER-CRY, CLOCK-BMAL, PER, and CRY and (ii) a complement NF formed by REV-ERB, BMAL, and CLOCK-BMAL. The time constants appearing in the properties are in minute units.

Property R1 Similar to *Per* and *Cry*, the expression level of *Bmal* gene is also oscillating [118]. We formulated this property as

$$\mathbf{F}^{\leq 500}([1.5 \leq Bmal] \wedge \mathbf{F}^{\leq 500}([Bmal \leq 0.8] \wedge \mathbf{F}^{\leq 500}([1.5 \leq Bmal] \wedge \mathbf{F}^{\leq 500}([Bmal \leq 0.8] \wedge \mathbf{F}^{\leq 500}([1.5 \leq Bmal])))))$$

The property was verified to be *true* under the wild type condition. It was verified to be *false* under *Cry* mutant condition but *true* in the *Rev-Erb* mutant condition, which is consistent with the experimental data in [118, 119]. This suggests that the oscillatory behaviour of *Bmal* mRNA

is induced by the core negative feedback mediated by PER-CRY, instead of the complement negative feedback mediated by REV-ERB.

Property R2 It has been observed that the peaks of *Bmal* mRNA are always located between two successive *Per* or *Cry* mRNA peaks [119]. The corresponding formula is

$$\mathbf{F}^{\leq 500}([Bmal \leq 0.8] \wedge [2.0 \leq Per] \wedge [2.0 \leq Cry] \wedge \mathbf{F}^{\leq 500}([1.5 \leq Bmal] \wedge [Per \leq 0.8] \wedge [Cry \leq 0.8] \wedge \mathbf{F}^{\leq 500}([Bmal \leq 0.8] \wedge [2.0 \leq Per] \wedge [2.0 \leq Cry] \wedge \mathbf{F}^{\leq 500}([1.5 \leq Bmal] \wedge [Per \leq 0.8] \wedge [Cry \leq 0.8]))))$$

The above query was verified to be *true* under wild type condition. If we remove the dependence between *Bmal* transcription and PER-CRY concentration, the property R2 was verified to be *false*, while the property R1 was verified to *true* (i.e. oscillating). Thus, our results suggest that the complement negative feedback mediated by REV-ERB is responsible for maintaining the oscillatory behaviour of *Bmal* mRNA level while PER-CRY plays a role in delaying the *Bmal* expression responses.

6.8 PERFORMANCE

Table 11 is a summary of the performance of the verification of all properties for the three models for the three hybrid systems along with the number of samples taken to complete the verification.

In our experiments, we used $J = 10$ as the number of intermediate time steps for choosing mode transitions. We investigated whether this choice is sufficient for accurate simulation. We simulated 1000 independent realizations of the cardiac cell system with $J = 10$ and $J = 100$, and compared the distributions of the modes that the system is in at a series of discrete time points. The Kolmogorov-Smirnov statistical test did not reject the hypothesis that the two distributions are the same (at confidence level 95%). This indicates that using $J = 10$ is adequate.

For the GPU implementation, we used δ = 0.001 and α=0.01. The average runtime and the speed-up achieved for the properties which were verified to be *true* are summarized in Table 12 for both the case studies. For properties which were verified to be *false*, the hypothesis testing algorithm, Algorithm 2 terminates after sampling 1 trajectory. Our parallelization scheme for the trajectory sampling procedure achieves approximately 6× speed-up for both the case studies. It is noted that our parallelization scheme can be further enhanced for handling larger hybrid systems in future.

Property	Condition	Decision	# samples
C1	Epicardial, Healthy	True	459
C1	Endocardial, Healthy	True	459
C1	Midmyocardial, Healthy	True	459
C1	Epicardial, Diseased	False	1
C1	Endocardial, Diseased	False	1
C1	Midmyocardial, Diseased	False	1
C2	Epicardial, Transient	True	459
C2	Endocardial, Transient	True	459
C2	Midmyocardial, Transient	True	459
C2	Epicardial, Sustained	False	1
C2	Endocardial, Sustained	False	1
C2	Midmyocardial, Sustained	False	1
C3	Epicardial, $\tau_{s2} = 16$	True	459
C3	Epicardial, $\tau_{s2} = 2$	False	1
C3	Endocardial	False	1
C3	Midmyocardial	False	1
R1	Wild type	True	459
R1	Cry mutant	False	1
R1	Rev-Erb mutant	True	459
R2	Wild type	True	459
R2	Without PER-CRY dependence	False	1
R1	Without PER-CRY dependence	True	459

Table 11: Results summary of SMC for hybrid systems

Model	Average CPU runtime (s)	Average GPU runtime (s)	Speed-up
Cardiac cell (C1, C2, C3)	846	144	5.9 ×
Circadian clock (R1, R2)	253	41.5	6.1 ×

Table 12: Peformance of the GPU implementation for properties which were verified to be *true*

6.9 SUMMARY

We have presented an approximate probabilistic verification method for analyzing the dynamics of a hybrid system H in terms of a Markov chain M. For bounded time properties, we have shown a strong correspondence between the behaviours of H and M. We have also extended this result to handle quantitative atomic propositions in Appendix A.1 and shown a similar correspondence result for the sub-dynamics consisting of robust trajectories. Thus the intractable verification problem for H can be solved approximately using its Markov chain approximation. Accordingly, we have devised a statistical model checking procedure to verify that M almost certainly meets a BLTL specification and then applied this procedure to two examples to demonstrate the applicability of our approximation scheme. Our GPU accelerated parallel implementation of the trajectory sampling procedure achieves significant speed-up when compared with a CPU implementation.

7

CONCLUSION

We briefly summarize the key contributions of the thesis and look at possible directions for future work. The focus of our work has been on approximation methods for the complex dynamics of biopathways. We have studied both single system of ODEs and the much more involved setting of hybrid systems. Our approximations are probabilistic in nature and consequently they are also accompanied by a statistical model checking procedure. This then provides the basis for carrying out analysis tasks such as parameter estimation and sensitivity analysis. A second focus has been on GPU based implementations in order to mitigate the very high computational demands of the various analysis tasks.

In Chapter 3, we first recalled from [22] how the dynamics of a system of ODEs can be approximated as a dynamic Bayesian network. This DBN approximation consists of pre-computing a representative sample of trajectories induced by the system of ODEs. We then developed a GPU based parallelization scheme that exploits the fine-grained parallelism in the generation of a single trajectory by using multiple dedicated compute threads. Further by employing latency-hiding and load-balancing techniques, we mapped the entire DBN approximation scheme to the GPU platform. We showed our method achieved significant performance improvement.

Next, in Chapter 4, we recalled a statistical model checking framework for analysis of a single system of ODEs, developed in [24]. By attaching a probability distribution to the set of initial states of the ODEs, we first approximated the ODEs system as a discrete-time Markov chain.

One can then use an SMC procedure to verify whether the system satisfies dynamical properties expressed in BLTL. The key point here is this can be achieved without explicitly constructing the discrete-time Markov chain which is in any case an intractable problem. One needs to just sample from the initial states and then generate a trajectory using numerical simulation. One main advantage of SMC based analysis, as against the DBN construction based approach is the (or lack thereof) model construction cost. Also, the complexity of the model checking algorithm is independent of the size of the system. As a result, the required number of samples only depends on the probabilistic distribution and the error bounds associated with the statistical test. Yet SMC requires a large number of simulations and this brings us to the construction of a parallelized statistical model checking framework.

For porting the SMC based analysis technique to GPUs, in Chapter 5, we introduced an automaton-based BLTL path checking algorithm. The online procedure we constructed was efficient in that the algorithm examines a trajectory as it is being generated. Instead of generating the entire trajectory and then checking whether it satisfies a given property, it incrementally simulates the ODEs model and checks whether the current trajectory satisfies the BLTL formula. The gains due to the reduced memory usage were significant and it reflected in the performance of our parallelized parameter estimation procedure.

We demonstrated the applicability of the two approximation techniques with the help of biopathway models taken from the Biomodels database [88]. The key feature in the GPU implementations in Chapter 3 and in Chapter 5 is the novel way of handling the GPU threads for generation of a single trajectory of the ODEs system. The fine-grained parallelism —inherent in the fact that the next state value of each variable can be computed independently by the current state value of the

other variables— renders itself to an efficient GPU implementation. To this end, a heterogeneous pool of multiple threads were instantiated to handle the simulation of a single trajectory. As a result, our method achieves higher GPU utilization due to the large number of parallel threads spawned. Also because the thread pool shares the intermediate data of a trajectory, our method gains from huge reduction in memory usage.

We then built on these parallelized approximation schemes to analyze hybrid systems. In Chapter 6, we developed a probabilistic approximation of the dynamics of a hybrid system as a Markov chain. Based on the correspondence we established between the behaviour of the hybrid automaton and the Markov chain using BLTL, we developed a statistical model checking procedure to verify dynamical properties by sampling trajectories of the hybrid system. Our approximation scheme was applied to verify properties of a circadian rhythm model and a cardiac cell model. Consequently, this approximation technique was then implemented on a GPU and our parallelization method achieved significant speed-up.

We note that with the advent of affordable GPUs in solving computationally intensive problems, analysis tasks which involve drawing a prohibitively large number of numerical simulations can benefit greatly from our parallelization techniques.

7.1 FUTURE WORK

It would be interesting to augment the current probabilistic analysis framework for a single system of ODEs with additional tools which can help in the synthesis of non-trivial temporal properties. This can be achieved by learning the properties from the simulation profiles of the

dynamical system. We have some preliminary results and are focusing along this direction to automatically mine requirements of dynamical systems.

As an extension of the probabilistic approximation method for the hybrid systems, one could consider more sophisticated stochastic assumptions regarding the time points and value states at which the mode transitions take place. These assumptions will however have to be justified and motivated by the modeling problem at hand, especially in systems biology applications. Yet another valuable extension will be to study a network of hybrid systems. This will enable us to model the cross talk, feed-forward and feed-back loops involving multiple signaling pathways. A further discretization of the continuous component of the hybrid system could also be coupled with the proposed approach to reduce the complexity and increase the robustness of biological models [120].

On the GPU front, currently we deal with a maximum of 12 state variables for the circadian clock model. However, for handling larger hybrid systems in future, one would have to overcome the stringent memory restrictions imposed by the GPU hardware for models with many state variables. To enhance the usability of our approach, we are currently working on developing sophisticated load balancing techniques in this regard.

When constructing dynamical models to explain experimental observations, one often ends up with a population of models with different structures corresponding to different hypotheses about the underlying system. With sufficient GPU units available, one can evaluate the quality of a large number of these models in parallel using our method. One can also explore the parameter landscape to identify regions most likely to induce the desired pathway responses to chosen stimuli. Our

future work will involve exploring such issues in the context of model comparison.

We are exploring the applicability of our approximation techniques to partial differential equations (PDEs) based systems. Coupled with our heterogeneous code generation scheme for GPUs, this would open up the parallelized techniques for analysis of a rich class of systems in fields like fluid dynamics. Another appealing direction of future research would be to explore parallelized analysis schemes using many-core processors like Intel Xeon Phi coprocessors.

We believe that our approximation and parallelization techniques open the door for studying large dynamical systems in a scalable and cost-effective manner.

BIBLIOGRAPHY

[1] Bree B Aldridge, John M Burke, Douglas A Lauffenburger, and Peter K Sorger. Physicochemical modelling of cell signalling pathways. *Nature cell biology*, 8(11):1195–1203, 2006.

[2] Kevin S Brown, Colin C Hill, Guillermo A Calero, Christopher R Myers, Kelvin H Lee, James P Sethna, and Richard A Cerione. The statistical mechanics of complex signaling networks: nerve growth factor signaling. *Physical biology*, 1(3):184, 2004.

[3] Radu Grosu, Gregory Batt, Flavio H. Fenton, James Gilmm, Colas Le Guernic, Scott A. Smolka, and Ezio Bartocci. From cardiac cells to genetic regulatory networks. In *CAV'11*, pages 396–411, 2011.

[4] M. Nabauer, D. J. Beuckelmann, P. Uberfuhr, and G. Steinbeck. Regional differences in current density and rate-dependent properties of the transient outward current in subepicardial and subendocardial myocytes of human left ventricle. *Circulation*, 93:169–177, 1996.

[5] E. Drouin, F. Charpentier, C. Gauthier, K. Laurent, and H. Le Marec. Electrophysiologic characteristics of cells spanning the left ventricular wall of human heart: evidence for presence of m cells. *J Am Coll Cardiol*, 26:185–192, 1995.

[6] H. Kitano. Systems biology: a brief overview. *Science*, 295(5560):1662–1664, 2002.

[7] Daniel T Gillespie et al. Exact stochastic simulation of coupled chemical reactions. *J. phys. Chem*, 81(25):2340–2361, 1977.

[8] Harley H McAdams and Adam Arkin. Stochastic mechanisms in gene expression. *Proceedings of the National Academy of Sciences*, 94(3):814–819, 1997.

[9] Michael A Gibson and Jehoshua Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The journal of physical chemistry A*, 104(9):1876–1889, 2000.

[10] Daniel T Gillespie. Approximate accelerated stochastic simulation of chemically reacting systems. *The Journal of Chemical Physics*, 115(4):1716–1733, 2001.

[11] Haluk Resat, H Steven Wiley, and David A Dixon. Probability-weighted dynamic monte carlo method for reaction kinetics simulations. *The Journal of Physical Chemistry B*, 105(44):11026–11034, 2001.

[12] Michael B Elowitz, Arnold J Levine, Eric D Siggia, and Peter S Swain. Stochastic gene expression in a single cell. *Science*, 297(5584):1183–1186, 2002.

[13] Yang Cao, Hong Li, and Linda Petzold. Efficient formulation of the stochastic simulation algorithm for chemically reacting systems. *The journal of chemical physics*, 121(9):4059–4067, 2004.

[14] Leon Glass and Stuart A Kauffman. Co-operative components, spatial localization and oscillatory cellular dynamics. *Journal of theoretical biology*, 34(2):219–237, 1972.

[15] Leon Glass and Stuart A Kauffman. The logical analysis of continuous, non-linear biochemical control networks. *Journal of theoretical Biology*, 39(1):103–129, 1973.

[16] Hidde De Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*, 9(1):67–103, 2002.

[17] Grégory Batt, Calin Belta, and Ron Weiss. Temporal logic analysis of gene networks under parameter uncertainty. *IEEE Transactions on Automatic Control*, 53(Special Issue):215–229, 2008.

[18] Herbert M Sauro. *Enzyme kinetics for systems biology*. Future Skill Software, 2011.

[19] Hiroaki Kitano. Computational systems biology. *Nature*, 420(6912):206–210, 2002.

[20] M.W. Hirsch, S. Smale, and R.L. Devaney. *Differential equations, dynamical systems, and an introduction to chaos*. Academic Press, 2012.

[21] B. Liu, D. Hsu, and PS Thiagarajan. Probabilistic approximations of ODEs based bio-pathway dynamics. *Theor. Comput. Sci.*, 412(21):2188–2206, 2011.

[22] Bing Liu, Andrei Hagiescu, Sucheendra K. Palaniappan, Bipasa Chattopadhyay, Zheng Cui, Weng-Fai Wong, and P. S. Thiagarajan. Approximate probabilistic analysis of biopathway dynamics. *Bioinformatics*, 28(11):1508–1516, 2012.

[23] John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computation on graphics hardware. In *Computer*

graphics forum, volume 26, pages 80–113. Wiley Online Library, 2007.

[24] Sucheendra K Palaniappan, Benjamin M Gyori, Bing Liu, David Hsu, and PS Thiagarajan. Statistical model checking based calibration and analysis of bio-pathway models. In *Computational Methods in Systems Biology*, pages 120–134. Springer, 2013.

[25] Grégory Batt, Delphine Ropers, Hidde De Jong, Johannes Geiselmann, Michel Page, and Dominique Schneider. Qualitative analysis and verification of hybrid models of genetic regulatory networks: Nutritional stress response in escherichia coli. In *Hybrid Systems: Computation and Control*, pages 134–150. Springer, 2005.

[26] Thao Dang, Colas Le Guernic, and Oded Maler. Computing reachable states for nonlinear biological models. In *International Conference on Computational Methods in Systems Biology*, pages 126–141. Springer, 2009.

[27] Adám Halász, Vijay Kumar, Marcin Imielinski, Calin Belta, Oleg Sokolsky, Sen Pathak, and Harvey Rubin. Analysis of lactose metabolism in e. coli using reachability analysis of hybrid systems. *IET Systems Biology*, 1(2):130–148, 2007.

[28] Shuai Che, Jie Li, Jeremy W Sheaffer, Kevin Skadron, and John Lach. Accelerating compute-intensive applications with gpus and fpgas. In *Application Specific Processors, 2008. SASP 2008. Symposium on*, pages 101–107. IEEE, 2008.

[29] Tesla P100 GPU accelerator. `https://www.nvidia.com/object/tesla-p100.html`.

[30] Victor Bryant. *Metric spaces: iteration and application*. Cambridge University Press, 1985.

[31] E. Klipp, R. Herwig, A. Kowald, C. Wierling, and H. Lehrach. *Systems biology in practice: concepts, implementation and application*. Wiley-VCH, Weinheim, 2005.

[32] James R Norris. *Markov chains*. Number 2. Cambridge university press, 1998.

[33] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

[34] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[35] Amir Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[36] Edmund Clarke and E Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. *Logics of programs*, pages 52–71, 1982.

[37] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[38] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[39] Rajeev Alur and David L Dill. A theory of timed automata. *Theoretical computer science*, 126(2):183–235, 1994.

[40] Rajeev Alur and Thomas A Henzinger. A really temporal logic. *Journal of the ACM (JACM)*, 41(1):181–203, 1994.

[41] Paolo Zuliani, André Platzer, and Edmund M Clarke. Bayesian statistical model checking with application to stateflow/simulink verification. *Formal Methods in System Design*, 43(2):338–367, 2013.

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

[43] Håkan LS Younes, Marta Kwiatkowska, Gethin Norman, and David Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer*, 8(3):216–228, 2006.

[44] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. In *VMCAI'04*, pages 73–84, 2004.

[45] Sumit K Jha, Edmund M Clarke, Christopher J Langmead, Axel Legay, André Platzer, and Paolo Zuliani. A bayesian approach to model checking biological systems. In *Computational Methods in Systems Biology*, pages 218–234. Springer, 2009.

[46] Håkan LS Younes. Error control for probabilistic model checking. In *Verification, Model Checking, and Abstract Interpretation*, pages 142–156. Springer, 2006.

[47] Thomas A. Henzinger. The theory of hybrid automata. In *LICS'96*, pages 278–292, 1996.

[48] Bing Liu, PS Thiagarajan, and David Hsu. Probabilistic approximations of signaling pathway dynamics. In *International Conference on Computational Methods in Systems Biology*, pages 251–265. Springer, 2009.

[49] Lorenzo Dematté and Davide Prandi. Gpu computing for systems biology. *Briefings in bioinformatics*, 11(3):323–333, 2010.

[50] Yanxiang Zhou, Juliane Liepe, Xia Sheng, Michael PH Stumpf, and Chris Barnes. Gpu accelerated biochemical network simulation. *Bioinformatics*, 27(6):874–876, 2011.

[51] John D Owens, Mike Houston, David Luebke, Simon Green, John E Stone, and James C Phillips. Gpu computing. *Proceedings of the IEEE*, 96(5):879–899, 2008.

[52] Mark Silberstein, Assaf Schuster, Dan Geiger, Anjul Patney, and John D Owens. Efficient computation of sum-products on gpus through software-managed cache. In *Proceedings of the 22nd annual international conference on Supercomputing*, pages 309–318. ACM, 2008.

[53] Xiaochun Ye, Dongrui Fan, Wei Lin, Nan Yuan, and Paolo Ienne. High performance comparison-based sorting algorithm on many-core gpus. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–10. IEEE, 2010.

[54] Francois Bodin and Stephane Bihan. Heterogeneous multicore parallel programming for graphics processing units. *Scientific Programming*, 17(4):325–336, 2009.

[55] Michael Wolfe. Implementing the pgi accelerator model. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, pages 43–50. ACM, 2010.

[56] Shane Ryoo, Christopher I Rodrigues, Sara S Baghsorkhi, Sam S Stone, David B Kirk, and Wen-mei W Hwu. Optimization principles and application performance evaluation of a multithreaded gpu using cuda. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*, pages 73–82. ACM, 2008.

[57] Cedric Bastoul. Code generation in the polyhedral model is easier than you think. In *Proceedings of the 13th International Conference on Parallel Architectures and Compilation Techniques*, pages 7–16. IEEE Computer Society, 2004.

[58] Long Chen, Oreste Villa, and Guang R Gao. Exploring fine-grained task-based execution on multi-gpu systems. In *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*, pages 386–394. IEEE, 2011.

[59] Amir H Hormati, Mehrzad Samadi, Mark Woh, Trevor Mudge, and Scott Mahlke. Sponge: portable stream programming on graphics engines. In *ACM SIGPLAN Notices*, volume 46, pages 381–392. ACM, 2011.

[60] Andrei Hagiescu, Huynh Phung Huynh, Weng-Fai Wong, and Rick Siow Mong Goh. Automated architecture-aware mapping of streaming applications onto gpus. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 467–478. IEEE, 2011.

[61] Lian Li, Hui Feng, and Jingling Xue. Compiler-directed scratch-pad memory management via graph coloring. *ACM Transactions on Architecture and Code Optimization (TACO)*, 6(3):9, 2009.

[62] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[63] Akio Maeda, Yu-ichi Ozaki, Sudhir Sivakumaran, Tetsuro Akiyama, Hidetoshi Urakubo, Ayako Usami, Miharu Sato, Kozo Kaibuchi, and Shinya Kuroda. Ca2+-independent phospholipase

a2-dependent sustained rho-kinase activation exhibits all-or-none response. *Genes to Cells*, 11(9):1071–1083, 2006.

[64] Albert Goldbeter and Olivier Pourquié. Modeling the segmentation clock as a network of coupled oscillations in the notch, wnt and fgf signaling pathways. *Journal of theoretical biology*, 252(3):574–585, 2008.

[65] S.L. Spencer, S. Gaudet, J.G. Albeck, J.M. Burke, and P.K. Sorger. Non-genetic origins of cell-to-cell variability in TRAIL-induced apoptosis. *Nature*, 459(7245):428–432, 2009.

[66] Berend Snijder and Lucas Pelkmans. Origins of regulated cell-to-cell variability. *Nature reviews Molecular cell biology*, 12(2):119–125, 2011.

[67] Andrea Y Weiße, Richard H Middleton, and Wilhelm Huisinga. Quantifying uncertainty, variability and likelihood for ordinary differential equation models. *BMC systems biology*, 4(1):144, 2010.

[68] Carmen G. Moles, Pedro Mendes, and Julio R. Banga. Parameter estimation in biochemical pathways: A comparison of global optimization methods. *Genome Res.*, 13(11):2467 –2474, 2003.

[69] Thomas P Runarsson and Xin Yao. Stochastic ranking for constrained evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*, 4(3):284–294, 2000.

[70] Håkan LS Younes and Reid G Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Information and Computation*, 204(9):1368–1409, 2006.

[71] D.E. Goldberg. *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley, 1989.

[72] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE micro*, 28(2):39–55, 2008.

[73] Edmund M Clarke, James R Faeder, Christopher J Langmead, Leonard A Harris, Sumit Kumar Jha, and Axel Legay. Statistical model checking in biolab: Applications to the automated analysis of t-cell receptor signaling pathway. In *Computational Methods in Systems Biology*, pages 231–250. Springer, 2008.

[74] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *Runtime Verification*, pages 122–135. Springer, 2010.

[75] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *Computer Aided Verification*, pages 266–280. Springer, 2005.

[76] Peter Bulychev, Alexandre David, Kim Gulstrand Larsen, Marius Mikučionis, Danny Bøgsted Poulsen, Axel Legay, and Zheng Wang. Uppaal-smc: Statistical model checking for priced timed automata. *arXiv preprint arXiv:1207.1272*, 2012.

[77] Abraham Wald. Sequential tests of statistical hypotheses. *The Annals of Mathematical Statistics*, 16:117–186, 1945.

[78] Lars Kuhtz and Bernd Finkbeiner. Efficient parallel path checking for linear-time temporal logic with past and bounds. *arXiv preprint arXiv:1210.0574*, 2012.

[79] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[80] Benjamin Barre, Mathieu Klein, Maxime Soucy-Boivin, Pierre-Antoine Ollivier, and Sylvain Hallé. Mapreduce for parallel trace validation of ltl properties. In *Runtime Verification*, pages 184–198. Springer, 2013.

[81] Jiri Barnat, Lubos Brim, Ivana Cerná, Sven Drazan, Jana Fabriková, Jan Láník, David Safránek, and Hongwu Ma. Biodivine: A framework for parallel analysis of biological models. In *Proceedings Second International Workshop on Computational Models for Cell Processes, COMPMOD 2009, Eindhoven, the Netherlands, November 3, 2009.*, pages 31–45, 2009.

[82] Jiri Barnat, Lubos Brim, Milan Ceska, and Tomas Lamr. Cuda accelerated ltl model checking. In *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, pages 34–41. IEEE, 2009.

[83] Kosuke Oshima, Takeshi Matsumoto, and Masahiro Fujita. Hardware implementation of bltl property checkers for acceleration of statistical model checking. In *Proceedings of the International Conference on Computer-Aided Design*, pages 670–676. IEEE Press, 2013.

[84] Alexandre David, Dehui Du, Kim Guldstrand Larsen, Axel Legay, and Marius Mikučionis. Optimizing control strategy using statistical model checking. In *NASA formal methods*, pages 352–367. Springer, 2013.

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

[86] Luca Bortolussi and Guido Sanguinetti. Learning and designing stochastic processes from logical constraints. In *Quantitative Evaluation of Systems*, pages 89–105. Springer, 2013.

[87] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without bdds. In *Intl. Conf. on Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99)*, volume 1579. Springer, 1999.

[88] N. Le Novere, B. Bornstein, A. Broicher, M. Courtot, M. Donizelli, H. Dharuri, L. Li, H. Sauro, M. Schilstra, B. Shapiro, J.L. Snoep, and M. Hucka. BioModels Database: A free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Res.*, 34:D689–D691, 2006.

[89] John Denholm Lambert. *Numerical methods for ordinary differential systems: the initial value problem*. John Wiley & Sons, Inc., 1991.

[90] Akio Maeda, Yu-ichi Ozaki, Sudhir Sivakumaran, Tetsuro Akiyama, Hidetoshi Urakubo, Ayako Usami, Miharu Sato, Kozo Kaibuchi, and Shinya Kuroda. Ca2+-independent phospholipase a2-dependent sustained rho-kinase activation exhibits all-or-none response. *Genes to Cells*, 11(9):1071–1083, 2006.

[91] Akio Maedo, Yuichi Ozaki, Sudhir Sivakumaran, Tetsuro Akiyama, Hidetoshi Urakubo, Ayako Usami, Miharu Sato, Kozo Kaibuchi, and Shinya Kuroda. Ca^{2+}-independent phospholipase A2-dependent sustained Rho-kinase activation exhibits all-or-none response. *Genes Cells*, 11:1071–1083, 2006.

[92] Doug Bruce, Pras Pathmanathan, and Jonathan P Whiteley. Modelling the effect of gap junctions on tissue-level cardiac electrophysiology. *Bulletin of mathematical biology*, 76(2):431–454, 2014.

[93] Evelyn Buckwar and Martin G Riedler. An exact stochastic hybrid model of excitable membranes including spatio-temporal evolution. *Journal of mathematical biology*, 63(6):1051–1093, 2011.

[94] Eugene Asarin, Oded Maler, and Amir Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical computer science*, 138(1):35–65, 1995.

[95] Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *HSCC'06*, pages 257–271, 2006.

[96] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *HSCC'05*, pages 258–273, 2005.

[97] Edmund Clarke, Ansgar Fehnker, Zhi Han, Bruce Krogh, Olaf Stursberg, and Michael Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *TACAS'03*, pages 192–207, 2003.

[98] Rajeev Alur, Thomas A Henzinger, Gerardo Lafferriere, and George J Pappas. Discrete abstractions of hybrid systems. *P. IEEE*, 88(7):971–984, 2000.

[99] Manindra Agrawal, Frank Stephan, PS Thiagarajan, and Shaofa Yang. Behavioural approximations for restricted linear differential hybrid automata. In *HSCC'06*, pages 4–18, 2006.

[100] Thomas A Henzinger and Peter W Kopke. Discrete-time control for rectangular hybrid automata. *Theor. Comput. Sci.*, 221(1):369–392, 1999.

[101] Edmund M Clarke, Orna Grumberg, and Doron A Peled. *Model checking*. MIT press, 1999.

[102] Rajeev Alur and Thomas A. Henzinger. Modularity for timed and hybrid systems. In *CONCUR'97*, pages 74–88, 1997.

[103] Alessandro Abate, Aaron D Ames, and S Shankar Sastry. Stochastic approximations of hybrid systems. In *ACC'05*, pages 1557–1562, 2005.

[104] A Agung Julius and George J Pappas. Approximations of stochastic hybrid systems. *IEEE T. Automat. Contr.*, 54(6):1193–1203, 2009.

[105] Alexandre David, Dehui Du, Kim G. Larsen, Axel Legay, Marius Mikučionis, Danny Bøgsted Poulsen, and Sean Sedwards. Statistical model checking for stochastic hybrid systems. In *HSB'12*, pages 122–136, 2012.

[106] Sicun Gao, Soonho Kong, and Edmund Clarke. Delta-complete reachability analysis (part i). In *Technical report, CMU SCS, CMU-CS-13-131*, 2013.

[107] Paolo Ballarini, Hilal Djafri, Marie Duflot, Serge Haddad, and Nihal Pekergin. COSMOS: a statistical model checker for the hybrid automata stochastic logic. In *QEST'11*, pages 143–144, 2011.

[108] Christos G Cassandras and John Lygeros. *Stochastic hybrid systems*. CRC Press, 2010.

[109] Henk AP Blom, John Lygeros, M Everdij, S Loizou, and K Kyriakopoulos. *Stochastic hybrid systems: Theory and safety critical applications*. Springer Heidelberg, 2006.

[110] Willard Stephen. General topology, 1970.

[111] Alfonso Bueno-Orovio, Elizabeth M. Cherry, and Flavio H. Fenton. Minimal model for human ventricular action potentials in tissue. *J. Theor. Biol.*, 253:544–560, 2008.

[112] H Matsuno, S. T. Inouye, Y. Okitsu, Y. Fujii, and S. Miyano. A new regulatory interaction suggested by simulations for circadian genetic control mechanism in mammals. *J Bioinform Comput Biol*, 4(1):139–153, 2006.

[113] Bing Liu, Soonho Kong, Sicun Gao, and E Clarke. Parameter identification using delta-decisions for biological hybrid systems. Technical report, CMU SCS Technical Report, CMU-CS-13-136, 2014.

[114] K Tanaka, S Zlochiver, K. L. Vikstrom, M Yamazaki, J Moreno, M Klos, A. V. Zaitsev, R Vaidyanathan, D. S. Auerbach, S Landas, G Guiraudon, J Jalife, O Berenfeld, and J. Kalifa. Spatial distribution of fibrosis governs fibrillation wave dynamics in the posterior left atrium during heart failure. *Circ. Res.*, 8(101):839–847, 2007.

[115] B. Liu, S. Kong, S. Gao, P. Zuliani, and E. M. Clarke. Parameter synthesis for cardiac cell hybrid models using δ-decisions. In *CMSB'14*, pages 99–113, 2014.

[116] F. Fenton and A. Karma. Vortex dynamics in 3D continuous myocardium with fiber rotation: filament instability and fibrillation. *Chaos*, 8:20–47, 1998.

[117] K. Nakamura, R. Yoshida, M. Nagasaki, S. Miyano, and T. Higuchi. Parameter estimation of in silico biological pathways with particle filtering towards a petascale computing. In *PSB'09*, pages 227–238, 2009.

[118] L. Shearman, S. Sriram, D. Weaver, E. Maywood, I. Chaves, B. Zheng, K. Kume, C. Lee, G. van der Horst, M. Hastings, and S. Reppert. Interacting molecular loops in the mammalian circadian clock. *Science*, 288:1013–1019, 2000.

[119] J. K. Kim and D. B. Forger. A mechanism for robust circadian timekeeping via stoichiometric balance. *Mol Syst Biol*, 8:1–14, 2012.

[120] Luca Bortolussi and Alberto Policriti. The importance of being (a little bit) discrete. *Electronic Notes in Theoretical Computer Science*, 229(1):75–92, 2009.

A

APPENDIX

A.1 QUANTITATIVE SPECIFICATIONS

To specify quantitative properties we fix a finite set of atomic propositions AP_{qt} of the form $\langle x_i < c \rangle$ or $\langle x_i > c \rangle$ where c is a rational constant. In what follows we shall assume for convenience that all the atomic propositions that we encounter are members of AP_{qt}. It will be straightforward to extend our arguments to include qualitative atomic propositions as well.

We partition $\Re^n$ into hypercubes according to the constants mentioned in the quantitative atomic propositions in AP_{qt}. (Actually one could just focus on the members of AP_{qt} that appear in a given specification but we wish to deal with specifications later). Accordingly, define C_i to be the set of rational constants so that $c \in C_i$ iff an atomic proposition of the form $\langle x_i < c \rangle$ or $\langle x_i > c \rangle$ appears in AP_{qt}. We next define for each dimension i the set of intervals

$$\mathcal{I}_i = \{(-\infty, c_i^1), \{c_i^1\}, (c_i^1, c_i^2), \{c_i^2\}, \ldots (c_i^m, +\infty)\}$$

where $C_i = \{c_i^1 < c_i^2 < \ldots < c_i^m\}$. In case $C_i = \emptyset$ we set $\mathcal{I}_i = \{(-\infty, +\infty)\}$.

This leads to the set of hypercubes $\mathcal{H}$ given by $\mathcal{H} = \{\prod_i I_i \mid I_i \in \mathcal{I}_i\}$. Clearly $\mathcal{H}$ is a partition of $\Re^n$. The states of the Markov chain M_{qt} we wish to define as the approximation of H will be the states of M defined previously but now refined using $\mathcal{H}$. More precisely we define $M_{qt} =$

$(Y_{qt}, \Rightarrow_{qt})$ inductively as follows: $\epsilon \in Y_{qt}$ and it is the initial state of M_{qt}. Every other state in Y_{qt} will be of the form $(\rho, X, \mathfrak{h}, \mathbf{P}_X)$ where ρ is a path in H, X is an open subset of $\mathbb{R}^n$ of finite non-zero measure, $\mathfrak{h} \in \mathcal{H}$ and $\mathbf{P}_X$ is a probability distribution over X. Furthermore $X \subseteq \mathfrak{h}$.

A.1.1 *The two semantics*

For interpreting *BLTL* formulas over M_{qt} it will be convenient to assume the following syntax in which negation is immediately followed by a quantitative atomic proposition:

$$A \mid \neg A \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid G^{\leq k}\varphi \mid F^{\leq k}\varphi \mid \varphi_1 \mathbf{U}^{\leq k} \varphi_2.$$

Clearly, every BLTL formula can be transformed into an equivalent formula that has the above syntax. This can be achieved by pushing negation inwards using equivalences such as $\neg(\varphi_1 \vee \varphi_2) \equiv \neg\varphi_1 \wedge \neg\varphi_2$, $\neg G^{\leq k}\varphi \equiv F^{\leq k}\neg\varphi$, $\neg(\varphi_1 \mathbf{U}^{\leq k}\ \varphi_2) \equiv G^{\leq k}\neg\varphi_2 \vee (\neg\varphi_2 U^{\leq k}(\neg\varphi_1 \wedge \neg\varphi_2))$ etc.

The trajectory semantics is defined along previous lines but the atomic propositions are handled as follows. Let $\tau = (q_0, \mathbf{v}_0)\ (q_1, \mathbf{v}_1)\ \ldots\ (q_k, \mathbf{v}_k)$ be a finite trajectory and $0 \leq \ell \leq k$. Then $\tau, \ell \models_{H,qt} \langle x_i < c\rangle$ iff $\mathbf{v}_\ell(i) < c$. On the other hand $\tau, \ell \models_{H,qt} \neg\langle x_i < c\rangle$ iff $\tau, \ell \not\models_H \langle x_i < c\rangle$. The clauses for the other cases are defined in the obvious way. As before τ is a (trajectory) model of ψ iff $\tau \in TRJ^{K+1}$ and $\tau, 0 \models_{H,qt} \psi$.

To interpret BLTL formulas over M_{qt}, let $\pi = \eta_0\eta_1 \ldots \eta_k$ be a path in M_{qt} with $\eta_0 = \epsilon$ and $\eta_\ell = (\rho q_\ell, X_\ell, \mathfrak{h}_\ell, \mathbf{P}_{X_\ell})$ for $0 < \ell \leq k$. Let ψ be a BLTL formula and $0 < \ell \leq k$. Then $\pi, \ell \models_{qt} \psi$ is given by:

- $\pi, \ell \models_{qt} \langle x_i < c\rangle$ iff there exists $\mathbf{v} \in X_\ell$ such that $\mathbf{v}(i) < c$.
- $\pi, \ell \models_{qt} \neg\langle x_i < c\rangle$ iff there exists $\mathbf{v} \in X_\ell$ such that $\mathbf{v}(i) \geq c$.

- The remaining clauses are defined in the obvious way.

For $\mathbf{v} \in \Re^n$ let $\mathbf{v} \models A$ denote the fact that $\mathbf{v}(i) < c$ in case $A = \langle x_i < c \rangle$ and $\mathbf{v}(i) > c$ in case $A = \langle x_i > c \rangle$. Next suppose $(\rho, X, \mathfrak{h}, \mathbf{P}_X)$ is a state of M_{qt} and $A \in AP_{qt}$. Then $X \subseteq \mathfrak{h}$ by construction. Furthermore it is easy to check that $\mathbf{v} \models A$ for every $\mathbf{v} \in \mathfrak{h}$ or $\mathbf{v} \not\models A$ for every $\mathbf{v} \in \mathfrak{h}$. Thus the semantics defined above will be consistent in the sense it will be the case that either $\pi, \ell \models_{qt} A$ or $\pi, \ell \models_{qt} \neg A$ but not both.

Let $\mathcal{B}$ be the set of paths of length $K+2$ that start from the initial state of M_{qt}. Now we define $models_{qt}(\psi) \subseteq \mathcal{B}$ via: $\pi \in models_{qt}(\psi)$ iff $\pi, 1 \models_{qt} \psi$. We can now define the probability of satisfaction of a formula in M_{qt}. Let $\pi = \eta_0 \eta_1 \ldots \eta_{K+1} \in \mathcal{B}$. Then $\Pr(\pi) = \prod_{0 \leq \ell < K} p_\ell$, where $\eta_\ell \stackrel{p_\ell}{\Rightarrow} \eta_{\ell+1}$ for $0 \leq \ell < K+1$. This leads to

$$\Pr(models_{qt}(\psi)) = \sum_{\pi \in models_{qt}(\psi)} \Pr(\pi).$$

We let $M_{qt} \models \psi$ denote the fact $\Pr(models_{qt}(\psi)) = 1$.

A.1.2 *The correspondence result*

We shall relate the behavior of H to that M_{qt} using the notion of *robust* trajectories. To start with, for $\mathbf{v} \in \Re^n$ we let $hc(\mathbf{v})$ be the hypercube $\mathfrak{h}$ in $\mathcal{H}$ such that $\mathbf{v} \in \mathfrak{h}$. Since $\mathcal{H}$ is a partition of $\Re^n$ we have that $hc(\mathbf{v})$ exists and is unique. In what follows we let ℓ range over $\{0, 1, \ldots, K\}$. We now define the equivalence relation $\approx \subseteq TRJ^{K+1}$ as follows: Let $\tau, \tau' \in TRJ^{K+1}$ with $\tau(\ell) = (q_\ell, \mathbf{v}_\ell)$ and $\tau'(\ell) = (q'_\ell, \mathbf{v}'_\ell)$. Then $\tau \approx \tau'$ iff $q_\ell = q'_\ell$ and $hc(\mathbf{v}_\ell) = hc(\mathbf{v}'_\ell)$ for each ℓ. We let $[\tau]$ denote the $\approx$-equivalence class containing τ.

Next suppose $\tau \in TRJ^{K+1}$ with $\tau(\ell) = (q_\ell, \mathbf{v}_\ell)$. Let $\mathcal{Q}(\tau, \ell) = q_\ell$ and $\mathcal{V}(\tau, \ell) = \mathbf{v}_\ell$. Define $[\tau](\ell) = \{\mathcal{V}(\tau', \ell) \mid \tau' \in [\tau]\}$. It is easy to verify that $[\tau](\ell)$ is a measurable set (but perhaps with measure 0) for each ℓ.

The trajectory $\tau \in TRJ^{K+1}$ is said to be *robust* iff $\mu([\tau](\ell)) > 0$ for every ℓ. We will say that H robustly satisfies the specification ψ-and this is denoted by $H \models_R \psi$ iff $\tau, 0 \models_H \psi$ for every robust trajectory τ in TRJ^{K+1}. It is now straightforward to show (along the lines of the proof of Theorem 2) show:

Theorem 4 *$H \models_R \psi$ iff $M_{qt} \models \psi$.*

First the following properties of the Markov chain M_{qt} can easily be proved along the lines of the proof of Theorem 1.

Lemma 2

1. *$X_j^{\mathfrak{h}}$ is open and is of finite measure for each j and each $\mathfrak{h} \in \mathcal{H}$.*
2. *If $(\rho q_j, X_j^{\mathfrak{h}}, \mathfrak{h}, \boldsymbol{P}_{X_j^{\mathfrak{h}}}) \in \mathrm{Y}_{qt}$ then $\mu(X_j^{\mathfrak{h}}) > 0$.*
3. *$\boldsymbol{P}_{X_j^{\mathfrak{h}}}$ is a probability distribution for each j and each $\mathfrak{h} \in \mathcal{H}$.*
4. *$M_{qt} = (\mathrm{Y}_{qt}, \Rightarrow_{qt})$ is an infinite-state Markov chain whose underlying graph is a finitely branching tree.*

We wish to show that for quantitative specifications, H robustly satisfies a BLTL specification ψ if and only if M_{qt} satisfies ψ with probability 1. We begin with:

Lemma 3 *Let $\tau = (q_0, \boldsymbol{v}_0), (q_1, \boldsymbol{v}_1), \ldots (q_K, \boldsymbol{v}_K) \in TRJ^{K+1}$. Then the following statements are equivalent.*

1. *τ is robust.*
2. *There exist open sets of non-zero measure O_j and $\mathfrak{h}_j \in \mathcal{H}$ such that $\boldsymbol{v}_j \in O_j \subseteq [\gamma][j] \subseteq \mathfrak{h}_j$ for $0 \leq j \leq K$.*
3. *$\boldsymbol{v}_j(i) \notin C_i$ for every $j \in \{0, 1, \leq K\}$ and every $i \in \{1, 2, \ldots, n\}$.*

PROOF In what follows we let j range over $\{0, 1, \ldots, K\}$. Suppose τ is robust. Let $hc(\mathbf{v}_j) = \mathfrak{h}_j$ for each j. By the definition of $\approx$, we have $\mathbf{v}_j \in [\tau](j) \subseteq \mathfrak{h}_j$ for each j. Since $\mu([\tau](j)) > 0$ we have $\mu(\mathfrak{h}_j) > 0$ for each j. This implies that $\mathfrak{h}_j(i)$ is a finite open interval for $1 \leq i \leq n$. But then $[\tau](j) \subseteq \mathfrak{h}_j$ and $\mu([\tau](j)) > 0$ now together imply that there exists a non-empty open set O_j of finite measure such that $\mathbf{v}_j \in O_j \subseteq [\tau](j)$ for each j. Thus (1) implies (2).

Next suppose part (2) of the lemma holds. Then $\mu([\tau](j)) > 0$ for each j. Thus τ is robust and we have (2) implies (1).

To show that (2) implies (3) assume that $\mathbf{v}_j(i) \in C_i$ for some j and i. Then $\mu(hc(\mathbf{v}_j)) = 0$. We need to find $\mathfrak{h}_j$ and an open set of non-zero measure such that $\mathbf{v}_j \in O_j \subset [\tau](j) \subseteq \mathfrak{h}_j$. This implies $hc(\mathbf{v}_j) = \mathfrak{h}_j$. But then $\mu(\mathfrak{h}_j) = 0$ implies there can not exist an open set O_j of *non-zero measure* satisfying $\mathbf{v}_j \in O_j \subseteq \mathfrak{h}_j$. Hence (2) can not hold and this shows (2) implies (3).

Next suppose (3) holds. Let $\mathfrak{h}_j = hc(\mathbf{v}_j)$ for each j. Then (3) implies $\mu(\mathfrak{h}_j) > 0$ for each j. Let $\tau^{(j)}$ be the j-length prefix of τ for each j.

Since INIT is open $O_0 = \text{INIT} \cap \mathfrak{h}_0$ is open. It is non-empty since $\mathbf{v}_0 \in O_0$ and hence has non-zero measure. Furthermore $[\tau^{(0)}](0) = O_0$. We now have $\mathbf{v}_0 \in O_0 \subseteq [\tau^{(0)}](0) \subseteq \mathfrak{h}_0$. Assume inductively $0 < j < K$ and for $0 \leq k \leq j$ there exist open sets O_k of non-zero measure such that $\mathbf{v}_k \in O_k \subseteq [\tau^{(j)}](k) \subseteq \mathfrak{h}_k$.

Since τ is a trajectory there exist g_j and $t_j \in (0, 1)$ such that $q_j \xrightarrow{g_j} q_{j+1}$ and $\Phi_{q_j,t_j}(\mathbf{v}_j) \in g_j$ and $\mathbf{v}_{j+1} = \Phi_{q_{j+1},1-t_j}(\Phi_{q_j,t_j}(\mathbf{v}_j))$. Let $Y_j = [\tau^{(j)}](j)$ and $Y'_{j+1} = \bigcup_{\mathbf{v}\in Y_j}\{\Phi_{q_{j+1},1-t}(\Phi_{q_j,t}(\mathbf{v}))|t \in \mathbb{T}(\mathbf{v})\}$ where $\mathbb{T}(\mathbf{v}) = \{t|\Phi_{q_j,t}(\mathbf{v}) \in g\}$. Clearly $[\tau^{(j+1)}](j+1) = Y'_{j+1} \cap \mathfrak{h}_{j+1}$. Next define $O'_{j+1} = \Phi_{q_{j+1},1-t_j}(\Phi_{q_j,t_j}(O_j))$. Since both $\Phi^{-1}_{q_j,1-t_j}$ and $\Phi^{-1}_{q_j,t_j}$ are continuous bijections, O'_{j+1} is an open set and $\mathbf{v}_{j+1} \in O'_{j+1}$. Let $O_{j+1} = O'_{j+1} \cap \mathfrak{h}_{j+1}$. Since $\mathbf{v}_{j+1} \in \mathfrak{h}_{j+1}$ and $\mathfrak{h}_{j+1}$ is open we have O_{j+1} is open and non-empty and hence with non-

zero measure. Further $O_{j+1} \subseteq [\tau^{(j+1)}](j+1) \subseteq \mathfrak{h}_{j+1}$. This establishes the induction hypothesis and hence (3) implies (2). □

We define the notion of *compatibility* as before. Let $\pi = \eta_0\eta_1 \ldots \eta_k$ be a path in M_{qt} with $\eta_j = (q_0q_1 \ldots q_{j-1}, X_j^{\mathfrak{h}_j}, \mathfrak{h}_j, \Pr_{X_j^{\mathfrak{h}_j}})$ for $0 < j \leq k$, and $\eta_0 = \epsilon$. Let $\tau = (q'_1, \mathbf{v}_1)(q'_2, \mathbf{v}_1) \ldots (q'_{k'}, \mathbf{v}_{k'})$ be a trajectory. Then we say that π and τ are *compatible* iff $k = k'$ and for $1 \leq j \leq k$, $q_j = q'_j$ and $\mathbf{v}_j \in X_j^{\mathfrak{h}_j}$. As it will turn out, if τ and π are compatible then τ will be robust.

In what follows we shall assume that our BLTL specifications involve only quantitative atomic propositions in AP_{qt} and the formulas obey the syntax in which negation is immediately followed by an atomic proposition. Further the semantic notions $\models_H$ and $\models_{M_{qt}}$ (abbreviated as $\models_{qt}$) are defined in the expected way.

Lemma 4 *1. Suppose the trajectory $\tau = (q_1, \boldsymbol{v}_1)(q_2, \boldsymbol{v}_1) \ldots (q_k, \boldsymbol{v}_k) \in TRJ$ and the path $\pi = \eta_0\eta_1 \ldots \eta_k$ in M_{qt} with $\eta_0 = \epsilon$ are compatible. Let ψ be a BLTL specification and $j \in \{1, \ldots k\}$. Then $\tau, j \models_H \psi$ iff $\pi, j \models_{qt} \psi$.*

2. Suppose π is a path in M_{qt} starting from ϵ. Then there exists a robust trajectory τ in TRJ such that π and τ are compatible.

3. Suppose τ is a robust trajectory in TRJ. Then there exists a path π in M_{qt} starting from ϵ such that τ and π are compatible.

PROOF

1. From the definitions it follows that if $A \in AP_{qt}$ and $\mathfrak{h} \in \mathcal{H}$ then $\mathbf{v} \models A$ for every $\mathbf{v} \in \mathfrak{h}$ or $\mathbf{v} \models \neg A$ for every $\mathbf{v} \in \mathfrak{h}$ but not both. Since $\mathbf{v}_j \in \mathfrak{h}_j$ we then have $\tau, j \models_H A$ iff $\pi, j \models_{qt} A$ and $\tau, j \models_H \neg A$ iff $\pi, j \models_{qt} \neg A$ for every atomic proposition. The remaining cases now follow easily by structural induction on ψ.

2. Let $\pi = \eta_0\eta_1 \dots \eta_k$ in M_{qt} with $\eta_0 = \epsilon$ and $\eta_j = (q_0q_1 \dots q_{j-1}, X_j^{\mathfrak{h}_j}, \mathfrak{h}_j, \Pr_{X_j^{\mathfrak{h}_j}})$ for $0 < j \leq k$. For notational convenience we will write X_j instead of $X_j^{\mathfrak{h}_j}$.

   Since $\mu(X_k) > 0$ we can fix $\mathbf{v}_k \in X_k$. Further $\mathfrak{h}_k$ being a product of open intervals in $\Re$ with $X_k \subseteq \mathfrak{h}_k$, we can find an open set O_k of non-zero measure such that $\mathbf{v}_k \in O_k \subseteq X_k$. Thus we have $\mathbf{v}_k \in O_k \subseteq X_k \subseteq \mathfrak{h}_k$. From the construction of M_{qt} it follows there exists $q_{k-1} \xrightarrow{g} q_k$ and $\mathbb{T}(\mathbf{v}) \subseteq (0,1)$ for each $\mathbf{v} \in X_k$ such that $\Phi^{-1}_{q_k,1-t}(\mathbf{v}) \in g$ for every $t \in \mathbb{T}(\mathbf{v})$. Let $Y_{k-1} = \bigcup_{\mathbf{v}\in X_k}\{\Phi^{-1}_{q_{k-1},t}(\Phi^{-1}_{q_k,1-t}(\mathbf{v})) \mid t \in \mathbb{T}(\mathbf{v})\}$. From the construction of it follows that $Y_{k-1} \subseteq X_{k-1}$.

   Next let $O_{k-1} = \bigcup_{\mathbf{v}\in O_k}\{\Phi^{-1}_{q_{k-1},t}(\Phi^{-1}_{q_k,1-t}(\mathbf{v})) \mid t \in \mathbb{T}(\mathbf{v})\}$. Clearly $O_{k-1}$is an open set of non-zero measure with $O_{k-1} \subseteq Y_{k-1}$. Moreover we can fix $\mathbf{v}_{k-1} \in O_{k-1}$ such that $\mathbf{v}_{k-1} = \Phi^{-1}_{q_k,1-t}(\mathbf{v}_k)$ for some $t \in \mathbb{T}(\mathbf{v}_k)$. Continuing this way we can find $\mathbf{v}_j, O_j, Y_j$ for $1 \leq j \leq k$ (with $Y_k = X_k$) such that $\tau = (q_1, \mathbf{v}_1)(q_2, \mathbf{v}_2) \dots (q_k, \mathbf{v}_k)$ is a trajectory and $\mathbf{v}_j \in O_j \subseteq Y_j \subseteq \mathfrak{h}_j$ for $1 \leq j \leq k$. From the construction of M_{qt} it follows that $Y_j = [\tau](j)$ for $1 \leq j \leq k$. From Lemma 3 it follows that π and τ are compatible. It is also clear due to Lemma 3 that τ is robust.

3. Suppose $\tau = (q_1, \mathbf{v}_1)(q_2, \mathbf{v}_1) \dots (q_k, \mathbf{v}_k) \in TRJ$ is robust. Then by Lemma 3 there exist open sets O_j of non zero measure and $\mathfrak{h}_j \in \mathcal{H}$ such that $\mathbf{v}_j \in O_j \subseteq [\tau](j) \subseteq \mathfrak{h}_j$ for $1 \leq j \leq k$. Let $\tau^{(j)}$ denote the j-length prefix of τ for $1 \leq j \leq k$. We now define $X_j = [\tau^{(j)}](j)$ for $1 \leq j \leq k$. Then using the construction of M_{qt} it is easy to show that there exists distributions Pr_j over X_j such that $\pi = \epsilon\eta_1\eta_2 \dots \eta_k$ is a path in M_{qt} with $\eta_j = (q_j, X_j, \mathfrak{h}_j, Pr_j)$ for $1 \leq j \leq k$ and that π is compatible with τ.

□

We can now prove Theorem 5.

Theorem 5 *$H \models_R \psi$ iff $M_{qt} \models \psi$.*

PROOF *Suppose $H \not\models_R \psi$. Then there exists $\tau \in TRJ$ such that τ is robust and $\tau, 0 \not\models_H \psi$. By Lemma 4, there exists a path π in M_{qt} which is compatible with τ. Hence again by Lemma 4 we then have $\pi \notin models_{M_{qt}}(\psi)$ which leads to $Pr_{<1}(\psi)$. Next suppose that $Pr_{<1}(\psi)$. Then there exists a path π in M_{qt} such that $\pi, 1 \not\models_{M_{qt}} \psi$. By Lemma 4, there exists a robust trajectory τ which is compatible with π and $\tau, 0 \not\models_H \psi$. This implies $H \not\models_R \psi$.* □

Finally, we wish to show that the number of non-robust trajectories are negligible compared with the robust ones. Hence they do not contribute much towards the dynamics of H. For that we need the following lemma.

Lemma 5 *Suppose $\tau = (q_0, \boldsymbol{v}_0)(q_1, \boldsymbol{v}_1) \ldots (q_k, \boldsymbol{v}_k)$ is a non-robust trajectory and $\tau^{(j)}$ is the j-length prefix of τ for $1 \leq j \leq k+1$. Let $\mathfrak{h}_j = hc(\boldsymbol{v}_j)$ and $Y_j = [\tau^{(j+1)}](j+1)$ for $0 \leq j \leq k$. Then Y_j is measurable and $Y_j \subseteq \mathfrak{h}_j$ for $0 \leq j \leq k$. Furthermore Y_j is of measure o for each j in $\{0, 1, ..., k\}$.*

PROOF Since τ is not robust, there exists $j : 0 \leq j \leq k$ such that $\mathbf{v}_j(i) = c_i \in C_i$ for some i and hence for all $\mathbf{v} \in \mathfrak{h}_j$, $\mathbf{v}(i) = c_i$ which implies $\mu(\mathfrak{h}_j) = 0$. We induct on j. For $j = 0$, $Y_0 = \text{INIT} \cap \mathfrak{h}_0$ is measurable and has measure o. Suppose $q_0 \xrightarrow{g} q_1$ and let $Y_1' = \bigcup_{\mathbf{v} \in Y_0} \{\Phi_{q_1, 1-t}(\Phi_{q_0,t}(\mathbf{v})) \mid t \in \mathbb{T}(\mathbf{v})\}$ where $\mathbb{T}(\mathbf{v}) = \{t \mid \Phi_{q_0,t}(\mathbf{v}) \in g\}$. Then $Y_1 = Y_1' \cap \mathfrak{h}_1$. Let $\hat{Y}_1 = \Phi_{q_1}((0,1) \times \Phi_{q_0}((0,1) \times Y_0) \cap g)$. Since $\mu(Y_0) = 0$ hence $\mu((0,1) \times Y_0) = 0$. Now both Φ_{q_1} and Φ_{q_0} are Lipschitz, and hence $\mu(\hat{Y}_1) = 0$ [since the image of a set of measure o has measure o under a Lipschitz function]. Now note that $Y_1 \subseteq \hat{Y}_1$ and hence Y_1 must be measurable and $\mu(Y_1) = 0$. Continuing this way, we can show that Y_j is measurable for all $j : 2 \leq j \leq k$ and $\mu(Y_j) = 0$.

Next suppose $j > 0$. By a similar argument we can show that Y_ℓ is measurable for all $j < \ell \leq k$ and $\mu(Y_\ell) = 0$. Let $q_{j-1} \xrightarrow{g} q_j$ and let $Y'_{j-1} = \bigcup_{\mathbf{v} \in Y_j} \{\Phi^{-1}_{q_{j-1},1-t}(\Phi^{-1}_{q_j,t}(\mathbf{v})) \mid t \in \mathbb{T}(\mathbf{v})\}$ where $\mathbb{T}(\mathbf{v}) = \{t \mid \Phi_{q_{j-1},t}(\mathbf{v}) \in g\}$. Then $Y_{j-1} = Y'_{j-1} \cap \mathfrak{h}_{j-1}$. Let $\hat{Y}_{j-1} = \Phi_{q_{j-1}}((-1,0) \times \Phi_{q_j}((-1,0) \times Y_j) \cap g)$. Since $\mu(Y_j) = 0$ hence $\mu((-1,0) \times Y_j) = 0$. Now both Φ_{q_j} and $\Phi_{q_{j-1}}$ are Lipschitz, and hence $\mu(\hat{Y}_{j-1}) = 0$ [since the image of a set of measure o has measure o under a Lipschitz function]. Now note that $Y_{j-1} \subseteq \hat{Y}_{j-1}$ and hence Y_{j-1} must be measurable and $\mu(Y_{j-1}) = 0$. Continuing this way, we can show that Y_m is measurable for all $m : 0 \leq m < j$ and $\mu(Y_m) = 0$. □ □

Thus by the above lemma, if a trajectory $\tau \in TRJ^{K+1}$ is not robust then there exists a $j \in \{0, 1, \ldots, K\}$ such that $\mu(Y_j) = 0$. This implies that in the product topology of $Q^{K+1} \times \mathbb{R}^{K+1}$, $[\tau]$ has measure o. Thus, the contribution made by the non-robust trajectories to the dynamics of H is negligible.

Thus in terms of the sub-dynamics consisting of robust trajectories there is again a strong relationship between the behaviors of H and M_{qt}. It also turns out that in measure-theoretic terms the non-robust trajectories can be ignored. More precisely if one starts with the discrete topology over Q^{K+1} and the usual topology over $\Re^{n^{K+1}}$ one can easily define a natural measure space over the product topology $Q^{K+1} \times \Re^{n^{K+1}}$. In this space for every non-robust trajectory τ the representation of $[\tau]$ will be measurable but with measure 0. In this sense the contributions made by the non-robust trajectories to the dynamics of H are negligible.

A.1.3 *Trajectory simulation for quantitative specifications*

Algorithm 3 gives the procedure for simulating robust trajectories for the verification of quantitative BLTL specifications. By Lemma 3, a tra-

jectory is robust iff it does not hit any of the constants mentioned in the atomic propositions. The procedure is the same as Algorithm 1 before, except that whenever a value state $\mathbf{v}_k$ at any time step k hits a constant mentioned in any of the atomic propositions, we discard $\mathbf{v}_k$ and start the simulation again from the value state of the previous time step.

Algorithm 3 Robust trajectory simulation

Input: Hybrid automaton $H = (Q, q_{in}, \{F_q(\mathbf{x})\}_{q \in Q}, \mathcal{G}, \rightarrow, \text{INIT})$, maximum time step K.
Output: Trajectory τ

1: Sample $\mathbf{v}_0$ from INIT uniformly. If $\mathbf{v}_0(i) \in C_i$ for any i, repeat.
2: Set $q_0 := q_{in}$ and $\tau := (q_0, \mathbf{v}_0)$.
3: **for** $k := 1 \dots K$ **do**
4: **repeat**
5: Generate time points $T := \{t_1, \dots, t_J\}$ uniformly in $(0, 1)$.
6: Simulate $\mathbf{v}^\ell := \Phi_{q_{k-1}}(t_\ell, \mathbf{v}_{k-1})$, for $\ell \in \{1, \dots, J\}$
7: Let $\widehat{\mathbb{T}}_j := \{t \in T : \mathbf{v}^\ell \in g_j\}$ be the time points where g_j is enabled.
8: Pick g_ℓ randomly according to probabilities $p_j := \frac{|\widehat{\mathbb{T}}_j|}{\sum_{j'=1}^{m} |\widehat{\mathbb{T}}_{j'}|}$.
9: Pick t_ℓ uniformly at random from $\widehat{\mathbb{T}}_\ell$.
10: Simulate $\mathbf{v}' := \Phi_{q'}(1 - t_\ell, \mathbf{v}^\ell)$, where q' is the target of g_ℓ.
11: **until** $\mathbf{v}'(i) \notin C_i$ for any i
12: Set $q_k := q'$, $\mathbf{v}_k := \mathbf{v}'$, and extend $\tau := (q_0, \mathbf{v}_0) \dots (q_k, \mathbf{v}_k)$.
13: **end for**
14: **return** τ

To see that the algorithm terminates with probability 1, note that if $\mathbf{v}_0 \in \mathfrak{h}$ and $\mathfrak{h}(i) = \{c\}$ for some $c \in C_i$ then $\mu(\mathfrak{h}) = 0$. Thus Step 1 repeats with probability 0. As a result with probability 1 it will be repeated only a finite number of times. Similarly the repeat loop of Step 4-11 will terminate with probability 1.

A.2 PERFORMANCE OF THE HYBRID SYSTEM SAMPLING ALGORITHM

We measured the average runtime for simulating a single trajectory (see Algorithm 1) for the room heating system and the cardiac cell system under varying values of Δ and J based on the MATLAB implementa-

tion. Figure 23 (a) shows that the runtime scales linearly with $1/\Delta$, the number of time steps within a unit time. The relationship of simulation time as a function of J is also empirically linear, shown in Figure 23 (b).

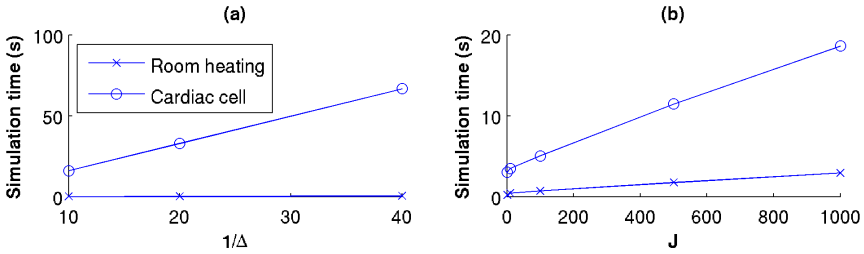


Figure 23: The relationship of simulation time with choice of Δ and J