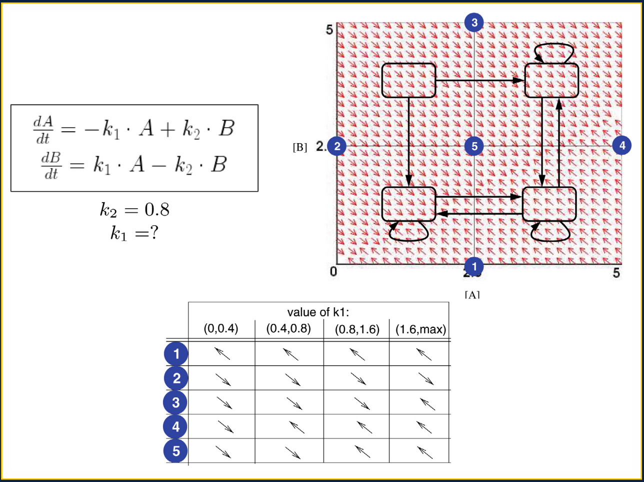
Tutorial

LNCS 7938

Marco Bernardo
Erik de Vink
Alessandra Di Pierro
Herbert Wiklicky (Eds.)

Formal Methods for Dynamical Systems

13th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2013
Bertinoro, Italy, June 2013, Advanced Lectures



$$\frac{dA}{dt} = -k_1 \cdot A + k_2 \cdot B$$
$$\frac{dB}{dt} = k_1 \cdot A - k_2 \cdot B$$

$$k_2 = 0.8$$
$$k_1 = ?$$

	value of k1:			
	(0,0.4)	(0.4,0.8)	(0.8,1.6)	(1.6,max)
1	↖	↖	↖	↖
2	↘	↘	↘	↘
3	↘	↘	↘	↖
4	↘	↖	↖	↖
5	↘	↘	↖	↖

Springer

Lecture Notes in Computer Science 7938

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Marco Bernardo Erik de Vink
Alessandra Di Pierro Herbert Wiklicky (Eds.)

Formal Methods for Dynamical Systems

13th International School
on Formal Methods for the Design of Computer,
Communication, and Software Systems, SFM 2013
Bertinoro, Italy, June 17-22, 2013
Advanced Lectures

Springer

Volume Editors
Marco Bernardo
Università di Urbino "Carlo Bo", Dipartimento di Scienze di Base e Fondamenti
61029 Urbino, Italy
E-mail: marco.bernardo@uniurb.it

Erik de Vink
Technische Universiteit Eindhoven, Dept. of Mathematics and Computer Science
5612 AZ Eindhoven, The Netherlands
E-mail: evink@win.tue.nl

Alessandra Di Pierro
Università di Verona, Dipartimento di Informatica
37134 Verona, Italy
E-mail: alessandra.dipierro@univr.it

Herbert Wiklicky
Imperial College London, Department of Computing
London SW7 2BZ, UK
E-mail: herbert@doc.ic.ac.uk

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-38873-6 e-ISBN 978-3-642-38874-3
DOI 10.1007/978-3-642-38874-3
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2013940207

CR Subject Classification (1998): D.2, F.3, F.1, G.1, D.3, F.4, E.4

LNCS Sublibrary: SL 2 – Programming and Software Engineering

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

This volume presents a set of papers accompanying the lectures of the 13th International School on Formal Methods for the Design of Computer, Communication, and Software Systems (SFM). This series of schools addresses the use of formal methods in computer science as a prominent approach to the rigorous design of the above-mentioned systems. The main aim of the SFM series is to offer a good spectrum of current research in foundations as well as applications of formal methods, which can be of help for graduate students and young researchers who intend to approach the field. SFM 2013 was devoted to dynamical systems and covered several topics including chaotic dynamics, information theory, systems biology, hybrid systems, quantum computing, and automata-based models and model checking.

The five papers collected in this volume represent the broad range of topics of the school. The paper by Köpf and Rybalchenko addresses the automation of the analysis of quantitative information-theoretic confidentiality properties through approximation and randomization techniques. Gratie, Iancu, and Petre introduce some of the basics of modeling with ODEs in biology by focussing on computational, numerical techniques for reaction-based models. The paper by Brim, Češka, and Šafránek presents a selection of approaches used for modeling biological systems and formalizing their interesting properties in temporal logics, together with high-performance model-checking techniques. Bortolussi and Hillston describe recent work on the use of fluid approximation techniques in the context of stochastic model checking for population models in which a large number of individual agents interact. Finally, Pachos's paper is an introduction to topological quantum computation.

We believe that this book offers a useful view of what has been done and what is going on worldwide in the field of formal methods for dynamical systems. We wish to thank all the speakers and all the participants for a lively and fruitful school. We also wish to thank the entire staff of the University Residential Center of Bertinoro for the organizational and administrative support.

June 2013

Marco Bernardo
Erik de Vink
Alessandra Di Pierro
Herbert Wiklicky

Table of Contents

Automation of Quantitative Information-Flow Analysis

Boris Köpf[1] and Andrey Rybalchenko[2]

[1] IMDEA Software Institute, Spain
[2] Technische Universität München, Germany
boris.koepf@imdea.org,
rybal@in.tum.de

Abstract. Quantitative information-flow analysis (QIF) is an emerging technique for establishing information-theoretic confidentiality properties. Automation of QIF is an important step towards ensuring its practical applicability, since manual reasoning about program security has been shown to be a tedious and expensive task. In this chapter we describe a approximation and randomization techniques to bear on the challenge of sufficiently precise, yet efficient computation of quantitative information flow properties.

1 Introduction

The goal of an information-flow analysis is to keep track of sensitive information during computation. If a program does not expose any information about its secret inputs to unauthorized parties, it has secure information flow, a property that is often formalized as noninterference. In many cases, achieving noninterference is expensive, impossible, or simply unnecessary: Many systems remain secure as long as the amount of exposed secret information is sufficiently small. Consider for example a password checker. A failed login attempt reveals some information about the secret password. However, for well-chosen passwords, the amount of leaked information is so small that a failed login-attempt will not compromise the security of the system.

Quantitative information-flow analysis (QIF) is a technique for establishing bounds on the information that is leaked by a program. The insights that QIF provides go beyond the binary output of Boolean approaches, such as non-interference analyzers. This makes QIF an attractive tool to support gradual development processes, even without explicitly specified policies. Furthermore, because information-theory forms the foundation of QIF, the quantities that QIF delivers can be directly associated with operational security guarantees, such as lower bounds for the expected effort of uncovering secrets by exhaustive search.

Automation of QIF is an important step towards ensuring its practical applicability, since manual reasoning about program security has been shown to be a tedious and expensive task [45]. Technically, successful automation of QIF must determine tight, yet efficiently computable bounds on the information-theoretic characteristics of the program. For deterministic programs with uniformly distributed inputs, these characteristics can be expressed by a partition of the secret program inputs. In this partition, each block is defined by the preimage of some program output. The computation of some information-theoretic characteristics, e.g., Shannon entropy, from this partition

M. Bernardo et al. (Eds.): SFM 2013, LNCS 7938, pp. 1–28, 2013.

requires the enumeration of all blocks and the estimation of their respective sizes. Other measures, e.g., min-entropy, only depend on the number of blocks in the partition. Exact computation for both kinds of characteristics is prohibitively hard, thus suggesting the exploration of approximation-based approaches. In the presence of approximation, characterizing the deviation from the exact result becomes an important question.

In this chapter, we describe approximation and randomization techniques to tackle the challenge of automating QIF for deterministic programs. The presented approach avoids the trap of block enumeration by a sampling method that uses the program itself to randomly choose blocks with probabilities corresponding to their relative sizes. Each sample amounts to a program execution and indexes a block by the corresponding program output. We obtain an under-approximation for each block using symbolic execution and symbolic backward propagation along the sequence of program statements traversed during a sample run. We obtain an over-approximation of each block in two steps. First, we transform the given program such that the input state becomes explicitly available in the set of reachable program states by memorizing it in an auxiliary variable. Second, an over-approximation of the reachable states of the transformed program that we obtain by applying abstract interpretation [24] delivers an over-approximation of blocks indexed by program outputs. Finally, we use the indexing by program outputs to put together under- and over-approximations for each sampled block. Thus, we obtain the necessary ingredients for the computation of information-theoretic guarantees, namely, lower and upper bounds for the remaining uncertainty about the program input.

The distinguishing feature of the presented technique is that it ensures fast convergence of the sampling process and provides formal guarantees for the quality of the obtained bounds. The proof builds upon a result by Batu et al. [9] stating that the entropy of a random variable can be estimated accurately and with a high confidence level using only a small number of random samples. Since Batu et al. [9] require an oracle revealing the probability of each sample of the random variable, in the first step towards the QIF setting we identify a correspondence between the sampling oracle and the preimage computation for the program whose QIF properties are analyzed. In the second step, we prove that the confidence levels and convergence speed of the exact setting, which relies on the actual preimages, can also be obtained in the approximative setting where only over- and under-approximations of preimages are known. This result allows our approach to replace the exhaustive analysis of all, say n-many, possible preimages with the treatment of a randomly chosen set of $O((\log n)^2)$ preimage samples.

This chapter extends and generalizes the results from [41]. In particular, the chapter contains a discussion of non-uniformly distributed secrets and adversarially chosen inputs.

Outline. In the next section, we illustrate our approach on example programs. Then, we give basic definitions in Section 3. In Section 4, we present over- and under-approximation techniques. Section 5 describes how randomization combines over- and under-approximations to deliver a quantitative information flow analysis. Finally, we discuss related work in Section 6.

2 Illustration

We illustrate our method on two example programs whose quantitative information-flow analysis is currently out of reach for the existing automatic approaches. The examples present the computation of Shannon entropy and min-entropy, respectively, and require dealing with loops and data structures, which our method handles automatically using approximation and randomization techniques. We computed certain intermediate assertions for the following examples using BoundGen, an automatic tool for the discovery of resource usage bounds [23].

2.1 Estimating Shannon Entropy

As a first example we consider the electronic purse program from [2] as shown below.

```
l = 0;
// assume(h < 20);
while(h>=5){
  h = h-5;
  l = l+1;
}
```

The program receives as input the nonnegative balance of a bank account in the integer variable h and debits a fixed amount of 5 from this account until the balance is insufficient for this transaction, i.e., until h < 5.

Our goal is to determine the remaining uncertainty about the initial value of h after learning the final value of l, where we use Shannon entropy as a measure of uncertainty. A high remaining uncertainty implies a large lower bound on the expected number of steps that are required for determining the initial value of h by brute-force search [47], which corresponds to a formal, quantitative confidentiality guarantee.

For uniformly distributed inputs, one can express the remaining Shannon entropy as a weighted sum of the logarithms of the sizes of the preimages of the program [38]. A large average preimage size corresponds to a large remaining uncertainty about the input and implies a large expected effort for determining the program's input after observing the output.

One way to precisely compute the remaining Shannon entropy is to compute the partition induced by the preimages of the program, which may require the enumeration of all pairs of program paths. Moreover, it requires the enumeration of all blocks in the computed partition [2]. Both enumeration problems severely limit the size of the systems that can be analyzed using this precise approach.

For the purse program and nonnegative h, the partition induced by the preimages of the program is

$$\{\{0, 1, 2, 3, 4\}, \{5, 6, 7, 8, 9\}, \dots\}$$

and is represented by the following formula that states that two initial values, say h and $\overline{h}$, are in the same block.

$$\exists k \geq 0 : 5k \leq h \leq 5k + 4 \wedge 5k \leq \overline{h} \leq 5k + 4$$

The quantified variable k corresponds to the number of loop iterations that the program executes when started on h and $\overline{h}$. Such existentially quantified assertions are out of reach for the existing automatic tools for reasoning about programs. The current approaches simplify the problem by (implicitly) bounding the number of loop iterations and replacing the existential quantification with finite disjunction.

For example, the algorithm from [2] bounds the number of loop iterations by making finite the set of possible valuations of h, which is achieved by introducing the assumption in line 2. With this restriction, we obtain the following characterization of the partition.

$$\begin{aligned}
&(0 \le h \le 4 \wedge 0 \le \overline{h} \le 4) \vee \\
&(5 \le h \le 9 \wedge 5 \le \overline{h} \le 9) \vee \\
&(10 \le h \le 14 \wedge 10 \le \overline{h} \le 14) \vee \\
&(15 \le h \le 19 \wedge 15 \le \overline{h} \le 19)
\end{aligned}$$

As mentioned above, such solutions are only partial and overly restrictive, since the program properties derived for values below the loop bound do not necessarily carry over beyond this limit.

In this chapter, we show that the precise computation of each preimage can be replaced by the computation of the under- and over-approximation of the preimage. We also show that, by running the program on randomly chosen inputs and approximating only the preimages of the corresponding outputs, one can derive upper and lower bounds for the remaining uncertainty about the program's secret input. These bounds are valid with a provably high level of confidence, for a number of samples that is poly-logarithmic in the size of the input domain.

Approximation. To compute *over-approximations* of preimages, we augment the program by declaring copies of input variables (so-called *ghost variables*) and adding an assignment from inputs into these copies as the first program statement. In our program, we declare a new variable _h and insert the initialization statement `_h = h;` before line 1. Let $\overline{F_{reach}}$ be the set of reachable final states of the augmented program. $\overline{F_{reach}}$ keeps track of the relation between the ghost variables and the output of the original program. As the ghost variables are not modified during computation, this set corresponds to the input-output relation ρ_{IO} of the original program, i.e., $\overline{F_{reach}} = \rho_{IO}$. While the exact computation of $\overline{F_{reach}}$ is a difficult task, we can rely on abstract interpretation techniques for computing its over-approximation [24]. In particular, we can bias the over-approximation towards the discovery of the relation between the ghost variables and low outputs by using constraints and borrow existing implementations originally targeted for resource bound estimation, e.g. [23, 32]. We apply the bound generator BoundGen [23] and obtain

$$\overline{F_{reach}}^{\sharp} \;=\; -5 + 5\,\mathtt{l} \le \mathtt{\_h} \le -1 + 5\,\mathtt{l}\;.$$

The predicate $\overline{F_{reach}}^{\sharp}$ represents an over-approximation of the input-output relation ρ_{IO}. (Here, the outcome happens to be a precise description.) Hence for each low output value l, the set of ghost input values _h, such that l and _h are related by $\overline{F_{reach}}^{\sharp}$, over-approximates the preimage of l with respect to the original program. The size of these approximated preimages can be determined using tools for counting models, e.g., we use LattE [43] for dealing with linear arithmetic assertions. In our example, we obtain 5 as an upper bound for the size of the preimage of each value of l.

To compute *under-approximations* of preimages, we symbolically execute the program on a randomly chosen input and determine the preimage with respect to the obtained path through the program. This computation relies on the relation between program inputs and outputs along the path that the execution takes. We establish this relation by combining the transition relations of individual steps. This technique can be efficiently automated using existing symbolic execution engines both at the source code level, e.g., KLEE [14] and DART [29], and at the binary level, e.g., BitScope [12].

For example, for an input of $\mathtt{h} = 37$, this relation is determined to be

$$35 \le \mathtt{h} \le 39 \wedge \mathtt{l} = 8\ .$$

(Again, the result happens to be a precise description.) Hence, the preimage size of $\mathtt{l} = 8$ is at least 5.

Randomization. The direct approximation of the leak as described above requires the computation of bounds for the size of each preimage. Note that the number of preimages can be as large as the input domain (e.g. when the program's output fully determines the input), which severely limits scalability. We overcome this limitation using a randomized approach, where we run the program on randomly chosen inputs and approximate only the preimages of the corresponding outputs. Leveraging a result from [9], we show how this set of preimages can be used for deriving bounds on the information-theoretic characteristics of the entire program. These bounds are valid with a provably high level of confidence, for a number of samples that is logarithmic in the size of the input domain.

Technically, we show that, for an arbitrary $\delta > 0$, the remaining uncertainty H about the secret input is bounded by the following expression

$$\frac{1}{n}\sum_{i=1}^{n} \log_2 m_i^{\flat} - \delta \;\le\; H \;\le\; \frac{1}{n}\sum_{i=1}^{n} \log_2 m_i^{\sharp} + \delta\ ,$$

where n is the number of samples and $m_i^{\flat}$ and $m_i^{\sharp}$ are the upper and lower bounds for the size of the preimage corresponding to the ith sample. If the secret input is chosen from a set I, these bounds hold with a probability of more than p for a number of samples n such that

$$n = \frac{(\log_2 \#(I))^2}{(1-p)\delta^2}\ .$$

For our example and $I = \{0, \ldots, 2^{64} - 1\}$, our analysis delivers coinciding lower and upper bounds of 5 for the sizes of the preimages (except for the preimage containing $2^{64} - 1$, which is smaller). As a consequence, we obtain entropy bounds of

$$\log_2 5 - 0.1 \le H \le \log_2 5 + 0.1$$

that hold with a probability of more than 0.99 when considering 10^8 samples.

2.2 Estimating Min-entropy

The following program implements an algorithm for the bit-serial modular exponentiation of integers. More precisely, the program computes $\mathtt{x}^{\mathtt{k}}$ mod `n`, where `n` is the constant modulus and `k` is maintained by the program in a binary array of constant length `len`.

```
1    int m = 1;
2    for (int i = 0; i<len; i++) {
3      m = m*m mod n;
4      if ( k[i] == 1 ) {
5        m = m*x mod n;
6      }
7    }
```

Due to the conditional execution of the multiplication operation `m=m*x mod n` in line 5, the running time of this program reveals information about the entries of the array `k`. Such variations have been exploited to recover secret keys from RSA decryption algorithms based on structurally similar modular exponentiation algorithms [37]. We analyze an abstract model of the timing behavior of this program, where we assume that each multiplication operation consumes one time unit. We make this model explicit in the program semantics by introducing a counter `time` that is initialized with 0 and is incremented each time a multiplication operation takes place.

Our goal is to quantify the remaining min-entropy about the content of `k` after observing the program's execution time, i.e., given the final value of `time`. The min-entropy captures the probability of correctly guessing a secret at the first attempt. In contrast to Shannon entropy, computation of min-entropy does not require the enumeration of blocks and estimation of their sizes. For min-entropy, we only need to estimate how many blocks (or alternatively how many possible outputs) the program can produce [59].

This simplification can be exploited when dealing with programs that manipulate data structures. As the example shows, applications often keep secret the content of data structures, while some of their properties, e.g., list length or even number of elements satisfying a Boolean query, are revealed as outputs. In such cases, our method can estimate the input's remaining min-entropy despite the difficulties of automatic reasoning about data structures. To succeed, our method applies the over- and under-approximation techniques presented in Section 2.1, however without the addition of ghost variables. No ghost variables are needed, since the actual block content given by the secret data structure elements is irrelevant for the min-entropy computation.

We extend a result from [59] to show that under-approximations of the remaining min-entropy can be computed from over-approximations of the size of the set of reachable states F_{reach}, which in our example is the set of possible values of the variable `time`. By applying the bound generator BoundGen [23], we obtain the following over-approximation ${F_{reach}}^\sharp$ of F_{reach},

$$ {F_{reach}}^\sharp \;\equiv\; \mathtt{len} \le \mathtt{time} \le 2\mathtt{len}\,, $$

which shows that $\#(F_{reach}{}^{\sharp}) \leq \texttt{len} + 1$. We use this bound to infer that, after observing `time`, the remaining uncertainty about the exponent `k` is still larger than

$$\log_2 \frac{2^{\texttt{len}}}{\texttt{len} + 1} = \texttt{len} - \log_2(\texttt{len} + 1)\,,$$

given that `k` is drawn uniformly from $\{0, \ldots, 2^{\texttt{len}} - 1\}$. An alternative interpretation is that the expected reduction in uncertainty about the exponent `k` is at most $\log_2(\texttt{len} + 1)$ bits.

3 Preliminaries

In this section, we give the necessary definitions for dealing with programs and information-flow analysis.

3.1 Programs and Computations

Following [46] we treat programs as transition systems, and rely on existing translation procedures from programs written in particular programming languages to transition systems, e.g., [36]. A program $P = (S, I, \mathcal{T})$ consists of the following components.

- S - a set of *states*.
- $I \subseteq S$ - a set of *initial* states.
- $\mathcal{T}$ - a finite set of *transitions* such that each transition $\tau \in \mathcal{T}$ is given a binary *transition relation* over states, i.e., $\rho_\tau \subseteq S \times S$.

For a program represented as source code, states are determined by the valuation of the declared program variables and the program counter, and transition correspond to program statements.

Let F be the set of *final* program states that do not have any successors, i.e.,

$$F \;=\; \{s \in S \mid \forall s' \in S\ \forall \tau \in \mathcal{T} : (s, s') \notin \rho_\tau\}\,.$$

A *computation* of P is a sequence of program states $s_1, \ldots, s_n$ such that s_1 is an initial state, i.e., $s_1 \in I$, s_n is a final state, i.e., $s_n \in F$, and each pair s and s' of consecutive states follows a program transition, i.e., $(s, s') \in \rho_\tau$ for some $\tau \in \mathcal{T}$. We assume that final states do not have any successors, i.e., there is no pair of states s and s' such that $s \in F$ and $(s, s') \in \rho_\tau$ for some $\tau \in \mathcal{T}$.

A program is *deterministic* if for each state s there is at most one transition that assigns a successor to s and there is at most one such successor. Formally,

$$\forall s\ \forall s'\ \forall s''\ \forall \tau\ \forall \tau' : ((s, s') \in \rho_\tau \land (s, s'') \in \rho_{\tau'}) \rightarrow (s' = s'' \land \tau = \tau')\,.$$

In this chapter we only consider deterministic programs.

A *path* is a sequence of transitions. We write ϵ for the *empty* path, i.e., the path of length zero. Let $\circ$ be the *relational composition* function for binary relations, i.e., for binary relations X and Y we have $X \circ Y \;=\; \{(x, y) \mid \exists z : (x, z) \in X \land (z, y) \in Y\}$. Then, a *path relation* is a relational composition of transition relations along the path, i.e., for $\pi = \tau_1, \ldots, \tau_n$ we have $\rho_\pi = \rho_{\tau_1} \circ \ldots \circ \rho_{\tau_n}$. A path π is *feasible* if its path relation is not empty, i.e., $\rho_\pi \neq \emptyset$.

Let ρ be the *program transition relation* defined as follows.

$$\rho \quad = \quad \bigcup_{\tau \in \mathcal{T}} \rho_\tau$$

We write ρ^* for the transitive closure of ρ. The *input-output* relation ρ_{IO} of the program relates each initial state s with the corresponding final states, i.e.,

$$\rho_{IO} \quad = \quad \rho^* \cap (I \times F) \,.$$

A final state s' is *reachable* from an initial state s if $(s, s') \in \rho_{IO}$. We write F_{reach} for the set of reachable final states, i.e.

$$F_{reach} \quad = \quad \{s' \in F \mid \exists s \in I : \ (s, s') \in \rho_{IO}\}$$

Given a final state $s' \in F$, we define its *preimage* $P^{-1}(s')$ to be the set of all initial states from which s' is reachable, i.e.,

$$P^{-1}(s') \quad = \quad \{s \mid (s, s') \in \rho_{IO}\} \,.$$

The preimage of an unreachable state is the empty set.

3.2 Qualitative Information Flow: What Leaks?

We characterize partial knowledge about the elements of a set A in terms of *partitions* of A, i.e., in terms of a family $\{B_1, \ldots, B_r\}$ of pairwise disjoint *blocks* such that $B_1 \uplus \cdots \uplus B_r = A$. A partition of A models that each $a \in A$ is known up to its enclosing block B_i such that $a \in B_i$. We compare partitions using the imprecision order $\sqsubseteq$ defined by

$$\{B_1, \ldots, B_r\} \sqsubseteq \{B'_1, \ldots, B'_{r'}\} \quad = \quad \forall i \in \{1, \ldots, r\} \ \exists j \in \{1, \ldots, r'\} : B_i \subseteq B'_j \,.$$

With the imprecision order, larger elements correspond to less knowledge about the elements of A. Let $\sqsubset$ be the irreflexive part of $\sqsubseteq$.

Knowledge about Initial States. We consider a deterministic program P that implements a total function, i.e., for each input state s there is one final state s' such that $(s, s') \in \rho_{IO}$. We assume that the initial state of each computation is secret. Furthermore, we assume an attacker that knows the program, in particular its transition relation, and the final state of each computation. In our model, the attacker does not know any intermediate states of the computation.

The knowledge gained by the attacker about initial states of computations of the program P by observing their final states is given by the partition Π that consists of the preimages of reachable final states, i.e.,

$$\Pi \quad = \quad \{P^{-1}(s') \mid s' \in F_{reach}\} \,. \tag{1}$$

There are two extreme cases. The partition $\Pi = \{I\}$ consisting of a single block captures that P reveals no information about its input. In contrast, the partition $\Pi = \{\{s\} \mid s \in I\}$ where each block consists of a single element captures the case that P fully discloses its input. For the remaining, intermediate cases such that $\{\{s\} \mid s \in I\} \sqsubset \Pi \sqsubset \{I\}$, the partition Π captures that P leaks partial information about its input.

Knowledge Refinement by Interaction. More generally, a program P may receive and produce both secret (*high*) and public (*low*) inputs, where the low inputs may be read or modified by an attacker. If the high input remains fixed over several runs of the program, the attacker can use the low inputs to influence the computation and thereby refine his knowledge about the high input. A restricted version of the above scenario can be reduced to the setting of programs with only high inputs and can be analyzed using the methods presented in this chapter.

Specifically, we consider the case where the attacker runs the program using a fixed finite set of low inputs $l_1, \ldots, l_n$. We model this scenario using a finite set of programs $P_1, \ldots, P_n$ where each P_i corresponds to the program P with the low input set to the value l_i. An attacker running the program P with two low inputs l_i and l_j and observing the final states s_i and s_j, respectively, can hence narrow down the set of possible secrets to $P_i^{-1}(s_i) \cap P_j^{-1}(s_j)$. More generally, the partition

$$\Pi \;=\; \{P_1^{-1}(s_1) \cap \cdots \cap P_n^{-1}(s_n) \mid s_1, \ldots, s_n \in F\} \tag{2}$$

characterizes what an attacker can learn about a fixed secret input after running P with the low inputs $l_1, \ldots, l_n$ and observing the corresponding outputs, see e.g. [38].

Notice that the partition Π corresponds to the set of preimages of the function $f\colon I \to F^n$ where each component f_i is defined by the input-output behavior of program P_i. This function f can be computed by the independent composition of the programs $P_1, \ldots, P_n$, see e.g. [7]. For example, for the case $n = 2$ we obtain an independent composition by creating a copy P' of P and replacing every program variable x that occurs in P by a fresh variable x'. An analysis of the following program with input h' and output (l, l') then corresponds to an analysis of P with respect to two runs with high input h and low inputs l_1 and l_2, respectively.

```
$l = l_1$; $l' = l_2$; $h' = h$
$P(h, l)$
$P'(h', l')$
return $(l, l')$
```

This construction easily generalizes to $n > 2$, however at the expense of an exponentially growing state-space. In this way, analyzing a deterministic program with respect to a fixed finite set $l_1, \ldots, l_n$ of low inputs can be reduced to the analysis of a program without low inputs. For simplicity of exposition we will focus on programs without low inputs in the remainder of the chapter.

3.3 Quantitative Information Flow: How Much Leaks?

In the following, we use information theory to characterize the information that P reveals about its input. This characterization has the advantage of being compact and easy to compare. Moreover, it yields concise interpretations in terms of the effort needed to determine P's input from the revealed information.

We assume that the program's initial state is drawn according to a probability distribution p on I and we suppose that p is known to the attacker. For a random variable $X\colon I \to \mathcal{X}$ with range $\mathcal{X}$, we define $p_X\colon \mathcal{X} \to \mathbb{R}$ as $p_X(x) = \sum_{s \in X^{-1}(x)} p(s)$, which we will also denote by $\Pr(X = x)$. For analyzing the program P, there are two random variables of particular interest. The first random variable $U\colon I \to I$ models the random choice of an input in I, i.e., $U(s) = s$. The second random variable $V\colon I \to F$ captures the input-output behavior of P, i.e., $V(s) = s'$ where $(s, s') \in \rho_{IO}$.

Shannon Entropy. The *(Shannon) entropy* [58] of a random variable $X\colon I \to \mathcal{X}$ is defined as

$$H(X) = -\sum_{x \in \mathcal{X}} p_X(x) \log_2 p_X(x) \, .$$

The entropy is a lower bound for the average number of bits required for representing the results of independent repetitions of the experiment associated with X. Thus, in terms of guessing, $H(X)$ is a lower bound for the average number of questions with binary outcome that need to be asked to determine X's value [13]. Given another random variable $Y\colon I \to \mathcal{Y}$, we write $H(X|Y = y)$ for the entropy of X given that the value of Y is y. The *conditional entropy* $H(X|Y)$ of X given Y is defined as the expected value of $H(X|Y = y)$ over all $y \in \mathcal{Y}$, namely

$$H(X|Y) = \sum_{y \in \mathcal{Y}} p_Y(y) H(X|Y = y) \, ,$$

and it captures the remaining entropy about X when Y is observed.

Consider now a program P and the corresponding random variables U and V, as defined above. Then $H(U)$ is the observer's initial uncertainty about the secret input and $H(U|V)$ is the observer's expected uncertainty about the input after running the program. We will use $H(U|V)$ as a measure of information flow because it is associated with operational security guarantees: The expected effort for determining the secret input by exhaustive search is bounded from below by $2^{H(U|V)-2}$, see [47] and [11,39].

Min-entropy. The min-entropy of a random variable X captures the probability of correctly determining the value of X in a single guess using an optimal strategy, i.e., by choosing the most likely value. From this probability, it is straightforward to derive bounds on the probability for correctly determining the value of X in an arbitrary number of guesses. Formally, the *min-entropy* H_∞ is defined as

$$H_\infty(X) = -\log_2 \left(\max_{x \in \mathcal{X}} p_X(x) \right) \, .$$

The *conditional min-entropy* $H_\infty(X|Y)$ quantifies the expected probability of correctly determining the secret in one guess after having observed the outcome of Y [59] and is defined by

$$H_\infty(X|Y) = -\log_2\left(\sum_{y\in Y} p_Y(y) \max_{x\in X} p_{X|Y=y}(x)\right).$$

As before, $H_\infty(U|V)$ quantifies the expected probability of correctly determining the value of U in one guess after observing the output V of the program P.

Note that the success probability of a single guess can also be estimated using the conditional Shannon entropy, e.g. using Fano's inequality. However, as pointed out in [59], this estimation is not always accurate. Hence, it is preferable to use min-entropy to compute the success probability of single guesses.

Leakage vs. Security. The information *leaked* by P is the reduction in uncertainty about the input U when the output V is observed. For the case of Shannon entropy, the leakage L is given by

$$L = H(U) - H(U|V)\,, \tag{3}$$

and it can be defined analogously using alternative measures of uncertainty. Many approaches in the literature focus on computing the leakage rather than the remaining uncertainty. If the initial uncertainty $H(U)$ is known, Equation (3) gives a direct correspondence between the leakage L and the remaining uncertainty $H(U|V)$. In the following, we focus on the remaining uncertainty rather than on the leakage, because the remaining uncertainty enjoys a more direct interpretation in terms of an attacker's difficulty for recovering secrets and, hence, security.

(Non-)Uniform Input Distributions. For the important case where p is the uniform distribution, we have

$$\Pr(V = s') = \frac{\#(P^{-1}(s'))}{\#(I)}\,, \tag{4}$$

i.e., one can characterize the distribution of V in terms of the sizes of the preimages of P. Moreover, one can give formulas for remaining uncertainty about the input of P in terms of the number and the sizes of the partition Π induced by the preimages of P, see [38, 59]. These formulas provide the interface between the qualitative and the quantitative viewpoints.

Proposition 1. *If the input of P is uniformly distributed, we obtain the following expressions for the remaining uncertainty about the input after observing the output of P in terms conditional Shannon and min-entropies, respectively.*

$$H(U|V) = \frac{1}{\#(I)} \sum_{B\in\Pi} \#(B) \log_2 \#(B) \tag{5}$$

$$H_\infty(U|V) = \log_2 \frac{\#(I)}{\#(\Pi)} \tag{6}$$

In scenarios where the input is distributed non-uniformly, Proposition 1 cannot be directly applied. However, a recent result shows that for Shannon entropy, the case of non-uniform input distributions can be reduced to the case of uniform input distributions [1]. The key idea behind the reduction is to consider the non-uniform input distribution p as being generated by a deterministic program D that receives uniformly distributed input.

The key requirement for this generator program is that, for each $s \in I$, the size of $D^{-1}(s)$ be proportional to $p(s)$. The following program satisfies this requirement by construction. Here, we assume that $I = \{s_1, \dots, s_{\#(I)}\}$ and that the variable r is initialized by values drawn uniformly from [0, 1]. For rational distributions, the program can be easily adapted to one that receives uniformly distributed input from an integer range. While the above construction is obviously not practical for large I, efficient generator programs often occur in practice. E.g., the Personal Identification Numbers (PINs) used in electronic banking are often not uniformly distributed, but derived from uniform bitstrings using decimalization techniques [21]. Another example are the keys of a public-key cryptosystem, which are typically not uniformly distributed bitstrings. However, they are produced by a key generation algorithm that operates on uniformly distributed input. More generally, a large number of randomized algorithms expect uniformly distributed randomness. For a language-based perspective on distribution generators, see [54].

```
$j := 1$; $c := p(s_1)$
while $j < \#(I) \wedge r > c$
    $j := j + 1$
    $c := c + p(s_j)$
return $s_j$
```

Given a generator program D, the remaining uncertainty about the inputs of a program P can be expressed as the difference in the remaining uncertainty about the inputs of P ; D and D. Modeling the uniformly distributed input by a random variable U and interpreting program D as a random variable, one obtains the following connection [1].

Proposition 2.

$$H(D|V) = H(U|V \circ D) - H(U|D)$$

Notice that (5) applies to both terms on the right hand side of Proposition 2 and completes the reduction to the uniform case. In the remainder of the chapter we hence focus on the uniform case. For a more detailed treatment of the non-uniform case, refer to [1]. Furthermore, we will only consider logarithms with base two, and will omit the base from the formulas to simplify notation.

3.4 Towards Automation

Proposition 1 immediately suggests the following approach for automatically determining the remaining uncertainty about the inputs of a program P.

1. For computing $H(U|V)$, first enumerate the elements of Π and then determine their sizes.
2. For computing $H_\infty(U|V)$, determine the number of elements in Π.

In [2] it was shown how the partition Π can be obtained by computing relational weakest preconditions, how its blocks can be enumerated using SAT solvers, and how the sizes of the blocks can be determined using model counting [30]. Unfortunately, the exact computation of these properties can be prohibitively expensive (see also [15, 60]).

4 Bounding Information Leaks

In this section, we present a method for the automatic derivation of upper and lower bounds on the remaining uncertainty about a program's inputs.

We consider bounds that over- and under-approximate the remaining uncertainty both qualitatively and quantitatively. On the qualitative side, we show how to compute over- and under-approximations of the set of blocks in Π. Moreover, we show how to compute over- and under-approximations for each block in Π. On the quantitative side, we show how these over- and under-approximations can be used for computing bounds on the remaining uncertainty in terms of min-entropy and Shannon entropy.

4.1 Bounding Block Count

Computation of min-entropy requires estimation of the number of blocks in the partition Π, which is equal to the cardinality of the set of reachable final states F_{reach}, see (1) and (6). The set F_{reach} can be over-approximated by applying abstract interpretation techniques [24] on the program P. Abstract interpretation allows one to incrementally compute an approximation of the set of reachable program states by relying on the approximation of individual program transitions. The set F_{reach} can be under-approximated by symbolic execution and backward propagation along the sequence of program statements traversed during the execution.

Over-approximation of F_{reach}. For the computation of the over-approximation ${F_{reach}}^\sharp \supseteq F_{reach}$ we will use an *abstraction* function $\alpha : 2^S \to 2^S$ that over-approximates a given set of states, i.e., for each $X \subseteq S$ we have $X \subseteq \alpha(X)$. For simplicity of exposition we assume that the abstract values are sets of program states, which leads to the concretization function that is the identity function and hence is omitted. The presented approach can use more sophisticated abstract domains without any modifications.

In theory, the set of reachable final states F_{reach} can be computed by iterating the one-step reachability function $post$: $2^{S\times S} \times 2^S \to 2^S$ defined below. Note that we put the first parameter in the subscript position to simplify notation.

$$post_\rho(X) \quad = \quad \{s' \mid \exists s \in X : (s, s') \in \rho\}$$

The iteration process applies $post_\rho$ on the set I zero, one, two, . . . , many times, takes the union of the results and restricts it to the final states. The resulting set is the intersection of the final states with the least fixpoint of $post_\rho$ containing I by the Kleene fixpoint theorem. Formally, we have

$$\begin{aligned} F_{reach} &= F \cap (I \cup post_\rho(I) \cup post_\rho(post_\rho(I)) \cup \dots) \\ &= F \cap lfp(post_\rho, I) \ . \end{aligned}$$

Unfortunately, it is not practical to compute the result of iterating $post_\rho$ arbitrarily many times, i.e., the iteration may diverge for programs over unbounded (infinite) sets of states.

Abstract interpretation overcomes the above fundamental problem of computing F_{reach} by resorting to an approximation of $post_\rho$ using the abstraction function α. That is, instead of iterating $post_\rho$ we will iterate its composition with α, i.e., we will compute the restriction of the abstract least fixpoint to the final states. Let $\bullet$ be a binary *function composition* operator such that $f \bullet g = \lambda x. f(g(x))$. Then

$$\begin{aligned} {F_{reach}}^\sharp &= F \cap (\alpha(I) \cup (\alpha \bullet post_\rho)(\alpha(I)) \cup (\alpha \bullet post_\rho)^2(\alpha(I)) \cup \dots) \\ &= F \cap lfp(\alpha \bullet post_\rho, \alpha(I)) \ . \end{aligned}$$

Instead of computing the exact set F_{reach}, we compute its over-approximation, i.e.,

$$F_{reach} \subseteq {F_{reach}}^\sharp \ .$$

By choosing an adequate abstraction function α we can enforce termination of the iteration process after a finite number of steps, as no new states will be discovered. In other words, after some $k \geq 0$ steps we obtain

$$(\alpha \bullet post_\rho)^{k+1}(\alpha(I))) \subseteq \bigcup_{i=0}^{k} (\alpha \bullet post_\rho)^k(\alpha(I)) \ ,$$

while maintaining the desired precision of the over-approximation. Here, we can rely on an extensive body of research in abstract interpretation and efficient implementations of abstract domains, including octagons and polyhedra [3,25,35,50].

Under-Approximation of F_{reach}. When computing the under-approximation ${F_{reach}}^\flat \subseteq F_{reach}$ we follow an approach that is different from its over-approximating counterpart, since we cannot rely on abstract interpretation for computing an under-approximation. Despite the established theoretical foundation, abstract interpretation does not yet provide practical under-approximating abstractions.

Instead, we can resort to a practical approach for computing under-approximations by symbolic execution of the program along a selected set of program paths. This approach has been successful for efficient exploration of state spaces (for finding runtime safety violations), e.g., VERISOFT, KLEE, DART, and BITSCOPE [12,14,28,29].

Let $\pi_1, \dots, \pi_n \in \mathcal{T}^+$ be a finite set of non-empty program paths, which can be chosen randomly or according to particular program coverage criteria. This set of paths determines a subset of reachable finite states in the following way.

$${F_{reach}}^\flat \quad = \quad \bigcup_{\pi \in \{\pi_1, \dots, \pi_n\}} \{s' \mid \exists s \in S : (s, s') \in \rho_\pi \cap (I \times F)\}$$

```
procedure MkPath
input
    $s \in I$ - initial state
begin
    $\pi := \epsilon$
    while $s \notin F$ do
        $(\tau, s') :=$ choose $\tau \in \mathcal{T}$ such that $(s, s') \in \rho_\tau$
        $s := s'$
        $\pi := \pi \cdot \tau$
    done
    return $(\pi, s)$
end.
```

Fig. 1. Function MkPath computes the program path and the final state for a given initial state

In Figure 1 we describe a possible implementation of a symbolic execution function MkPath that creates a program path for a given initial state. The termination of MkPath follows from the requirement that P implements a total function.

Given the over- and under-approximations $F_{reach}{}^{\sharp}$ and $F_{reach}{}^{\flat}$, we can bound the number of blocks in the partition Π as formalized in the following theorem.

Theorem 1. *The over- and under-approximations of the set of reachable final states yield over- and under-approximations of the number of blocks in the partition Π. Formally,*

$$\#(F_{reach}{}^{\flat}) \leq \#(\Pi) \leq \#(F_{reach}{}^{\sharp}) \; .$$

Proof. The theorem statement is a direct consequence of the bijection between F_{reach} and Π under the inclusion $F_{reach}{}^{\flat} \subseteq F_{reach} \subseteq F_{reach}{}^{\sharp}$.

4.2 Bounding Block Sizes

Computation of block sizes in Π requires identification of the blocks as sets of initial states. Our technique approaches the computation of Π through an intermediate step that relies on the input-output relation ρ_{IO}. We formulate the computation of the input-output relation as the problem of computing sets of reachable states, which immediately allows one to use tools and techniques of abstract interpretation and symbolic execution as presented above. Then, given ρ_{IO}, we compute Π following (1).

In order to compute the input-output relation ρ_{IO} we augment the program P such that the set of reachable states keeps track of the initial states. We construct an augmented program $\overline{P} = (\overline{S}, \overline{I}, \overline{F}, \overline{\mathcal{T}})$ from the program P as follows.

- $\overline{S} = S \times S$.
- $\overline{I} = \{(s, s) \mid s \in I\}$.
- $\overline{F} = S \times F$.

- $\overline{\mathcal{T}} = \{\overline{\tau} \mid \tau \in \mathcal{T}\}$, where for each transition $\overline{\tau} \in \overline{\mathcal{T}}$ we construct the transition relation $\rho_{\overline{\tau}}$ such that

$$\rho_{\overline{\tau}} \;=\; \{((s'', s), (s'', s')) \mid (s, s') \in \rho_\tau \wedge s'' \in S\}\,.$$

Similarly to P, we define $\overline{\rho} \;=\; \bigcup_{\overline{\tau} \in \overline{\mathcal{T}}} \rho_{\overline{\tau}}$.

Our augmentation procedure is inspired by the use of ghost variables for program verification, see e.g. [5]. Note that when constructing $\overline{P}$ we *do not* apply the self-composition approach [7], and hence we avoid the introduction of additional complexity to P. In fact, the construction of $\overline{P}$ from P can be implemented as a source-to-source transformation by declaring copies of input variables and adding an assignment from inputs into these copies as the first program statement.

The set of reachable states of the augmented program $\overline{P}$ corresponds to the input-output relation of the program P, as stated by the following theorem.

Theorem 2 (Augmentation). *The input-output relation of P is equal to the set of reachable final states of its augmented version $\overline{P}$, i.e.,*

$$\rho_{IO} = \overline{F_{reach}}\,.$$

Proof. The augmented program manipulates pairs of states of the original program. We observe that the augmented program does not modify the first component of its initial state, which stays equal to the initial value. Furthermore, the second component follows the transition relation of P. Thus, the theorem statement follows directly.

Now we apply abstract interpretation and symbolic execution techniques from Section 4.1 to the augmented program $\overline{P}$. We obtain the over-approximation $\overline{F_{reach}}^\sharp$ by abstract least fixpoint computation of $post_{\overline{\rho}}$, where α over-approximates sets of $\overline{S}$-states, and its restriction to the final states of $\overline{P}$, i.e.,

$$\overline{F_{reach}}^\sharp \;=\; \overline{F} \cap \mathit{lfp}(\alpha \bullet post_{\overline{\rho}},\; \alpha(\overline{I}))\,.$$

The computation of the under-approximation $\overline{F_{reach}}^\flat$ requires a finite set of paths $\pi_1, \ldots, \pi_n$ through the augmented program, however we could also use a set of paths through P and adjust accordingly. Again we use the paths for performing symbolic execution and applying existential quantification over the initial states to obtain $\overline{F_{reach}}^\flat$.

We finally put together over- and under-approximations of preimages of P, indexed by the corresponding final states.

Theorem 3 (Augmented Approximation). *Projection of the over- and under-approximations of reachable final states of the augmented program on the initial component over- and under-approximates respective blocks in the partition Π for the program P. Formally, for each $s' \in F$ we have*

$$\{s \mid (s, s') \in \overline{F_{reach}}^\flat\} \subseteq P^{-1}(s') \subseteq \{s \mid (s, s') \in \overline{F_{reach}}^\sharp\}\,.$$

Thus, given a reachable final state of P we can apply Theorem 3 to compute an over- and under-approximation of the corresponding block in the partition Π.

4.3 Information-Theoretic Bounds

We now show how, for uniformly distributed input, bounds on the size and the number of the elements of Π can be used for deriving bounds on the remaining uncertainty of a program in terms of Shannon entropy and min-entropy.

Shannon Entropy. For uniformly distributed inputs, one can express the probability $\Pr(V = s')$ of the program outputting s' in terms of the size of $P^{-1}(s')$, see (4). We define upper and lower bounds $p^\flat(s')$ and $p^\sharp(s')$ for $\Pr(V = s')$ on the basis of the under- and over-approximation of $P^{-1}(s')$ given in Theorem 3.

Formally, we assume $s' \in F_{reach}$ and define

$$
\begin{aligned}
p^\flat(s') &= \max\left\{\frac{\#(\{s \mid (s, s') \in \overline{F}_{reach}{}^\flat\})}{\#(I)}, \frac{1}{\#(I)}\right\} \\
p^\sharp(s') &= \frac{\#(\{s \mid (s, s') \in \overline{F}_{reach}{}^\sharp\})}{\#(I)} .
\end{aligned}
$$

From Theorem 3 then follows that

$$
p^\flat(s') \leq \Pr(V = s') \leq p^\sharp(s') \tag{7}
$$

for all $s' \in F_{reach}{}^\flat$. These bounds extend to all $s' \in F_{reach}$ because for $s' \in F_{reach} \setminus F_{reach}{}^\flat$, the value $p^\flat(s') = 1/\#(I)$ is an under-approximation of the probability $p(V = s')$.

The following theorem shows that we can bound $H(U|V)$ in terms of combinations of upper and lower bounds for the preimage sizes.

Theorem 4. *If U is uniformly distributed, the remaining uncertainty $H(U|V)$ is bounded as follows*

$$
\begin{aligned}
&\sum_{s' \in F_{reach}{}^\sharp} p^\sharp(s') \log p^\flat(s') + \log \#(I) \\
&\qquad \leq H(U|V) \\
&\qquad\quad \leq \sum_{s' \in F_{reach}{}^\flat} p^\flat(s') \log p^\sharp(s') + \log \#(I) .
\end{aligned}
$$

Proof. P implements a total function. As a consequence, V is determined by U. We obtain $H(U) = H(UV) = H(U|V) + H(V)$ and conclude $H(U|V) = H(U) - H(V)$. As U is uniformly distributed, we have $H(U) = \log \#(I)$. By definition of Shannon entropy,

$$
H(V) = \sum_{s' \in F_{reach}} \Pr(V = s')(-\log \Pr(V = s')) . \tag{8}
$$

Observe that $-\log$ is nonnegative and decreasing on $(0, 1]$. Together with the bounds from (7), this monotonicity implies that replacing in (8) the occurrences of $-\log \Pr(V = s')$ by $-\log p^\sharp(s')$, replacing the remaining occurrences of $\Pr(V = s')$ by $p^\flat(s')$, and dropping the summands corresponding to elements of $F_{reach} \setminus {F_{reach}}^\flat$ will only decrease the sum, which leads to the upper bound on $H(U|V)$. The lower bound follows along the same line.

Min-entropy. The following theorem shows that it suffices to over- and under-approximate the size of the range of a program P in order to obtain approximations for the remaining uncertainty about P's input.

Theorem 5. *If U is uniformly distributed, the remaining uncertainty $H_\infty(U|V)$ of a program P is bounded as follows*

$$\log \frac{\#(I)}{\#({F_{reach}}^\sharp)} \le H_\infty(U|V) \le \log \frac{\#(I)}{\#({F_{reach}}^\flat)} .$$

Proof. Smith [59] shows that $H_\infty(U|V) = \log(\#(I)/\#(\Pi))$. The assertion then follows from Theorem 1 and the monotonicity of the logarithm.

5 Randomized Quantification

In Section 4 we showed how to obtain bounds on the remaining uncertainty about a program's input by computing over- and under-approximations of the set of reachable states and the corresponding preimages.

While the presented approach for computing min-entropy bounds requires determining the size of (an approximation of) the set of reachable states (see Theorem 5), the approach for computing Shannon-entropy bounds requires *enumerating* this set (see Theorem 4). This enumeration constitutes the bottleneck of our approach and inhibits scalability to large systems.

In this section, we show that the enumeration of the set of reachable states can be replaced by sampling the preimages with probabilities according to their relative sizes. To this end, we run the program on a randomly chosen input and approximate the preimage of the corresponding output. We combine the sizes of the approximations of the preimage and obtain upper and lower bounds for the remaining uncertainty. Moreover, we give confidence levels for these bounds. These confidence levels are already close to 1 for a number of samples of as small as $O((\log \#(I))^2)$.

On a technical level, our result makes use of the fact that the random variable given by the logarithm of the size of randomly chosen preimages has small variance [9]. The Chebyshev inequality implies that the estimations obtained from sampling this random variable are likely to be accurate.

5.1 Rant: A Randomized Algorithm for Quantitative Information Flow Analysis

Given a program P, our goal is to compute bounds $H^\sharp$ and $H^\flat$ with quality guarantees for the remaining uncertainty $H(U|V)$ about the program's input when the program's output is known. More precisely, given a confidence level $p \in [0,1)$ and a desired degree of precision $\delta > 0$, we require that

$$H^\flat - \delta \;\leq\; H(U|V) \;\leq\; H^\sharp + \delta$$

with a probability of at least p.

Algorithm. Our procedure Rant computes such bounds in an incremental fashion. After an initialization phase in lines 1 and 2, Rant randomly picks an initial state $s \in I$, see line 4 in Figure 2. Then, Rant runs the program P on input s to determine the final state s' and the corresponding execution path π, see line 5. We use the technique described in Section 4.2 for determining an over-approximation of the preimage of s' in line 6. Note that $\overline{F_{reach}}^\sharp$ only needs to be computed once and can be re-used for all iterations of the while loop. We use the techniques described in 4.2 for determining an under-approximation of the preimage of s' in line 7. The variables $H^\sharp$ and $H^\flat$ aggregate the logarithms of the preimage sizes. After

$$n = \frac{(\log \#(I))^2}{(1-p)\delta^2}$$

many iterations of while loop, $H^\sharp$ and $H^\flat$ are normalized by n and returned as upper and lower bounds for $H(U|V)$, respectively.

Counting Preimage Sizes. The computations in lines 6 and 7 of Rant require an algorithm that, given a set A, returns the number of elements $\#(A)$ in A. If A is represented as a formula ϕ, this number corresponds to the number of models for ϕ. For example, if A is represented in linear arithmetic, this task can be performed efficiently using Barvinok's algorithm [8]. The Lattice Point Enumeration Tool (LattE) [43] provides an implementation of this algorithm.

Correctness. The following theorem states the correctness of the algorithm Rant.

Theorem 6. *Let P be a program, $\delta > 0$, and $p \in [0,1)$. Let U be uniformly distributed. If* Rant(P, δ, p) *outputs* $(H^\sharp, H^\flat)$*, then*

$$H^\flat - \delta \leq H(U|V) \leq H^\sharp + \delta$$

with a probability of more than p.

We need the following lemma for the proof of Theorem 6. The proof of the lemma is based on a result from [9].

```
function RANT
input
    $P$ : program
    $\delta > 0$ : desired precision
    $p \in [0, 1)$ : desired confidence level
vars
    $\pi$: program path
output
    $H^\sharp, H^\flat$ : upper and lower bounds for $H(U|V)$
begin
    $n := 0$
    $H^\flat := H^\sharp := 0$
    while $n < \log(\#(I))^2/((1-p)\delta^2)$ do
        $s :=$ choose from $I$ randomly
        $(\pi, s') :=$ MKPATH($s$)
        $H^\sharp := H^\sharp + \log\#(\{s'' \mid (s'', s') \in \overline{F}_{reach}^{\sharp}\})$
        $H^\flat := H^\flat + \log\#(\{s'' \mid (s'', s') \in \rho_\pi\})$
        $n := n + 1$
    done
    return $(H^\sharp/n, H^\flat/n)$
end.
```

Fig. 2. Randomized procedure RANT for computing an approximation of the remaining uncertainty about the input of a program. The relation $\overline{F}_{reach}^{\sharp}$ is computed once and re-used for all iterations of the while loop.

Lemma 1. *Let X be a random variable with range of size m and let $\delta > 0$. Let $x_1, \ldots, x_n$ be the outcomes of n independent repetitions of the experiment associated with X. Then*

$$-\frac{1}{n}\sum_{i=1}^{n} \log \Pr(X = x_i) - \delta \quad \leq \quad H(X) \quad \leq \quad -\frac{1}{n}\sum_{i=1}^{n} \log \Pr(X = x_i) + \delta$$

with a probability of more than $1 - \frac{(\log m)^2}{n\delta^2}$.

Proof. We define the random variable Y by $Y(x) = -\log \Pr(X = x)$. Then we have $E(Y) = H(X)$ for the expected value $E(Y)$ of Y. By additivity of the expectation, we also have $E(Z) = H(X)$ for the sum $Z = \frac{1}{n}\sum_{i=1}^{n} Y_i$ of n independent instances Y_i of Y. In [9] it is shown that the variance $Var[Z]$ of Z is bounded from above by

$$Var[Z] \leq \frac{(\log m)^2}{n} \ .$$

The Chebyshev inequality

$$\Pr(|Q - E(Q)| \geq \delta) \leq \frac{Var[Q]}{\delta^2} \qquad (9)$$

gives upper bounds for the probability that the value of a random variable Q deviates from its expected value $E(Q)$ by at least δ. We apply (9) to Z with the expectation and variance bounds derived above and obtain

$$\Pr(|Z - H(X)| \geq \delta) \leq \frac{(\log m)^2}{n\delta^2} .$$

Considering the complementary event and inserting the definition of Z we obtain

$$\Pr\left(\left|-\frac{1}{n}\sum_{i=1}^{n} \log \Pr(X = x_i) - H(X)\right| \leq \delta\right) \geq 1 - \frac{(\log m)^2}{n\delta^2} ,$$

from which the assertion follows immediately.

We are now ready to give the proof of Theorem 6.

Proof (Proof of Theorem 6). Let s'_i be the final state of P that is computed in line 5 of the ith loop iteration of RANT, for $i \in \{1, \ldots, n\}$. For uniformly distributed U, we have $H(U) = \log \#(I)$ and $\Pr(V = s'_i) = \#(P^{-1}(s'_i))/\#(I)$. As V is determined by U, $H(U|V) = H(U) - H(V)$. Replacing $H(V)$ by the approximation given by Lemma 1 we obtain

$$\begin{aligned} &H(U) + \frac{1}{n}\sum_{i=1}^{n} \log \Pr(V = s'_i) \\ =\ &\log \#(I) + \frac{1}{n}\sum_{i=1}^{n} \log \frac{\#(P^{-1}(s'_i))}{\#(I)} \\ =\ &\frac{1}{n}\sum_{i=1}^{n} \log \#(P^{-1}(s'_i)) \end{aligned}$$

Lemma 1 now implies that

$$\frac{1}{n}\sum_{i=1}^{n} \log \#(P^{-1}(s'_i)) - \delta \leq H(U|V) \leq \frac{1}{n}\sum_{i=1}^{n} \log \#(P^{-1}(s'_i)) + \delta$$

with a probability of more than

$$1 - \frac{(\log \#(F_{reach}))^2}{n\delta^2} ,$$

which is larger than

$$\min\left\{1 - \frac{(\log \#(I))^2}{n\delta^2}, 1 - \frac{(\log \#(F))^2}{n\delta^2}\right\} .$$

This statement remains valid if the preimage sizes on the left and right hand sides are replaced by under- and over-approximations, respectively. Finally, observe that the loop guard ensures that the returned bounds hold with probability of more than p, which concludes this proof.

Observe that the proof of Theorem 6 implies that in scenarios where $\#(F_{reach})$ is known and smaller than I, a smaller number of samples is already sufficient for obtaining a desired confidence level. For example, when analyzing a program with a single Boolean output, the bounds delivered by RANT are valid with a probability of more than

$$1 - \frac{1}{n\delta^2} \,.$$

In general, however, the computation of $\#(F_{reach})$ requires an additional analysis step. For simplicity, our presentation hence focusses on the weaker bounds in terms of $\#(I)$.

Note that, if the sizes of the preimages of P can be determined precisely, we have $H^\sharp = H^\flat$. Then Theorem 6 gives tight bounds for the value of $H(U|V)$. In this way, RANT can be used to replace the algorithm QUANT from [2].

5.2 Complexity of Approximating Entropy

The algorithm RANT relies on the approximation of the sizes of the preimages for given sampled outputs of the program. It is natural to ask whether bounds on the entropy can be estimated by sampling alone, i.e. without resorting to structural properties of the program.

A result by Batu et al. [9] suggests that this cannot be done. They show that there is no algorithm that, by sampling alone, can approximate the entropy of every random variable X with a range of size m within given multiplicative bounds. They also show that, for random variables with high entropy (more precisely $H(X) > \log(m/\gamma^2)$, for some $\gamma > 0$) any algorithm that delivers approximations H with

$$\frac{1}{\gamma} H \leq H(X) \leq \gamma H$$

is required to take at least $\Omega(m^{1/\gamma^2})$ samples.

However, if in addition to the samples, the algorithm has access to an oracle that reveals the probability $\Pr(X = x)$ with which each sample x occurs, the entropy can be estimated within multiplicative bounds using a number of samples that is proportional to $(\log m)/h$, where $h \leq H(X)$.

Lemma 1 extends this result to obtain additive bounds for $H(X)$. These bounds hold without any side-condition on $H(X)$, which allows us to determine the number of samples that are required for obtaining confidence levels that hold *for all* X with $\#(ran(X)) \leq m$. The algorithm RANT builds on this result and employs the techniques presented in Section 4 for approximating the probabilities of events on demand, allowing us to derive bounds on the information leakage of real programs.

6 Related Work

For an overview of language-based approaches to information-flow security, see the survey by Sabelfeld and Myers [56].

Denning is the first to quantify information flow in terms of the reduction in uncertainty about a program variable [26]. Millen [49] and Gray [31] use information theory to derive bounds on the transmission of information between processes in multiuser systems. Lowe [44] shows that the channel capacity of a program can be overapproximated by the number of possible behaviors.

The use of equivalence relations to characterize qualitative information flow was proposed by Cohen [22] and has since then become standard, see e.g. [6,7,27,57,61].

Clark, Hunt, and Malacaria [18] connect equivalence relations to quantitative information flow, and propose the first type system for statically deriving quantitative bounds on the information that a program leaks [19]. The analysis assumes as input upper and lower bounds on the entropy of the input variables and performs compositional reasoning on basis of those bounds. For loops with high guards, the analysis always reports the complete leakage of the guard.

Malacaria [45] characterizes the leakage of loops in terms of the loop's output and the number of iterations. In our model, the information that is revealed by the number of loop iterations can be captured by augmenting loops with observable counters, as shown in Section 2. In this way, our method can be used to automatically determine this information.

Mu and Clark [52] propose an automatic quantitative analysis based on probabilistic semantics. Their analysis delivers precise results, but is limited to programs with small state-spaces due to the explicit representation of distributions. An abstraction technique [51] addresses this problem by partitioning the (totally ordered) domain into intervals, on which a piecewise uniform distribution is assumed. Our approach is also based on partitioning the input domain, however, without the restriction to intervals. Furthermore, we avoid the enumeration of all blocks by the choice of a random subset.

Köpf and Basin [38] characterize the leaked information in terms of the number of program executions, where an attacker can adaptively provide inputs. The algorithms for computing this information for a concrete system rely on an enumeration of the entire input space and are difficult to scale to larger systems.

Backes, Köpf, and Rybalchenko [2] show how to synthesize equivalence relations that represent the information that a program leaks, and how to quantify them by determining the sizes of equivalence classes. Our approach shows that the exact computation of the sizes of the equivalence classes can be replaced by over- and underapproximations, and that the enumeration of equivalence classes can be replaced by sampling. This enables our approach to scale to larger programs, e.g., those with unbounded loops.

McCamant and Ernst [48] propose a dynamic taint analysis approach for quantifying information flow. Their method provides over-approximations of the leaked information along a particular path, but does not yield guarantees for all program paths, which is important for security analysis. For programs for which preimages can be approximated, our method can be used to derive upper and lower bounds for the leakage of *all* paths without the need for complete exploration.

Newsome, McCamant, and Song [53] use the feasible outputs along single program paths as lower bounds for channel capacity, and they apply a number of heuristics to approximate upper bounds on the number of reachable states of a program. They assume a fixed upper bound on the number of loop unrollings. In contrast, our technique does not require an upper bound on the number of loop iterations, and it comes with formal quality guarantees for the estimated quantities.

Heusser and Malacaria [34] use model-checking to verify assertions about the number of feasible outputs of a program. A valid assertion translates to an upper bound on the channel capacity of the program. In contrast, we apply model counting techniques to immediately obtain upper and lower bounds on the number of feasible outputs.

Chatzikokolakis, Chothia, and Guha [16] use sampling to build up a statistical model of a probabilistic program, which is treated as a black box. Based on this model, they compute the maximum leakage w.r.t. all possible input distributions. In contrast, our approach is based on the actual semantics of deterministic programs, as given by the source code, and we use a randomized algorithm to compute the adversary's remaining uncertainty about the input.

A number of alternative information measures have been considered in the literature. Di Pierro, Hankin, and Wiklicky [55] measure information flow in terms of the number of statistical tests an attacker has to perform in order to distinguish two computations based on different secrets. Clarkson, Myers, and Schneider [20] propose to measure information flow in terms of the accuracy of an attacker's belief about a secret, which may also be wrong. Reasoning about beliefs is out of the scope of entropy-based measures, such as the ones used in this chapter. One advantage of entropy-based measures is the direct connection to equivalence relations, which makes them amenable to automated reasoning techniques. Finally, we mention that information-theoretic notions of leakage are also used for analyzing anonymity protocols, see e.g. [17].

Our approach relies on abstract interpretation and symbolic execution techniques for the approximation of the set of program outputs. There exist efficient implementations of abstract interpreters with abstraction functions covering a wide spectrum of efficiency/precision trade-offs, see e.g. [4, 10, 33, 42]. In particular, for bounding the block count one could apply tools for discovery of all valid invariants captured by numeric abstract domains, e.g., octagons or polyhedra [3, 50]. Similarly, we can rely on existing dynamic engines for symbolic execution that can deal with various logical representation of program states, including arithmetic theories combined with uninterpreted function symbols and propositional logic, e.g., VERISOFT, KLEE, DART, and BITSCOPE [12, 14, 28, 29].

7 Conclusions

The exact computation of the information-theoretic properties of programs can be prohibitively hard. In this chapter, we presented algorithms based on approximation and randomization that allow for a tractable yet sufficiently precise approximation of these properties. As ongoing work, we are putting these algorithms to work for the automatic analysis of microarchitectural side-channels [40].

References

1. Backes, M., Berg, M., Köpf, B.: Non-Uniform Distributions in Quantitative Information-Flow. In: Proc. 6th ACM Conference on Information, Computer and Communications Security, ASIACCS 2011, pp. 367–374. ACM (2011)
2. Backes, M., Köpf, B., Rybalchenko, A.: Automatic Discovery and Quantification of Information Leaks. In: Proc. IEEE Symp. on Security and Privacy, S&P 2009, pp. 141–153. IEEE (2009)
3. Bagnara, R., Hill, P.M., Zaffanella, E.: The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. Science of Computer Programming 72(1-2) (2008)
4. Ball, T., Majumdar, R., Millstein, T., Rajamani, S.: Automatic predicate abstraction of C programs. In: Proc. ACM Conf. on Programming Language Design and Implementation, PLDI 2001, pp. 203–213. ACM (2001)
5. Ball, T., Millstein, T.D., Rajamani, S.K.: Polymorphic predicate abstraction. ACM Trans. Program. Lang. Syst. 27(2) (2005)
6. Banerjee, A., Naumann, D.A., Rosenberg, S.: Expressive declassification policies and modular static enforcement. In: Proc. IEEE Symp. on Security and Privacy, S&P 2008, pp. 339–353. IEEE (2008)
7. Barthe, G., D'Argenio, P., Rezk, T.: Secure information flow by self-composition. In: Proc. IEEE Computer Security Foundations Workshop, CSFW 2004, pp. 100–114. IEEE (2004)
8. Barvinok, A.: A Polynomial Time Algorithm for Counting Integral Points in Polyhedra when the Dimension is Fixed. Mathematics of Operations Research 19, 189–202 (1994)
9. Batu, T., Dasgupta, S., Kumar, R., Rubinfeld, R.: The complexity of approximating entropy. In: Proc. ACM Symp. on Theory of Computing, STOC 2002, pp. 678–687. ACM (2002)
10. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.: A static analyzer for large safety-critical software. In: Proc. ACM Conf. on Programming Language Design and Implementation, PLDI 2003, pp. 196–207. ACM (2003)
11. Boreale, M.: Quantifying information leakage in process calculi. Information and Computation 207(6), 699–725 (2009)
12. Brumley, D., Hartwig, C., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Song, D.: BitScope: Automatically dissecting malicious binaries. Technical Report CS-07-133, School of Computer Science, Carnegie Mellon University (2007)
13. Cachin, C.: Entropy Measures and Unconditional Security in Cryptography. PhD thesis, ETH Zürich (1997)
14. Cadar, C., Dunbar, D., Engler, D.R.: KLEE: Unassisted and automatic generation of high-coverage tests for complex systems programs. In: Proc. USENIX Symp. on Operating Systems Design and Implementation, OSDI 2008, pp. 209–224. USENIX (2008)
15. Cerny, P., Chatterjee, K., Henzinger, T.: The complexity of quantitative information flow problems. In: Proc. IEEE Computer Security Foundations Symposium, CSF 2011. IEEE (2011) (to appear)
16. Chatzikokolakis, K., Chothia, T., Guha, A.: Statistical Measurement of Information Leakage. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 390–404. Springer, Heidelberg (2010)
17. Chatzikokolakis, K., Palamidessi, C., Panangaden, P.: Anonymity protocols as noisy channels. Information and Computation 206(2-4), 378–401 (2008)

18. Clark, D., Hunt, S., Malacaria, P.: Quantitative Information Flow, Relations and Polymorphic Types. J. Log. Comput. 18(2), 181–199 (2005)
19. Clark, D., Hunt, S., Malacaria, P.: A static analysis for quantifying information flow in a simple imperative language. Journal of Computer Security 15(3), 321–371 (2007)
20. Clarkson, M.R., Myers, A.C., Schneider, F.B.: Belief in Information Flow. In: Proc. IEEE Computer Security Foundations Workshop, CSFW 2005, pp. 31–45. IEEE (2005)
21. Clulow, J.: The Design and Analysis of Cryptographic Application Programming Interfaces for Security Devices. Master's thesis, University of Natal, SA (2003)
22. Cohen, E.: Information Transmission in Sequential Programs. In: Foundations of Secure Computation, pp. 297–335. Academic Press (1978)
23. Cook, B., Gupta, A., Magill, S., Rybalchenko, A., Simsa, J., Vafeiadis, V.: Finding heap-bounds for hardware synthesis. In: Proc. Intl. Conf. on Formal Methods in Computer-Aided Design, FMCAD 2009, pp. 205–212. IEEE (2009)
24. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proc. ACM Symp. on Principles of Programming Languages, POPL 1977, pp. 238–252. ACM (1977)
25. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Proc. ACM Symp. on Principles of Programming Languages, POPL 1978, pp. 84–96. ACM (1978)
26. Denning, D.E.: Cryptography and Data Security. Addison-Wesley (1982)
27. Giacobazzi, R., Mastroeni, I.: Abstract Non-Interference: Parameterizing Non-Interference by Abstract Interpretation. In: Proc. ACM Symp. on Principles of Programming Languages, POPL 2004, pp. 186–197. ACM (2004)
28. Godefroid, P.: Software model checking: The VeriSoft approach. Formal Methods in System Design 26(2), 77–101 (2005)
29. Godefroid, P., Klarlund, N., Sen, K.: DART: directed automated random testing. In: Proc. ACM Conf. on Programming Language Design and Implementation, PLDI 2005, pp. 213–223. ACM (2005)
30. Gomez, C., Sabharwal, A., Selman, B.: Chapter 20: Model counting. In: Handbook of Satisfiability. Frontiers in Artificial Intelligence and Applications, vol. 185, IOS Press (2009)
31. Gray, J.W.: Toward a Mathematical Foundation for Information Flow Security. Journal of Computer Security 1(3-4), 255–294 (1992)
32. Gulwani, S., Jain, S., Koskinen, E.: Control-flow refinement and progress invariants for bound analysis. In: Proc. ACM Conf. on Programming Language Design and Implementation, PLDI 2009, pp. 375–385. ACM (2009)
33. Henzinger, T.A., Jhala, R., Majumdar, R., McMillan, K.L.: Abstractions from proofs. In: Proc. ACM Symp. on Principles of Programming Languages, POPL 2004, pp. 232–244. ACM (2004)
34. Heusser, J., Malacaria, P.: Quantifying information leaks in software. In: 26th Annual Computer Security Applications Conference, ACSAC 2010, pp. 261–269. ACM (2010)
35. Jeannet, B., Miné, A.: Apron: A library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
36. Jhala, R., Majumdar, R.: Software model checking. ACM Comput. Surv. 41, 21:1–21:54 (2009)

37. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Koblitz, N. (ed.) CRYPTO 1996. LNCS, vol. 1109, pp. 104–113. Springer, Heidelberg (1996)
38. Köpf, B., Basin, D.: An Information-Theoretic Model for Adaptive Side-Channel Attacks. In: Proc. ACM Conf. on Computer and Communications Security, CCS 2007, pp. 286–296. ACM (2007)
39. Köpf, B., Dürmuth, M.: A Provably Secure and Efficient Countermeasure against Timing Attacks. In: Proc. IEEE Computer Security Foundations Symposium, CSF 2009, pp. 324–335. IEEE (2009)
40. Köpf, B., Mauborgne, L., Ochoa, M.: Automatic Quantification of Cache Side-Channels. In: Madhusudan, P., Seshia, S.A. (eds.) CAV 2012. LNCS, vol. 7358, pp. 564–580. Springer, Heidelberg (2012)
41. Köpf, B., Rybalchenko, A.: Approximation and Randomization for Quantitative Information-Flow Analysis. In: Proc. 23rd IEEE Computer Security Foundations Symposium, CSF 2010, pp. 3–14. IEEE (2010)
42. Lalire, G., Argoud, M., Jeannet, B.: The interproc analyzer, `http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc/index.html`
43. Loera, J.A.D., Haws, D., Hemmecke, R., Huggins, P., Tauzer, J., Yoshida, R.: LattE, `http://www.math.ucdavis.edu/~latte/` (accessed November 08, 2008)
44. Lowe, G.: Quantifying Information Flow. In: Proc. IEEE Computer Security Foundations Workshop, CSFW 2002, pp. 18–31. IEEE (2002)
45. Malacaria, P.: Risk assessment of security threats for looping constructs. Journal of Computer Security 18(2), 191–228 (2010)
46. Manna, Z., Pnueli, A.: Temporal verification of reactive systems: Safety. Springer (1995)
47. Massey, J.L.: Guessing and Entropy. In: Proc. IEEE Intl. Symp. on Information Theory, ISIT 1994, p. 204. IEEE Computer Society (1994)
48. McCamant, S., Ernst, M.D.: Quantitative information flow as network flow capacity. In: Proc. ACM Conf. on Programming Language Design and Implementation, PLDI 2008, pp. 193–205. ACM (2008)
49. Millen, J.K.: Covert Channel Capacity. In: Proc. IEEE Symp. on Security and Privacy, S&P 1987, pp. 60–66. IEEE (1987)
50. Miné, A.: The Octagon abstract domain. Higher-Order and Symbolic Computation 19(1), 31–100 (2006)
51. Mu, C., Clark, D.: An Interval-based Abstraction for Quantifying Information Flow. ENTCS 253(3), 119–141 (2009)
52. Mu, C., Clark, D.: Quantitative Analysis of Secure Information Flow via Probabilistic Semantics. In: Proc. 4th International Conference on Availability, Reliability and Security, ARES 2009, pp. 49–57. IEEE Computer Society (2009)
53. Newsome, J., McCamant, S., Song, D.: Measuring channel capacity to distinguish undue influence. In: Proc. ACM Workshop on Programming Languages and Analysis for Security, PLAS 2009, pp. 73–85. ACM (2009)
54. Park, S., Pfenning, F., Thrun, S.: A Probabilistic Language based upon Sampling Functions. In: Proc. ACM Symposium on Principles of Programming Languages, POPL 2005 (2005)
55. Pierro, A.D., Hankin, C., Wiklicky, H.: Approximate Non-Interference. In: Proc. IEEE Computer Security Foundations Workshop, CSFW 2002, pp. 3–17. IEEE (2002)

56. Sabelfeld, A., Myers, A.C.: Language-based Information-Flow Security. IEEE J. Selected Areas in Communication 21(1), 5–19 (2003)
57. Sabelfeld, A., Myers, A.C.: A model for delimited information release. In: Futatsugi, K., Mizoguchi, F., Yonezaki, N. (eds.) ISSS 2003. LNCS, vol. 3233, pp. 174–191. Springer, Heidelberg (2004)
58. Shannon, C.E.: A Mathematical Theory of Communication. Bell System Technical Journal 27, 379–423, 623–656 (1948)
59. Smith, G.: On the foundations of quantitative information flow. In: de Alfaro, L. (ed.) FOSSACS 2009. LNCS, vol. 5504, pp. 288–302. Springer, Heidelberg (2009)
60. Yasuoka, H., Terauchi, T.: Quantitative information flow - verification hardness and possibilities. In: Proc. IEEE Computer Security Foundations Symposium, CSF 2010, pp. 15–27. IEEE (2010)
61. Zdancewic, S., Myers, A.C.: Robust declassification. In: Proc. IEEE Computer Security Foundations Workshop, CSFW 2001, pp. 15–23. IEEE (2001)

ODE Analysis of Biological Systems

Diana-Elena Gratie, Bogdan Iancu, and Ion Petre

Computational Biomodeling Laboratory
Turku Centre for Computer Science and Åbo Akademi University
Turku, Finland
{dgratie,biancu,ipetre}@abo.fi

Abstract. This chapter aims to introduce some of the basics of modeling with ODEs in biology. We focus on computational, numerical techniques, rather than on symbolic ones. We restrict our attention to reaction-based models, where the biological interactions are mechanistically described in terms of reactions, reactants and products. We discuss how to build the ODE model associated to a reaction-based model; how to fit it to experimental data and estimate the quality of its fit; how to calculate its steady state(s), mass conservation relations, and its sensitivity coefficients. We apply some of these techniques to a model for the heat shock response in eukaryotes.

Keywords: Biomodeling, reaction-based models, ODE-based models, ODE analysis, parameter estimation, model identifiability, model refinement, heat shock response.

1 Introduction

Mathematical modeling with ordinary differential equations (ODEs) has a very long tradition in biology and ecology. Efforts to apply ODEs to understand population dynamics started already in the 18th century (see, e.g., Malthus's growth model [40]) as an effort to apply the principles of physical sciences to biological sciences as well. This research area led to major developments both in biology and ecology, as well as in mathematics. The field has long been called *biomathematics*, *mathematical biology* or *theoretical biology* and it typically involved researchers from life sciences (biology, biochemistry and ecology in particular), mathematics and more recently, from computer science and engineering. It has recently witnessed an explosion of interest in the computer science community due to the fast-paced developments in quantitative laboratory technologies. The developments on the computational side have also been influential, allowing for analyzing ever larger models and opening the door to new fields of research such as computational drug design or personalized medicine.

This chapter is primarily targeting the computer science community. Many computer scientists working in biomodeling seem to prefer a discrete stochastic approach rather than one based on ODEs. Such a choice is in some ways natural for computer scientists as it leads to new types of applications of formalisms that are well-studied in computer science, such as Petri nets, process algebra, finite automata, etc. On the other hand, such

M. Bernardo et al. (Eds.): SFM 2013, LNCS 7938, pp. 29–62, 2013.

methods come with their own limitations, especially in terms of numerical simulations of large models and in parameter estimation. Moreover, ODE-based modeling offers a huge array of analysis methods, some of which do not have a correspondent on the discrete side. Our chapter aims to introduce some of the basics of modeling with ODEs in biology, especially in terms of building such a model, analyzing some of its properties, and estimating its parameters. We focus on computational, numerical techniques, rather than on symbolic ones. We restrict our attention to reaction-based models, where the biological interactions are mechanistically described in terms of reactions, reactants and products.

The chapter is structured as follows. We discuss in Section 2 the notion of reaction-based models and introduce briefly the stochastic modeling approach in terms of continuous time Markov chains. We then discuss in more details the modeling with ODEs in Section 3. The parameter estimation problem is discussed in Section 4. We then introduce in Section 5 several analysis techniques, including steady state analysis, sensitivity analysis, and identification of mass conservation relations. As a case-study we discuss the modeling of the heat shock response in Section 6. We conclude with discussions in Section 7.

2 Reaction-Based Models

Reaction-based models are formalized as sets of reactions that describe the given system in terms of mechanistic interactions between the species of interest. We discuss separately two types of reactions: reversible and irreversible. In the following we consider a model M consisting of a set of m species $\Sigma = \{S_1, S_2, ..., S_m\}$ and n (reversible or irreversible) reactions R_j, $1 \leq j \leq n$.

Generalities. If reaction R_j, $1 \leq j \leq n$ is irreversible, then it has the following form:

$$R_j \,:\, c_{1,j}S_1 + c_{2,j}S_2 + \ldots + c_{m,j}S_m \xrightarrow{k_j} c'_{1,j}S_1 + c'_{2,j}S_2 + \ldots + c'_{m,j}S_m. \tag{1}$$

On the other hand, if it is reversible, then it is of the following form:

$$R_j \,:\, c_{1,j}S_1 + c_{2,j}S_2 + \ldots + c_{m,j}S_m \underset{k_j^-}{\overset{k_j^+}{\rightleftarrows}} c'_{1,j}S_1 + c'_{2,j}S_2 + \ldots + c'_{m,j}S_m. \tag{2}$$

In both cases, $k_j \geq 0$ ($k_j^+, k_j^- \geq 0$, resp.) is the *kinetic rate constant* of the irreversible (reversible, resp.) reaction R_j and $c_{1,j}, \ldots, c_{m,j}, c'_{1,j}, \ldots, c'_{m,j} \geq 0$ are non-negative integers that represent the quantitative proportion in which species participate in a reaction. The *stoichiometric coefficient* of molecular species S_i in reaction R_j is $n_{i,j} = c'_{i,j} - c_{i,j}$. The stoichiometric coefficients can be represented in a *stoichiometric matrix* $\mathbf{N} = (n_{i,j})_{m \times n}$. The (i,j) entry of the matrix is the stoichiometric coefficient of species S_i in reaction R_j. If $n_{i,j} > 0$ ($n_{i,j} < 0$, resp.), then we say that S_i is *produced* (*consumed*, resp.) in reaction R_j.

The reactants (the species indicated on the left-hand side of the reaction) are referred to as *substrates*, while the species produced as a result of the reaction being triggered (indicated on the right-hand side of the reaction) are called *products*. A species S_i with $c_{i,j} = 0$ ($c'_{i,j} = 0$, resp.) is usually omitted from the left- (right-, resp.) hand side of reaction R_j.

Note that a reversible reaction can be divided into two different irreversible reactions, as follows:

$$\begin{aligned} R_j^{(1)} &: c_{1,j}S_1 + c_{2,j}S_2 + \ldots + c_{m,j}S_m \xrightarrow{k_j^+} c'_{1,j}S_1 + c'_{2,j}S_2 + \ldots + c'_{m,j}S_m \\ R_j^{(2)} &: c'_{1,j}S_1 + c'_{2,j}S_2 + \ldots + c'_{m,j}S_m \xrightarrow{k_j^-} c_{1,j}S_1 + c_{2,j}S_2 + \ldots + c_{m,j}S_m \end{aligned}$$

The sum $\sum_{i=1}^{m} c_{i,j}$ for an irreversible reaction R_j is called the *molecularity* of reaction R_j. We consider here only reactions with molecularity at most two. Reactions with molecularity three are very rare, due to the high improbability of having three molecular entities simultaneously colliding and forming a correct configuration that leads to the constitution of a molecular complex; a molecularity greater than three for an elementary reaction is unattainable, since a number of molecules greater than three cannot concomitantly collide [49].

Example 1. We consider here the representation of a simple ecological prey-predator model through coupled chemical reactions: the Lotka-Volterra system, [39, 60]. The model consists of species *Prey* and *Predator* and its reactions are shown in Table 1.

Table 1. The Lotka-Volterra model [39, 60]

$Prey \xrightarrow{k_1} 2 \times Prey,$	growth of prey population
$Prey + Predator \xrightarrow{k_2} 2 \times Predator,$	consumption of preys
$Predator \xrightarrow{k_3} \emptyset$	death of predators

The dynamics of the Lotka-Volterra system is periodical: the population of preys grows at a rate proportional to the current population, the presence of predators in the system induces a decrease in the population of preys at a rate proportional to the number of prey-predator encounters, the population of predators declines at a rate proportional to the current population of predators. We return to this example in the next section where we associate to it an ODE-based model. A plot of its numerical simulation is shown in Figure 1.

Associating a mathematical model. After building a reaction-based model, one then associates to it a mathematical model to facilitate quantitative analysis and simulation of the model. There are many approaches available for this, see e.g. [9, 12]. The two approaches that are most used (either in a direct way, or an indirect way, as the underlying semantic of a higher-level model) are the ODE-based approach and the one based on continuous time Markov chains (CTMCs). The modeling with CTMCs is described in more details elsewhere in this book; we only give it here a very brief presentation so that we can draw some comparison between the two in Section 7. We introduce the modeling with ODEs in more details in Section 3.

The stochastic approach is typically argued for on the basis of physical difficulties of ODE-based models with small populations [14, 15], or in terms of the network being too complex to describe in a deterministic way [61]. The stochastic formulation of a biochemical reaction network assumes homogeneity of substances and thermal equilibrium, see [16]. In this case, the model is usually described mathematically as a *continuous time Markov chain*, see [57]. Each species of the model becomes a time-dependent discrete stochastic variable indicating the number of individuals in that species, where time is modeled as a continuous variable. Formally, a stochastic process, $\{X(t), t \geq 0\}$, is a continuous-time Markov chain if for all $s, t \geq 0$, the following property is satisfied:

$$Pr\{X(s+t) = x_{s+t} | X(s) = x_s, X(u) = x_u, 0 \leq u \leq s\} = Pr\{X(s+t) = x_{s+t} | X(s) = x_s\}.$$

Intuitively, we say that the Markov chain is *memoryless*: its future dynamics depends only on the current state and not on the past states.

A continuous-time Markov chain is *time-homogeneous* if the following relation is satisfied:

$$Pr\{X(s+t) = j | X(s) = i\} = Pr\{X(t) = j | X(0) = i\}.$$

We discuss here only time-homogeneous systems.

Given a vector of non-negative integers $\mathbf{X} = (X_1, X_2, ..., X_m)$ and species $S_1, S_2, ..., S_m$ the *grand probability function* of the model, $Pr(\mathbf{X}, t)$, is the probability that there are X_1 species S_1, X_2 species S_2, ..., X_m species S_m at time t. We consider all species to be distributed randomly and homogeneously in the volume V. The central hypothesis for the stochastic formulation of chemical kinetics is that the probability of a particular combination of reactants to react according to a given reaction R in the next infinitesimal time interval $(t, t+dt)$ is $c_R dt$, for a certain constant c_R, called the *stoichiometric constant* of the reaction. The probability of a reaction occurring in the interval $(t, t+dt)$ is given by the formula $N_R \cdot c_R \cdot dt$, where by N_R we denote the number of combinations of reactants in the current state. For instance, for reaction $R^{(1)} : S_1 + S_2 \rightarrow S_3$, we have $N_{R^{(1)}} = X_1 \cdot X_2$. For reaction $R^{(2)} : 2S_1 \rightarrow S_3$, $N_{R^{(2)}} = X_1 \cdot (X_1 - 1)/2$.

For an infinitesimally small dt, the probability of the system being in a certain state at time $t+dt$ may be given by the following two scenarios: the system was in the current state at time t and no reaction occurred, or the system reached the current state as a result of a single reaction being triggered (the probability of having had two or more reactions is negligible). Denote by $a_k dt$ the probability of a reaction R_k occurring in the interval $(t, t+dt)$, given the state characterized by $\mathbf{X}$ at time t, and by $B_k dt$ the probability that reaction R_k occurs in the time interval $(t, t+dt)$ resulting in a state characterized by $\mathbf{X}$. The reasoning above can be formally written as follows:

$$Pr(\mathbf{X},t+dt) = Pr(\mathbf{X},t)(1-\sum_{k=1}^{n} a_k dt) + \sum_{k=1}^{n} B_k dt, \text{ i.e.,}$$

$$(Pr(\mathbf{X},t+dt) - Pr(\mathbf{X},t))/dt = -\sum_{k=1}^{n} a_k Pr(\mathbf{X},t) + \sum_{k=1}^{n} B_k, \text{ and so,}$$

$$\frac{\partial Pr(\mathbf{X},t)}{\partial t} = \sum_{k=1}^{n} (B_k - a_k Pr(\mathbf{X},t)). \tag{3}$$

Equation (3) is known in the literature as the *Chemical Master Equation*. A detailed mathematical analysis of a complex system using the chemical master equation has been proven to be intractable, see [61]. However, an alternative to the aforementioned approach is Gillespie's algorithm, introduced in [14, 15], that generates a random walk through the state space of the model, avoiding the solving of the master equation.

3 ODE-Based Models

We discuss in this section how to associate an ODE-based model to a reaction model. In this case, the dynamic behavior of the system is expressed in terms of the time-dependent evolution of each species' concentration. The deterministic framework of ordinary differential equations (ODEs) is often chosen as the default mathematical counterpart of a reaction-based system, sometimes followed-up by other modeling approaches. The basic quantities describing the ODE model are the concentrations $[S_1]$, $[S_2]$, ..., $[S_m]$ of the m species in the model, and the fluxes $v_1, v_2, ..., v_n$ of the n reactions in the model. The concentration is generally expressed either in terms of particle numbers (i.e. the number of molecules of species S, denoted $\#S$, in a solution with volume V), or in terms of moles of species S per volume V. The correspondence between the number of molecules and the number of moles is given by the relation:

$$\#S = [S] \cdot N_A,$$

where $N_A \approx 6.02214179 \cdot 10^{23}$particles/mol. The unit of $[S]$ is commonly denoted by M = mol $\cdot \mathrm{L}^{-1}$, where L is litre.

Without loss of generality, we will assume that all reactions are reversible and have the form in (2); an irreversible reaction is then a particular case, where one of the two kinetic constants is zero.

Each species S_i of the reaction model can be modeled as a function $[S_i] : \mathbb{R}_+ \to \mathbb{R}_+$ representing the time evolution of its concentration. The dependencies between the species can then be expressed in terms of a systems of ODEs in the variables $[S_i]$ modeling the change in $[S_i]$ as a function of all other variables:

$$d[S_i]/dt = \sum_{j=1}^{n} n_{i,j} v_j,$$

where v_j is the flux of reaction r_j and $n_{i,j}$ is the (i,j) stoichiometric coefficient. Here, we make the assumption that the only factor affecting the concentrations of the species are the reactions. Considering the vector of all reaction fluxes $v = (v_1, v_2, ..., v_n)^T$, the ODE representation of the entire reaction model can be written in a compact way as follows:

$$d[S]/dt = Nv, \tag{4}$$

where $[S] = ([S_1], [S_2], ..., [S_m])^T$ is a vector of the concentrations of all species in the reaction-based model, see [25].

Example 2. Consider the following reaction model:

$$\begin{aligned} &2A \rightarrow B, \\ &A + B \rightleftarrows C. \end{aligned}$$

Denote by v_1 and v_2 the fluxes of the two reactions in the model, respectively. (We discuss in the next section how the flux of a reaction is defined, depending on the kinetic law the modeler chooses.) Then the corresponding ODE model is:

$$\begin{bmatrix} \frac{d[A]}{dt} \\ \frac{d[B]}{dt} \\ \frac{d[C]}{dt} \end{bmatrix} = \begin{bmatrix} -2 & -1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}.$$

While the stoichiometries in a reaction-based model are constant, the concentrations of all species will vary in time as a function of the reaction fluxes, which are in turn dependent on the kinetics of each reaction and on the concentrations of all reactants. In the following, we describe in details two of the most common reaction kinetics: the mass-action principle, and Michaelis-Menten kinetics.

3.1 Law of Mass-Action

The most common biochemical kinetics follow the *mass action law*. It was introduced in [20,21], and it states that the *flux* (also called sometimes *rate*) of a reaction is proportional to the probability of the reactants colliding. Assuming a well-stirred environment, the probability of the substrates of a reaction colliding is proportional to their concentration to the power of their molecularity.

Example 3. For a simple reaction of the form

$$S_1 + S_2 \underset{k^-}{\overset{k^+}{\rightleftarrows}} 2P,$$

the reaction flux is

$$v = v_+ - v_- = k^+[S_1][S_2] - k^-[P]^2,$$

where v_+ represents the left-to-right (forward) reaction rate, v_- represents the right-to-left (backward) flux, k^+ is the left-to-right kinetic rate constant, and k^- represents the right-to-left kinetic rate constant. For the forward reaction, the molecularity of each of the two substrates S_1, S_2 is 1, and for the backward reaction the molecularity is 2. If the time is measured in seconds (s), and the concentration in M, then the unit for the reaction rates is M · s^{-1}. It follows that for monomolecular reactions (e.g. $S \rightarrow \emptyset$), the rate constant has unit s^{-1}, while for bimolecular reactions, the rate constant is measured in (M ·s)$^{-1}$.

Considering a general reversible reaction of the form (2), the reaction rate reads

$$v = v_+ - v_- = k_j^+ \prod_{i=1}^{m} [S_i]^{c_{i,j}} - k_j^- \prod_{i=1}^{m} [S_i]^{c'_{i,j}}.$$

The corresponding system of ODEs, following (4), is

$$\frac{d[S_i]}{dt} = n_{i,j} v = (c'_{i,j} - c_{i,j}) \left(k_j^+ \prod_{l=1}^{m} [S_l]^{c_{l,j}} - k_j^- \prod_{l=1}^{m} [S_l]^{c'_{l,j}} \right), 1 \leq l \leq m.$$

For reversible reactions, the ratio of substrate and product at steady state (i.e., when the forward and backward reaction rates are equal, $v_+ = v_-$) is a constant, K_{eq}, called the equilibrium constant:

$$K_{eq} = \frac{k_j^+}{k_j^-} = \frac{\prod_{i=1}^{m} [S_i]_{eq}^{c'_{i,j}}}{\prod_{i=1}^{m} [S_i]_{eq}^{c_{i,j}}},$$

where $[S_i]_{eq}$ represents the equilibrium concentration of species S_i.

The time course for a species S is obtained by integrating the corresponding ODE. For a simple decomposition reaction $S \xrightarrow{k} P_1 + P_2$, the time dynamics is described by the ODE $d[S]/dt = -k[S]$. Integrating over the interval [0, t) yields the analytical solution

$$\int_{S_0}^{S} d[S]/dt = - \int_{t=0}^{t} k dt \Rightarrow [S](t) = S_0 e^{-kt}.$$

Calculating the analytical solution for more complex models is however rarely possible.

Example 4. For the Lotka-Volterra model introduced in Example 1, the mass-action reaction fluxes for the three reactions in the system are the following:

$$v_1 = k_1[Prey], \quad v_2 = k_2[Prey][Predator], \quad v_3 = k_3[Predator].$$

The system of ODEs describing the dynamics of the Lotka-Volterra model is:

$$\begin{aligned} d[Prey]/dt &= v_1 - v_2 = k_1[Prey] - k_2[Prey][Predator] \\ d[Predator]/dt &= v_2 - v_3 = k_2[Prey][Predator] - k_3[Predator]. \end{aligned} \tag{5}$$

The periodic dynamics of the Lotka-Volterra model is depicted in Figure 1.

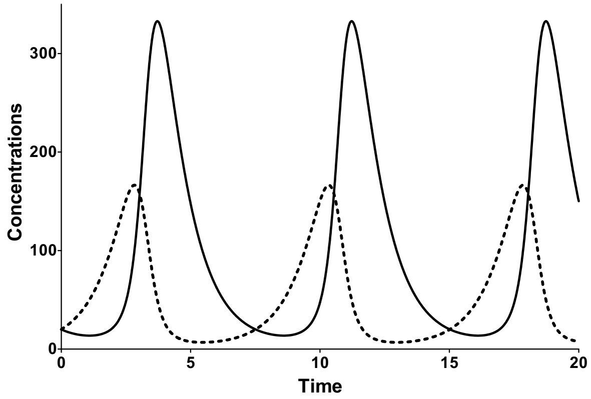


Fig. 1. The periodic time dynamics of the Lotka-Volterra model. The solid line represents the concentration of *Predator*, while the dotted line represents that of *Prey*. As the *Prey* population grows, the *Predator* population also grows; then there are more encounters *Predator-Prey* that reduce the *Prey* population; this reflects on the *Predator*, as they only multiply as long as they find food. When the size of *Predator* drops, the *Prey* population starts to grow, and the cycle repeats.

3.2 Kinetics of Enzymatic Reactions

Enzymatic reactions are a special class of biochemical reactions, where an enzyme is required for a reaction to take place, but the enzyme itself is not consumed during the reaction. The general form of an enzymatic reaction, as proposed in [7] based on previous experimental results of [28, 43], is:

$$S+E \underset{k_2}{\overset{k_1}{\rightleftarrows}} S:E \xrightarrow{k_3} P+E,$$

where E is an enzyme, S is the substrate of the reaction, $S:E$ is a substrate-enzyme complex, and P is the product. This system of reactions represents in fact the reaction $S \rightarrow P$, catalyzed by enzyme E. The system can be represented using mass-action kinetics, considering the following irreversible reactions:

$$S+E \xrightarrow{k_1} S:E, \quad S:E \xrightarrow{k_2} S+E, \quad S:E \xrightarrow{k_3} P+E. \tag{6}$$

The system of ODEs describing the mass-action dynamics of the reaction-based model (6) is the following:

$$\frac{d[S]}{dt} = -k_1[S][E] + k_2[S:E]; \tag{7}$$

$$\frac{d[E]}{dt} = -k_1[S][E] + k_2[S:E] + k_3[S:E]; \tag{8}$$

$$\frac{d[S:E]}{dt} = k_1[S][E] - k_2[S:E] - k_3[S:E]; \tag{9}$$

$$\frac{d[P]}{dt} = k_3[S:E]. \tag{10}$$

Michaelis-Menten kinetics. Because the system of ODEs (7) - (10) cannot be solved analytically, simplifying assumptions have been proposed. For example, the kinetic constants k_1, k_2 could be assumed to be much greater than k_3 ($k_1, k_2 \gg k_3$, see [43]), i.e. $[S:E]$ is negligible compared to $[S]$ and $[P]$, because the substrate-enzyme complex concentration is very low. This is called the *quasi-equilibrium* between the free enzyme E and the compound $S:E$.

This assumption has been further extended (see [7]) to considering that the system will eventually reach a state where the concentration of substrate-enzyme complex remains unchanged (*quasi-steady state* of $S:E$); the assumption only holds when $S_0 \gg E_0$. In this case we obtain:

$$d[S:E]/dt = 0, \text{ i.e., } k_1[S][E] - k_2[S:E] - k_3[S:E] = 0. \tag{11}$$

Note that the right hand side of (8) is the complement of the right hand side of (9). Adding them we get that $d[E]/dt + d[S:E]/dt = 0$. Equivalently,

$$[E] + [S:E] = E_{tot}, \text{or equivalently } [E] = E_{tot} - [S:E], \tag{12}$$

where E_{tot} is constant, standing for the total amount of enzyme in the system, either free or as part of the substrate-enzyme complex.

Considering the *quasi-steady state* assumption and (12), equation (11) can be rewritten as follows:

$$k_1[S]E_{tot} = k_1[S][S:E] + k_2[S:E] + k_3[S:E], \text{ i.e.,}$$

$$[S:E] = \frac{k_1[S]E_{tot}}{k_1[S] + k_2 + k_3}, \text{ i.e.,}$$

$$[S:E] = \frac{[S]E_{tot}}{[S] + \frac{k_2+k_3}{k_1}} \tag{13}$$

Introducing (13) into (10) yields the result

$$\frac{d[P]}{dt} = \frac{k_3[S]E_{tot}}{[S] + \frac{k_2+k_3}{k_1}}. \tag{14}$$

The Michaelis-Menten equation relates the reaction rate v of synthesizing the product P to the concentration of the substrate, $[S]$, by the relation:

$$v = \frac{d[P]}{dt} = \frac{V_{max}[S]}{[S] + K_m}, \tag{15}$$

where V_{max} represents the maximum rate achieved by the system, for saturated values of $[S]$. The Michaelis constant K_m is the concentration of substrate for which the reaction rate is half-maximal. Identifying the parameters of (15) into (14) yields the connection between the Michaelis-Menten kinetics and the mass-action deduced kinetics of an enzymatic reaction:

$$V_{max} = k_3 E_{tot}, \quad K_m = \frac{k_2 + k_3}{k_1}.$$

Assuming the *quasi-equilibrium*, the quantity k_3/k_1 is negligible, thus $K_m \cong k_2/k_1$. Figure 2 shows the dependency of the reaction rate v with $[S]$. For more details on Michaelis-Menten kinetics, we refer the reader to [37].

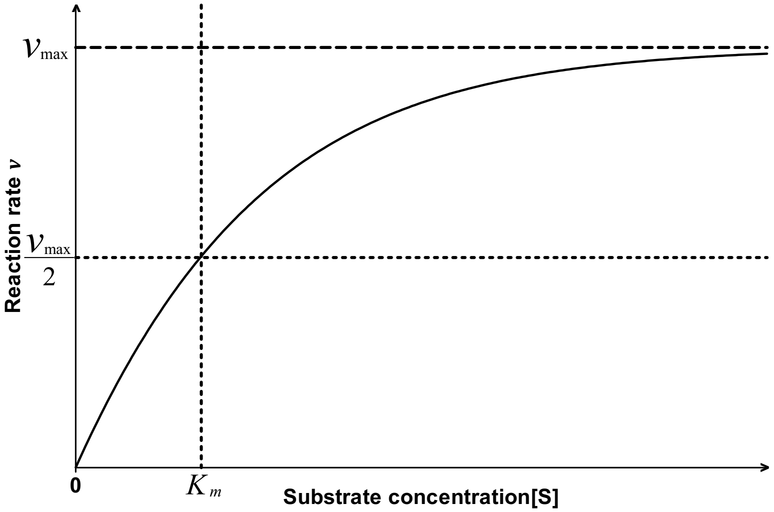


Fig. 2. Dependency of the reaction rate v with $[S]$ for Michaelis-Menten kinetics. v_{max} represents the maximum velocity, and K_m is the concentration of substrate for which the reaction rate is half-maximal.

Reversible Michaelis-Menten kinetics. Enzyme kinetics can often be reversible, and the Michaelis-Menten equation can be extended to a reversible reaction of the form

$$S + E \underset{k_2}{\overset{k_1}{\rightleftarrows}} X \underset{k_4}{\overset{k_3}{\rightleftarrows}} P + E, \tag{16}$$

where S and P are substrates, E is the enzyme, and X represents the intermediary enzyme-substrate compound. The mass-action irreversible reactions describing this system are:

$$\begin{aligned} S+E &\xrightarrow{k_1} X; \\ X &\xrightarrow{k_2} S+E; \\ X &\xrightarrow{k_3} P+E; \\ P+E &\xrightarrow{k_4} X. \end{aligned} \tag{17}$$

The system of ODEs describing the dynamics of the reaction-based model (17) is:

$$\frac{d[S]}{dt} = -k_1[S][E] + k_2[X]; \tag{18}$$

$$\frac{d[E]}{dt} = -k_1[S][E] + k_2[X] + k_3[X] - k_4[P][E]; \tag{19}$$

$$\frac{d[X]}{dt} = k_1[S][E] - k_2[X] - k_3[X] + k_4[P][E]; \tag{20}$$

$$\frac{d[P]}{dt} = k_3[X] - k_4[P][E]. \tag{21}$$

Following the reasoning for simple Michaelis-Menten equations, adding (19) and (20) yields

$$\frac{d[E]}{dt} + \frac{d[X]}{dt} = 0 \Rightarrow [E] + [X] = E_{tot}.$$

For the *quasi-steady state*, $d[X]/dt = 0$, i.e., $k_1[S](E_{tot} - [X]) - [X](k_2 + k_3) + k_4[P](E_{tot} - [X]) = 0$, which leads to

$$[X] = \frac{k_1[S]E_{tot} + k_4[P]E_{tot}}{k_1[S] + k_4[P] + k_2 + k_3}. \tag{22}$$

Introducing (22) into equation (21), after a few computations the formula reads

$$v = \frac{k_1k_3[S]E_{tot} - k_2k_4[P]E_{tot}}{k_1[S] + k_4[P] + k_2 + k_3} = \frac{k_3E_{tot}\frac{k_1[S]}{k_2+k_3} - k_2E_{tot}\frac{k_4[P]}{k_2+k_3}}{1 + \frac{k_1[S]}{k_2+k_3} + \frac{k_4[P]}{k_2+k_3}} = \frac{\frac{V_{fw}}{K_{mS}}[S] - \frac{V_{bw}}{K_{mP}}[P]}{1 + \frac{[S]}{K_{mS}} + \frac{[P]}{K_{mP}}},$$

where $K_{mS} = (k_2 + k_3)/k_1$ and $K_{mP} = (k_2 + k_3)/k_4$ are the Michaelis-Menten constants (i.e. for half-maximal forward and backward rate) for the substrate and product, respectively, and $V_{fw}(V_{bw})$ denotes the maximal rate in forward (backward) direction. An exact solution to this equation can be found in [44]. For details on the reversible Michaelis-Menten kinetics, we refer the reader to [22].

Other kinetic laws. Mass action and Michaelis-Menten are not the only existing kinetics. Some enzymatic reaction can follow Hill kinetics, Goldbeter-Koshland kinetics, or be subject to inhibition. We only introduce them briefly, discussing the types of reactions that are typically modeled in this way, and skipping the derivation of their mathematical formulations.

Goldbeter-Koshland kinetics, introduced in [18], applies to reversible reactions from substrate to product and back, catalyzed by different enzymes (e.g. phosphorylation and

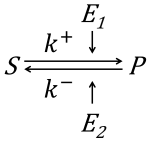


Fig. 3. Goldbeter-Koshland kinetics. P is produced from S in presence of enzyme E_1 and S is produced from P in presence of enzyme E_2.

dephosphorylation of proteins). The forward and backward reactions have Michaelis-Menten kinetics. The general form of such reactions is shown in Figure 3.

Hill kinetics, introduced in [29], are suitable for reactions where the enzyme can bind more molecules from the substrate S. Usually, the binding of the first S molecule changes the binding rate of the second molecule. The rate can either increase (called *positive cooperativity*), or decrease (called *negative cooperativity*). A general form of such reactions is the following:

$$E + S_1 \underset{k_1^-}{\overset{k_1^+}{\rightleftarrows}} ES_1;$$

$$ES_1 \xrightarrow{k_2} P + E;$$

$$ES_1 + S_2 \underset{k_3^-}{\overset{k_3^+}{\rightleftarrows}} ES_1S_2;$$

$$ES_1S_2 \xrightarrow{k_4} P + E.$$

Inhibition in a system with Michaelis-Menten kinetics (see (16)) can occur at different levels. An inhibitor I can bind to an enzyme in different states of the enzyme. When it binds (in a reversible reaction) to the free enzyme, the inhibition is called *competitive*, as both the substrate and the inhibitor are competing for binding the enzyme. When I binds reversibly to the enzyme-substrate complex, the reaction is called *uncompetitive inhibition*, as the enzyme is already bound to the substrate. When the inhibitor binds both to the free enzyme and to the enzyme-substrate complex, the inhibition is called *noncompetitive*. For a more detailed description of enzyme inhibition reactions, we refer the reader to [37].

4 Parameter Estimation

We discuss in this section the parameter estimation problem, including aspects of model identifiability, quantitative measures for model fit quality, model validation, and methods for model fitting.

4.1 Generalities: Relating the Mathematical Model to the Experimental Data

Relating the mathematical model to the experimental data is an essential step in the process of model building. This includes the validation of the model in terms of how well it can explain existing (quantitative or qualitative) experimental data and how well its predictions correspond to existing (non-quantitative) knowledge. There are several ways to approach this problem.

1. The modeler might have no a-priori hypothesis regarding the mathematical form and is instead strongly guided by data. The focus here is to capture the trend of the data and to predict the behavior in-between the data points and the emphasis is on the data. This approach is called *interpolation*.
2. The modeler has a clear hypothesis regarding the mathematical form she is building. For example, she might start from a reaction-based model and then associate to it an ODE-based model with mass-action kinetics as discussed in Section 3. The focus here is on finding values for all model parameters and the emphasis is on the model. This approach is called *model fitting*.
3. The modeler might replace a fitted model with an interpolating curve because of a need for better mathematical properties in further manipulations/analsysis of the model. This approach is called sometimes *model approximation*.

We only focus in this section on aspects related to model fitting. For a basic introduction to other approaches we refer to [17].

The main focus in model fitting is on the estimation of the unknown kinetic parameters of the model so that its predictions are consistent with a given set of data, usually presented in terms of time series. This step is often followed by a model validation step, where the model is compared with another set of data, that was not used in the fitting stage. In both cases, the task can be formulated as a mathematical optimization problem to minimize a cost function that quantifies the differences between the model predictions and the experimental measurements. The cost function can be seen as a distance measure between two vectors with non-negative real numbers as entries, one holding the experimental data, the other the model prediction for the time points where the data was collected. Some of the most widely used cost measures in this context are based on the *Chebyshev criterion*, *sums of absolute deviations*, and *least-squares*. We introduce briefly each of them in the following. In all cases, we are given a data set (x_i, y_i), $1 \le i \le m$ and a model $y = f(k,x)$, where $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ and $k \in \mathbb{R}^n$ is the vector of parameters, often with non-negative values.

In the Chebyshev criterion, the goal is to find $k \in \mathbb{R}^n$ that minimizes $\max\{|y_i - f(k,x_i)|, 1 \le i \le m\}$. In other words, the goal is to *minimize the largest absolute deviation* of a model value from the corresponding experimental value. The effect is that more weight is given to the worst outlier.

Another approach is to find $k \in \mathbb{R}^n$ that minimizes $\sum_{1 \le i \le m} |y_i - f(k,x_i)|$. In other words, the goal is to *minimize the sum of absolute deviations*. The effect is to treat each data point equally and to average the deviations over all experimental points.

In the third approach we mention here, the goal is to find $k \in \mathbb{R}^n$ that minimizes $\sum_{1 \le i \le m} |y_i - f(k,x_i)|^2$. This is the most widely used criterion in model fitting because the resulting optimization problem can be approached using calculus if f is a differentiable function (such as those obtained through the methods in Section 3).

The problem of estimating the parameters of kinetic models in systems biology is computationally difficult, see e.g., [4,42,45]. Regardless of which fitting criterion (score function) is used, the high number of variables in a typical biomodel makes an exact solution to the problem unfeasible in practice. There are however many approximation methods. Some of them are based on local approximation algorithms; they are faster in practice, but tend to converge to local optima. Others are based on global optimization algorithms; they are in general slower, but tend to converge to a global optimum. The global optimization methods can be based on deterministic searches [19, 33] or on stochastic ones [2, 6]. Even though the deterministic methods guaranty the convergence to a global optimum, the speed of the convergence is typically a major concern and in general, these methods cannot ensure the termination of the algorithm within a given finite time interval [45]. On the other hand, the intrinsic randomness of the stochastic approaches does not guarantee their convergence to an optimum [45]. However, many stochastic methods exhibit a good performance in practice – they are often capable of efficiently identifying a point in the vicinity of global solutions, see [45].

There are many modeling software environments, some commercial, others offering free access, that are used for model fitting. In most of our projects we chose COPASI [31] as a computational environment for parameter estimation. This software is a widely used tool in computational systems biology, having a documented good performance, see [4, 42, 45]. It includes a suite of various local and global, deterministic and stochastic parameter estimation algorithms, such as simulated annealing, genetic algorithms, evolution strategy using stochastic ranking, and particle swarm.

4.2 Alternative Model Fits and Model Identifiability

The problem of *model identifiability* adds to the difficulty of model fitting; it has to do with a model having several (sometimes very) different sets of parameter values, all yielding good model fits. The problem is that some numerical properties of the model, such as sensitivity coefficients, might be drastically different in different numerical setups, even if they all fit well the available data. This implies that there exist several models (or model setups) offering equally good, but different explanations for the available data. In such a situation, additional data is needed, focusing on the domains where the candidate models exhibit different behavior.

Even when only one model fit has been achieved, the modeler should evaluate the uniqueness of the parameter set. One way of doing this is to repeat the parameter estimation procedure, using some other available algorithms but the same data set. Such a procedure can in principle yield several different results, as demonstrated e.g. in [8, 53].

When searching for alternative numerical model fits, one can sample the distribution of the score functions measuring the distance between the model predictions through a simultaneous sampling on the range of all parameter values. For each parameter, one can generate a large sample, e.g. through partitioning its value range into a large number of equal sized subintervals (say, on the scale of tens of thousands) and randomly select a value from each of them. For all combinations of values for all parameters, one can then calculate the score of the model fit and thus sample the distribution of the score function. However, the direct implementation of this idea is clearly intractable for models with more than a few parameters due to the combinatorial explosion of the number of model simulations that need to be run. A fast, practical solution to this problem is the *Latin Hypercube Sampling* method (LHS) of [41]. This is a method to generate samples which are uniformly distributed over each parameter space, with the number of samples being independent of the number of parameters, see [26,27,50] for several applications. Let p be the number of parameters. The first step is to choose the size of the sample, N; this will also serve as the number of samples for each parameter. The range of each parameter is then partitioned into N intervals, with the length of each interval proportional to the probability of the parameter's value to fall in that interval; in particular, if the parameter is uniformly distributed in its range, then all subintervals are equal-sized. We then randomly select a value from each subinterval to generate a sample of N values for each parameter. The N values for parameter i are then stored on the i-th column of an $N \times p$ matrix. Finally, we randomly shuffle the values on all the columns of the matrix. The result is read from the matrix row-by-row, giving a sample of N combinations of parameter values. For a detailed description of this sampling scheme we refer to [41]. We discuss this method in the case of the heat shock response model in Section 6.

4.3 Fit-Preserving Model Refinement

Altering an already-fitted model, for example by adding a new component to it, replacing a module with another one, or adding new variables and reactions to it, will lead to losing its numerical fit. The problem is especially difficult in cases where the number of parameters in the new model is much larger than in the starting model. Rather then attempting to re-estimate all parameters, including those that were already fitted in the starting model, a computationally more efficient way is to build the larger model in an iterative way, ensuring in each step that its quantitative model fit is preserved. This method is called *quantitative model refinement* and has already been investigated in several different setups in [3,34,46]. We follow here the presentation of [34].

A given reaction-based model can undergo several types of refinement, for instance depending whether the focus lies on the reactants or reactions of the model. If one's focus lies on model's data, then the model could be refined so as to include more details regarding a species by having it substituted for several of its subspecies. The main interest in this type of refinement originates in the analysis of the possible behavioral intricacies the model refined as such would depict. This type of refinement is called *data refinement* and it consists in refining a set of variables so as to include more details about their internal states, attributes, etc. If the interest lies on the reactions of the model, one could refine the model by replacing for instance a reaction in the model describing a certain process by a set of reactions detailing on some intermediate steps of the process. This type of refinement is called *process refinement*.

Formal refinement arose from the field of software engineering as a necessity to embed an elementary set of specifications in a system's final implementation. The problem of quantitative model refinement has been addressed before in systems biology in particular related to rule-based modeling, which integrates *data refinement* through the notion of agent resolution ([23]). The focus lies here on rule refinement, a method designed to refine rules ensuring model fit preservation. Nevertheless, the refinement technique must preserve the quantitative systemic properties of the model, such as numerical fit and validation, see [46].

A model M consisting of a set of reactions of the form (1) could be formalized through a discrete or continuous approach, a deterministic or non-deterministic evolution, etc. This discussion focuses on a continuous mass-action formulation. To each variable $S_i, 1 \le i \le m$ we associate a time-dependent function $[S_i] : \mathbb{R}_+ \to \mathbb{R}_+$, which denotes the concentration of the species over time. According to the principle of mass action, see [37], the time evolution of the system may be specified by a system of ODEs as follows:

$$\frac{d[S_i]}{dt} = -\sum_{j=1}^{n}\left(k_j c_{i,j}\prod_{l=1}^{m}[S_l]^{c_{l,j}}\right) + \sum_{j=1}^{n}\left(k_j c'_{i,j}\prod_{l=1}^{m}[S_l]^{c'_{l,j}}\right), \quad 1 \le i \le m. \tag{23}$$

Assume now model M is refined discerning among various subspecies of S_1. The distinction among the subspecies of S_1 can be made by either different classes of S_1 or several biochemical configurations of S_1, as a result of various post-translational modifications such as acetylation, phosphorylation, etc. All subspecies characterized as such participate in all reactions S_1 took part in (in model M), with possible variations in the kinetics. Replacing species S_1 in model M by subspecies $V_1, \ldots, V_l$ brings about a new model M_R, whose set of species consists of the new variables $\{S'_2, S'_3, ..., S'_m\} \cup \{V_1, ..., V_l\}$, for some $l \ge 2$, where variables S'_j, $2 \le j \le m$ of M_R, match variables S_j of model M and $V_1, ..., V_l$ replace species S_1 in M_R. Moreover, each reaction R_j of model M is substituted for in model M_R by a reaction R'_j as follows:

$$\begin{aligned} R'_j \; : \; & (T_{1,j}V_1 + ... + T_{l,j}V_l) + c_{2,j}S'_2 + ... + c_{m,j}S'_m \xrightarrow{k'_j} \\ & (T'_{1,j}V_1 + ... + T'_{l,j}V_l) + c'_{2,j}S'_2 + ... + c'_{m,j}S'_m, \end{aligned}$$

where k'_j is the kinetic rate constant, and $T_{1,j}, ..., T_{l,j}, T'_{1,j}, ..., T'_{l,j}$ are nonnegative integers so that $T_{1,j} + ... + T_{l,j} = c_{1,j}$ and $T'_{1,j} + ... + T'_{l,j} = c'_{1,j}$.

Model M_R is a *data refinement of model M on variable* S_1 if and only if the subsequent conditions hold, see [34]:

$$[S_j](t) = [S'_j](t), \text{ for all } 2 \le j \le m, \tag{24}$$

$$[S_1](t) = [V_1](t) + \ldots + [V_l](t), \text{ for all } t \ge 0. \tag{25}$$

The refined model, M_R, involves a number of $m + l - 1$ species, while model M comprises only m species, M_R evolving linearly in the size of its data set. The number of reactions in M_R substituting for reaction R_j of M is the number of non-negative integer solutions of the subsequent system of equations:

$$\begin{aligned} T_{1,j} + T_{2,j} + ... + T_{l,j} &= c_{1,j}; \\ T'_{1,j} + T'_{2,j} + ... + T'_{l,j} &= c'_{1,j}; \end{aligned}$$

over the independent unknowns $T_{k,j}, T'_{k,j}, 1 \le k \le l$. The number of solutions of the first equation is given by the *multinomial coefficient* "l multichooses $c_{1,j}$", see [13]:

$$\left(\!\!\binom{l}{c_{1,j}}\!\!\right) = \binom{l + c_{1,j} - 1}{c_{1,j}} = \frac{(l + c_{1,j} - 1)!}{c_{1,j}!\,(l-1)!}.$$

Some values for the new kinetic parameters of M_R may be attained from the literature or they can be estimated experimentally. The parameters not attained as such require calculation through computational methods so that conditions (24) and (25) are fulfilled. The reiteration of the parameter estimation process is however computationally expensive. As an alternative, the method proposed in [34] describes an approach for setting the values of the unknown parameters in the refined model so that relations (24) and (25) hold. The approach promotes a choice of parameters symmetrical in $V_1, ..., V_l$.

4.4 Quantitative Measures for the Model Fit Quality

Given parameter estimation may yield several different outputs, depending on the methods that were used in the fitting, it is important to quantify the goodness of a model fit. In this way, the results of different parameter estimation rounds can be compared. Moreover, through a suitable normalization, even the fitting of different models, using different sets of data, may also be compared. Part of the challenge here is to avoid to discriminate against models deviations that may be large in absolute values, but relatively small compared to the experimental data.

We discuss here briefly a notion of model fit quality introduced in [38]. Their fit quality only takes into account one set of experimental data at a time and aims to give a measure of the average deviation of the model from the data, normalized on the scale of the numerical values of the model predictions. For a given experimental data set $\mathcal{E} = \{(x_i, y_i) \mid 1 \leq i \leq n\}$ and a model $M = f(k,x)$, the quality of M's fit with respect to $\mathcal{E}$ is denoted as $q(M,\mathcal{E})$ and is defined as follows:

$$q(M,\mathcal{E}) = \frac{\sqrt{\sum_{i=1}^{n}(f(k,x_i) - y_i)^2/n}}{\sum_{i=1}^{n} f(k,x_i)/n} \cdot 100\%.$$

It was argued in [38] that a low (say, lower than $15-20\%$) value of $q(M,\mathcal{E})$ could be considered as an indicator of a successful fit. We discuss the quality of the best fit for the heat shock response model in Section 6 and refer to [10] for more details on applying this measure.

5 Analysis of ODE-Based Models

We discuss in this section several computational analysis techniques for ODE-based models. We apply some of these techniques in the next section, on the heat shock response model.

5.1 Steady State Analysis

Steady states (also called *stationary states*, *fixed points*, *equilibrium points*) have the property that when taken as initial values for the model, they yield a constant dynamics; in other words, there is no change in the concentration of any of the species when starting from steady state values. This is one of the basic concepts in dynamical systems theory, extensively employed in modeling biological systems. There are several types of steady states: *stable*, *asymptotically stable*, *unstable* etc.

Consider a dynamical system $dx/dt = f(x(t))$, $x(0) = x_0$, where $f : \mathbb{R}^n \to \mathbb{R}^n$ is a continuous function with equilibrium point x_e. The equilibrium is *stable* if for every $\varepsilon > 0$ there exists δ_ε such that, if $||x_0 - x_e|| < \delta_\varepsilon$, then $||x(t) - x_e|| < \varepsilon, \forall t \geq 0$. A steady state is called *asymptotically stable* if there exists $\delta > 0$ such that if $||x_0 - x_e|| < \delta$, then $\lim_{t\to\infty} ||x(t) - x_e|| = 0$. A steady state is *unstable* if the conditions for stability are not met.

For a reaction-based model, the steady state behavior is characterized by the equation

$$\frac{d[S]}{dt} = 0,$$

or equivalently, considering Equation (4),

$$Nv = 0. \qquad (26)$$

The rate vector v that satisfies the steady state condition (26) can be obtained by solving the corresponding system of algebraic equations with the variables $[S_1], [S_2], \ldots, [S_m]$. The equation has nontrivial solutions (not all variables are zero) only if $\operatorname{rank}(N) < n$, where n is the number of reactions in the system, i.e. matrix N contains at least one pair of linearly dependent columns. The dependencies can be expressed by a so-called kernel matrix K, such that

$$NK = 0, \qquad (27)$$

where K has $c = n - \operatorname{rank}(N)$ columns. The columns k_i of matrix K are the vectors that span the null space (also termed kernel) of N, i.e. the subspace of the reaction rates space that contains all solutions to Equation (26), see [24]. Consequently, any vector J of steady-state fluxes can be expressed as a linear combination of K's columns,

$$J = \sum_{i=1}^{c} \alpha_i k_i.$$

The kernel matrix K is not uniquely determined. Another kernel matrix K' could be obtained for example by a multiplication $K' = KQ$, where Q has dimensions $[n - \operatorname{rank}(N)] \times [n - \operatorname{rank}(N)]$. Since K is a solution to Equation (27), so is K'. For details on how to determine the kernel matrix using Gauss's algorithm, we refer the reader to [37].

Example 5. Consider the following system of reactions:

$$2A \xrightarrow{k_1} B;$$
$$A + B \underset{k_2^-}{\overset{k_2^+}{\rightleftarrows}} C.$$

To compute the steady state, one needs to solve Equation (26), which reads as the following system of algebraic equations:

$$\underbrace{\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}}_{\mathbf{0}} = \underbrace{\begin{bmatrix} -2 & -1 \\ 1 & -1 \\ 0 & 1 \end{bmatrix}}_{\mathbf{N}} \cdot \underbrace{\begin{bmatrix} v_1 \\ v_2 \end{bmatrix}}_{\mathbf{v}}.$$

Considering mass action kinetics and denoting by $[A]_0, [B]_0, [C]_0$ the steady-state concentrations for the species in the model, the system reads:

$$\begin{aligned}
&-2k_1[A]_0^2 - k_2^+[A]_0[B]_0 + k_2^-[C]_0 = 0,\\
&k_1[A]_0^2 - k_2^+[A]_0[B]_0 + k_2^-[C]_0 = 0,\\
&k_2^+[A]_0[B]_0 - k_2^-[C]_0 = 0.
\end{aligned}$$

Solving the steady-state system of equations gives the solution $[A]_0 = 0, [B]_0 = \alpha, [C]_0 = 0$, where $\alpha > 0$ is arbitrary.

Example 6. Let us consider the Lotka-Volterra model expressed in Table 1. The ODEs characterising the system's dynamics are expressed in Equation (5). The steady state analysis leads to the system

$$\begin{aligned}
&k_1[Prey] - k_2[Prey][Predator] = 0,\\
&k_2[Prey][Predator] - k_3[Predator] = 0.
\end{aligned}$$

Solving this system of two equations gives the steady state points

$$([Prey]_s, [Predator]_s) \in \{(0,0), (k_3/k_2, k_1/k_2)\}.$$

To study the behavior of the Lotka-Volterra model around the steady states, one needs to examine the behavior of the concentrations around each equilibrium point, i.e. their tendency to increase or decrease. To do that, one studies the sign of the derivatives:

$$\begin{aligned}
&\tfrac{d[Prey]}{dt} \geq 0 \Rightarrow k_1 - k_2[Predator] \geq 0 \Rightarrow [Predator] \leq \tfrac{k_1}{k_2};\\
&\tfrac{d[Predator]}{dt} \geq 0 \Rightarrow k_2[Prey] - k_3 \geq 0 \Rightarrow [Prey] \geq \tfrac{k_3}{k_2}.
\end{aligned} \tag{28}$$

The behavior around the steady states is depicted in Figure 4.

5.2 Mass Conservation Relations

In this section we introduce mass conservation relations and their importance in modeling reaction-based systems. For a more detailed presentation and additional examples we refer to [24].

Identifying the mass conservation relations in a given model is one of the first analyzes that a modeler typically performs. It gives an insight into the dynamics of the model, but at the same time it reduces the number of free variables in the model. Mathematically, a mass conservation relation is a linear combination of concentrations of species that is constant in time:

$$g^T S = C, \tag{29}$$

where g is a vector with some constant entries, S is the species concentrations vector, and C is some constant. An implication of mass conservation relations is that some of the stoichiometric matrix rows are linearly dependent, i.e.

$$g^T N = 0^T. \tag{30}$$

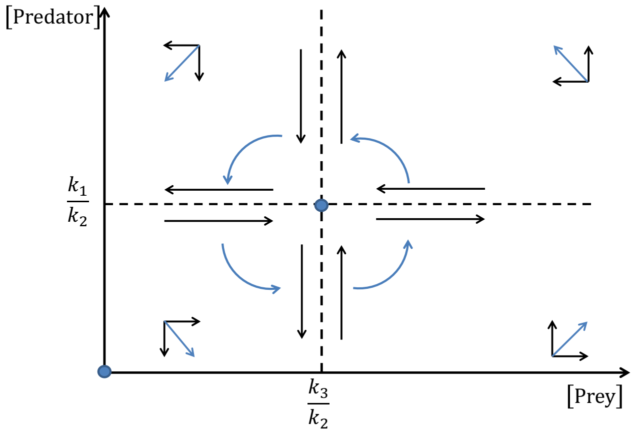


Fig. 4. Steady-state analysis of the Lotka-Volterra system. The blue dots are the two steady states of the system. The black arrows indicate how the concentration of each species increases or decreases (as established in (28)). The blue arrows are the combination of *Predator* and *Prey* concentrations tendencies, and they show how the dynamics of the system changes between the four areas delimited by the dotted lines. The behavior around the $(k_3/k_2, k_1/k_2)$ point suggests periodicity; this is confirmed by Figure 1. Both equilibrium points are unstable, as indicated by the blue arrows.

Equations (29) and (30) are equivalent. Derivating the former equation and taking into account Equation (4) yields

$$(g^T S)' = g^T \dot{S} = g^T N v = 0.$$

There may be more linearly independent vectors g that satisfy Equation (30), each denoting a different mass conservation relation. The number of mass conservation relations is given by $m-$ rank(N), where m is the number of species in the system. The full set of vectors g describing these mass conservation relations form a so-called *conservation matrix G* , see [24], with the property

$$GN = 0.$$

Consequently, G^T is a kernel matrix for N^T. A conservation matrix G can be determined using the Gauss algorithm, and it is not unique (any other matrix $G' = PG$, where P is any nonsingular matrix of appropriate dimensions, is a valid conservation matrix).

Example 7. Consider the following system of biochemical reactions:

$$\begin{array}{l} 2A \rightleftarrows A_2; \\ A_2 + B \rightleftarrows A_2 : B; \\ A_2 : B \rightarrow C + A_2 : B; \\ C \rightarrow \emptyset. \end{array}$$

The species vector, stoichiometric coefficients matrix and the conservation matrix read:

$$S = \begin{bmatrix} A \\ A_2 \\ B \\ A_2 : B \\ C \end{bmatrix}, \quad N = \begin{bmatrix} -2 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}, \quad G = \begin{bmatrix} 1 & 2 & 0 & 2 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}.$$

The mass conservation relations induced by G are:

$$\begin{aligned} &[A] + 2[A_2] + 2[A_2 : B] = C_1, \\ &[B] + [A_2 : B] = C_2, \end{aligned} \tag{31}$$

for some constants C_1, C_2.

The mass conservation relations are important for reducing the system of differential equations $\dot{S} = Nv$ that describe the dynamics of the model. Each mass conservation relation introduces one dependent variable, which can be expressed in terms of the independent variables, and thus eliminated from the system of ODEs. The two mass conservation relations in Equation (31) could be used to express the dependency between $[A], [B]$ and the rest of the species concentrations:

$$\begin{aligned} &[A] = C_1 - 2[A_2] - 2[A_2 : B], \\ &[B] = C_2 - [A_2 : B]. \end{aligned}$$

This reduces the initial system of ODEs from 5 to 3 equations.

5.3 Sensitivity Analysis

Sensitivity analysis is a method of estimating the changes that small perturbations in the parameters of a model induce in the system. With this type of analysis, one can estimate the robustness of a model against small changes, and also identify ways of inducing a desired change into the model. There exist many methods for sensitivity analysis, some suitable for spatially homogeneous constant-parameter reaction-based models, others suitable for systems with space- and time-dependent parameters, or stochastic models. For a review of multiple methods, we refer the reader to [59, 62]. One of the questions often encountered in biochemical systems is what changes should the system undergo such that the new steady state satisfies certain properties.

There are two types of sensitivity analysis: *local sensitivity analysis*, and *global sensitivity analysis*. In the global approach, all parameters are varied at once, and the sensitivity is measured over the entire range of each parameter. In the local analysis, only one parameter is varied at a time, within a small interval around some nominal value. Generally, it is assumed that input-output relationships are linear. We only focus here on local sensitivity analysis.

We consider the system of ODEs describing a system to be expressed as a function of the concentrations of all species and all the parameter values:

$$\frac{d[S_i]}{dt} = f_i([S_1], [S_2], ..., [S_m], \kappa), \tag{32}$$

where $\kappa = (k_1, k_2, ..., k_n)^T$ is the rate constants vector (assuming without loss of generality that the system comprises n irreversible reactions). Let $\mathcal{S}(t,\kappa) = ([S_1](t,\kappa), [S_2](t,\kappa), ..., [S_m](t,\kappa))^T$ be the solution of Equation (32) with respect to κ, also called *sensitivity matrix*. The elements of the matrix are the partial derivatives $\partial[S_i]/\partial k_j$, also called *first-order local sensitivity coefficients*.

There are many ways of determining the local sensitivity of the concentrations. The simplest method is the *brute force method* (also called *indirect method*, or *finite-difference method*), that uses the finite difference approximation. The j-th parameter, k_j, changes with the amount δk_j at time point t_1, and all other parameters remain unchanged. One can compute the new matrix $[S]$ using the change between the initial and the perturbed solution, see Equation (33). The method requires $n+1$ runs, one for the initial values of the parameters and n modifying each of the parameters at a time.

$$\frac{\partial[S](t_2)}{\partial k_j(t_1)} = \frac{[S](t2, k_j + \delta k_j) - [S](t_2, k_j)}{\delta k_j}, 1 \leq j \leq n. \tag{33}$$

This method is widely used because of its simplicity, but other more efficient methods exist, e.g. the *direct method*. This method solves the differential equations for the sensitivity coefficients $\partial[S_i]/\partial k_j$, by differentiating Equation (32). This results in the following set of sensitivity equations:

$$\frac{d}{dt}\frac{\partial[S]}{\partial k_j} = \mathcal{J}\frac{\partial[S]}{\delta k_j} + \frac{\partial f}{\partial k_j}, 1 \leq j \leq n,$$

where $\mathcal{J}$ is the Jacobian for Equation (32). For a complete mathematical derivation of this result, see [62].

Perturbations should be small enough to yield small errors in the indirect method, and large enough to surpass the simulation inaccuracies of ODE solvers, for the direct method, see [62]. Other methods of computing the sensitivity of a model to parameter changes exist, e.g. the Green function method, polynomial approximation method, AIM method, detailed in [54, 59].

Very often, sensitivity analysis is focused on the steady states, when concentrations are constant. In this case, the sensitivity coefficients are computed as solutions to the system

$$\frac{d}{dt}\frac{\partial[S]}{\partial k_j} = 0,$$

and reflect the dependency of the steady state on the parameters. If the steady state is asymptotically stable, then one can consider the limit $\lim_{t\to\infty}(\partial[S]/\partial k_j)(t), 1 \leq j \leq n$, called *stationary sensitivity coefficients*. The system can be written as

$$\frac{\partial[S]}{\partial k_j} = -\mathcal{J}F_j, 1 \leq j \leq n,$$

where $\mathcal{J}$ is the value of the jacobian at steady state, and F_j is the j-th column in the matrix $F = (\partial f_r/\partial k_s)_{m\times n}$ computed at steady state. Sensitivity coefficients can be computed in many software applications, e.g. in COPASI [31].

6 The Heat Shock Response Model

We consider in this section a larger modeling case-study to which we apply some of the techniques discussed in the chapter. The eukaryotic heat shock response is an evolutionarily conserved bio-regulatory network, crucial to cell survival. It acts as a defence mechanism that regulates the cellular response to proteotoxicity induced by diverse physiological and environmental stressors, such as elevated temperatures. Exposure of proteins to elevated temperatures causes protein misfolding, which results in the constitution of large aggregates that eventually induce apoptosis (controlled cell death). Protein homeostasis is promoted by augmenting the level of molecular chaperons.

6.1 The Reaction-Based Model

We consider here the basic molecular model for the heat shock response introduced in [53]. Elevated temperatures cause protein misfolding and accumulation of misfolded proteins in large conglomerates that induce cell death. The key role in homeostasis restoration is played by *heat shock proteins* (hsp), which chaperone the misfolded proteins, promoting the folding of proteins. The transactivation of hsp-encoding genes regulates the heat shock response. Heat shock factors (hsf) activate gene transcription. In the absence of stress, heat shock factors are present in a monomeric conformation and they are bound to a great extent to heat shock proteins. However, heat stress actuates the dimerization (hsf_2) and consequently trimerization (hsf_3) of heat shock factors, a DNA binding-competent conformation. Due to their high affinity toward the heat shock element (hse), hsf trimers bind to the heat shock elements, promoting the transcription and translation of the gene. Consequently, DNA binding activates hsp synthesis, see [53,55].

Once the heat stress is removed, hsp synthesis is turned off as follows: hsp's sequestrate free hsf's (residing in the constitution of hsp: hsf complexes), break hsf_2 and hsf_3 and induce DNA unbinding, see [53,55]. Subsequently, DNA transcription is turned off and the formation of new hsf trimers repressed. The heat shock response mechanism is switched back on when the temperature is again elevated, impelling the proteins in the cell (prot) to misfold and hsp: hsf complexes to break down. The reactions of the molecular model in [53] are shown in Table 2.

Table 2. The molecular model for the eukaryotic heat shock response proposed in [53]

$2\,\mathsf{hsf} \rightleftarrows \mathsf{hsf}_2$

$\mathsf{hsf} + \mathsf{hsf}_2 \rightleftarrows \mathsf{hsf}_3$

$\mathsf{hsf}_3 + \mathsf{hse} \rightleftarrows \mathsf{hsf}_3\!:\!\mathsf{hse}$

$\mathsf{hsf}_3\!:\!\mathsf{hse} \rightarrow \mathsf{hsf}_3\!:\!\mathsf{hse} + \mathsf{hsp}$

$\mathsf{hsp} + \mathsf{hsf} \rightleftarrows \mathsf{hsp}\!:\!\mathsf{hsf}$

$\mathsf{hsp} + \mathsf{hsf}_2 \rightarrow \mathsf{hsp}\!:\!\mathsf{hsf} + \mathsf{hsf}$

$\mathsf{hsp} + \mathsf{hsf}_3 \rightarrow \mathsf{hsp}\!:\!\mathsf{hsf} + 2\,\mathsf{hsf}$

$\mathsf{hsp} + \mathsf{hsf}_3\!:\!\mathsf{hse} \rightarrow \mathsf{hsp}\!:\!\mathsf{hsf} + 2\,\mathsf{hsf} + \mathsf{hse}$

$\mathsf{hsp} \rightarrow \emptyset$

$\mathsf{prot} \rightarrow \mathsf{mfp}$

$\mathsf{hsp} + \mathsf{mfp} \rightleftarrows \mathsf{hsp}\!:\!\mathsf{mfp}$

$\mathsf{hsp}\!:\!\mathsf{mfp} \rightarrow \mathsf{hsp} + \mathsf{prot}$

This molecular model is clearly on a high level of abstraction, for the sake of easing its analysis. For example, note that the eukaryotic cell presents various classes of heat shock proteins, denominated according to their molecular weight, e.g., Hsp60, Hsp70, Hsp90. However, in this molecular model, they are all referred to as belonging to the same class, with Hsp70 as common denominator. The same assumptions are made for hsf and hse. Furthermore, the model considers all proteins uniformly, distinguishing only between the ones that are correctly folded (prot) and the misfolded ones (mfp). The model contains also simplified representations of some cellular mechanisms, e.g., protein synthesis and degradation, see [53] for more details.

The molecular model in [53] satisfies the following three mass-conservation relations, for the total amount of hsf, the total amount of proteins (excluding hsp and hsf) and for the total amount of hse:

- $[\mathsf{hsf}] + 2[\mathsf{hsf_2}] + 3[\mathsf{hsf_3}] + 3[\mathsf{hsf_3\!:hse}] + [\mathsf{hsp\!:hsf}] = C_1$,
- $[\mathsf{prot}] + [\mathsf{mfp}] + [\mathsf{hsp\!:mfp}] = C_2$,
- $[\mathsf{hse}] + [\mathsf{hsf_3\!:hse}] = C_3$,

where C_1, C_2 and C_3 are constants.

6.2 The Mathematical Model

Given the molecular model in Table 2, we consider a mathematical model derived through the principle of mass action, formulated as a system of ordinary differential equations ([37]). The rate coefficient for protein misfolding (prot → mfp) is described by the following formula:

$$\varphi(T) = (1 - \frac{0.4}{e^{T-37}}) \cdot 1.4^{T-37} \cdot 1.45 \cdot 10^{-5} s^{-1},$$

where T is the temperature of the environment, expressed in $^\circ C$, in accordance to [52]. Each species X in the molecular model is associated to a continuous, time-dependent function $[X](t)$, expressing the concentration of the respective reactant. The dynamics of the system is described through the system of differential equations in Table 3.

The initial values of all species and the kinetic rate constants were estimated in [53], by imposing the following three conditions:

(i) At $37^\circ C$ the system is in a steady state, since the model should not reveal any response in the absence of the heat stress;
(ii) At $42^\circ C$, the numerical predictions for DNA binding ($[\mathsf{hsf_3\!:hse}](t)$) should be in accordance with the experimental data reported in [36];
(iii) At $42^\circ C$, the numerical prediction of the model for $[\mathsf{hsp}](t)$ should confirm the data obtained in [53] through a de-novo fluorescent reporter-based experiment.

The numerical setup obtained in [53] for the heat shock response model is shown in Table 4.

The estimation of parameters was based on the experimental data in [36] on DNA binding in HeLa cells for a temperature of $42^\circ C$. Moreover, the model should also be in a steady state at $37^\circ C$. Hence, seven more independent algebraic relations on the set of

Table 3. The system of ODE's associated with the biochemical model proposed in [53]

$$
\begin{aligned}
d[\mathsf{hsf}]/dt = {} & -2k_1^+[\mathsf{hsf}]^2 + 2k_1^-[\mathsf{hsf}_2] - k_2^+[\mathsf{hsf}][\mathsf{hsf}_2] + k_2^-[\mathsf{hsf}_3] \\
& - k_5^+[\mathsf{hsf}][\mathsf{hsp}] + k_5^-[\mathsf{hsp\colon hsf}] + k_6[\mathsf{hsf}_2][\mathsf{hsp}] \\
& + 2k_7[\mathsf{hsf}_3][\mathsf{hsp}] + 2k_8[\mathsf{hsf_3\colon hse}][\mathsf{hsp}]; \\
d[\mathsf{hsf}_2]/dt = {} & k_1^+[\mathsf{hsf}]^2 - k_1^-[\mathsf{hsf}_2] - k_2^+[\mathsf{hsf}][\mathsf{hsf}_2] + k_2^-[\mathsf{hsf}_3] \\
& - k_6[\mathsf{hsf}_2][\mathsf{hsp}]; \\
d[\mathsf{hsf}_3]/dt = {} & k_2^+[\mathsf{hsf}][\mathsf{hsf}_2] - k_2^-[\mathsf{hsf}_3] - k_3^+[\mathsf{hsf}_3][\mathsf{hse}] + k_3^-[\mathsf{hsf_3\colon hse}] \\
& - k_7[\mathsf{hsf}_3][\mathsf{hsp}]; \\
d[\mathsf{hse}]/dt = {} & -k_3^+[\mathsf{hsf}_3][\mathsf{hse}] + k_3^-[\mathsf{hsf_3\colon hse}] + k_8[\mathsf{hsf_3\colon hse}][\mathsf{hsp}]; \\
d[\mathsf{hsf_3\colon hse}]/dt = {} & k_3^+[\mathsf{hsf}_3][\mathsf{hse}] - k_3^-[\mathsf{hsf_3\colon hse}] - k_8[\mathsf{hsf_3\colon hse}][\mathsf{hsp}]; \\
d[\mathsf{hsp}]/dt = {} & k_4[\mathsf{hsf_3\colon hse}] - k_5^+[\mathsf{hsf}][\mathsf{hsp}] + k_5^-[\mathsf{hsp\colon hsf}] - k_6[\mathsf{hsf}_2][\mathsf{hsp}] \\
& - k_7[\mathsf{hsf}_3][\mathsf{hsp}] - k_8[\mathsf{hsf_3\colon hse}][\mathsf{hsp}] - k_{11}^+[\mathsf{hsp}][\mathsf{mfp}] \\
& + (k_{11}^- + k_{12})[\mathsf{hsp\colon mfp}] - k_9[\mathsf{hsp}]; \\
d[\mathsf{hsp\colon hsf}]/dt = {} & k_5^+[\mathsf{hsf}][\mathsf{hsp}] - k_5^-[\mathsf{hsp\colon hsf}] + k_6[\mathsf{hsf}_2][\mathsf{hsp}] \\
& + k_7[\mathsf{hsf}_3][\mathsf{hsp}] + k_8[\mathsf{hsf_3\colon hse}][\mathsf{hsp}]; \\
d[\mathsf{mfp}]/dt = {} & \varphi(T)[\mathsf{prot}] - k_{11}^+[\mathsf{hsp}][\mathsf{mfp}] + k_{11}^-[\mathsf{hsp\colon mfp}]; \\
d[\mathsf{hsp\colon mfp}]/dt = {} & k_{11}^+[\mathsf{hsp}][\mathsf{mfp}] - (k_{11}^- + k_{12})[\mathsf{hsp\colon mfp}]; \\
d[\mathsf{prot}]/dt = {} & -\varphi(T)[\mathsf{prot}] + k_{12}[\mathsf{hsp\colon mfp}].
\end{aligned}
$$

parameters and initial values are derived. Therefore, the model comprises 17 independent values that require estimation. The above-mentioned conditions are satisfied by the values in Table 4. These values have been attained by means of parameter estimation in COPASI [31]. The model is fit with regard to the DNA binding experimental data in [36]. The model predictions regarding $\mathsf{hsf_3\colon hse}$ compared with the experimental data of [36] are shown in Figure 5.

6.3 Model Validation

The model exhibits a very low rate for protein misfolding for a temperature of $37°C$ and a high rate for protein folding, in compliance with [5] and [35]. The model also predicts a transient increase in the level of hsf trimers, in accordance with [30]. The model confirms that dimers are only a transient form between monomers and trimers, and that the level of dimers is low throughout the simulation, regardless of the temperature.

Another validation test consisted in applying the heat shock response twice subsequently. The second heat shock was applied after the heat shock proteins had attained a maximal level. The model in [53] predicted the response to the second heat shock to be

Table 4. The numerical values of the parameters (A) and the initial values of the variables (B) of the heat shock response model proposed in [53]

A

Param.	Value	Units
k_1^+	3.49	$\frac{ml}{\#\cdot s}$
k_1^-	0.19	s^{-1}
k_2^+	1.07	$\frac{ml}{\#\cdot s}$
k_2^-	10^{-9}	s^{-1}
k_3^+	0.17	$\frac{ml}{\#\cdot s}$
k_3^-	$1.21 \cdot 10^{-6}$	s^{-1}
k_4	$8.3 \cdot 10^{-3}$	s^{-1}
k_5^+	9.74	$\frac{ml}{\#\cdot s}$
k_5^-	3.56	s^{-1}
k_6	2.33	$\frac{ml}{\#\cdot s}$
k_7	$4.31 \cdot 10^{-5}$	$\frac{ml}{\#\cdot s}$
k_8	$2.73 \cdot 10^{-7}$	$\frac{ml}{\#\cdot s}$
k_9	$3.2 \cdot 10^{-5}$	s^{-1}
k_{11}^+	$3.32 \cdot 10^{-3}$	$\frac{ml}{\#\cdot s}$
k_{11}^-	4.44	s^{-1}
k_{12}	13.94	s^{-1}

B

Variable	Initial conc.
[hsf]	0.67
[hsf$_2$]	$8.7 \cdot 10^{-4}$
[hsf$_3$]	$1.2 \cdot 10^{-4}$
[hse]	29.73
[hsf$_3$: hse]	2.96
[hsp]	766.88
[hsp: hsf]	1403.13
[mfp]	517.352
[hsp: mfp]	71.65
[prot]	1.15×10^8

greatly diminished in intensity. Indeed, a diminished response for the second heat shock could be anticipated since the level of heat shock proteins (hsp's) is already elevated as a consequence of the first heat shock. A similar result was reported in [52].

Another validation method consisted in simulating the model for a temperature of 43°*C* and comparing the results with those of [55]. The model in [53] predicts a prolonged transactivation for DNA binding, as opposed to the model in [55], but it is consistent with the experimental data in [1]. An experiment consisting in the removal of the heat shock at 42°*C* at the peak of the response exhibited an accelerated attenuation phase, complying with the results reported by [55].

An alternative verification scenario focused on the prediction of the evolution of heat shock proteins (hsp's) over time. This method required the use of a quantitative reporter system founded on *yellow fluorescent proteins* (yfp's). This method was based on the assumption that fluorescence intensity is virtually linear reported to the level of yfp's. As yfp's transactivation is regulated by their own heat shock elements, denoted in [53] by hse$'$, transcription and degradation kinetics ($k_4{}'$ and $k_9{}'$ respectively), their evolution in time may be described by the following differential equation:

$$d[\mathsf{yfp}]/dt = k_4'[\mathsf{hsf_3\colon hse'}] - k_9'[\mathsf{yfp}], \qquad (34)$$

for some positive constants k_4', k_9' accounting for the kinetic rate constants of yfp synthesis and of yfp degradation. The extended model, including equation (34), takes into

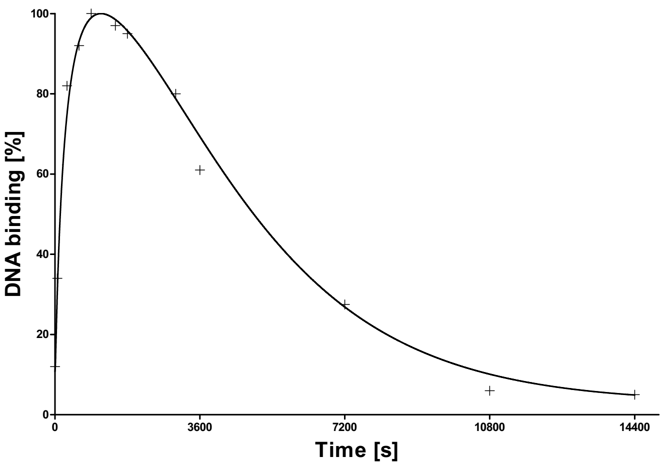


Fig. 5. The dynamic behavior of hsf3:hse in the best fitted model. The continuous line is the model prediction and the crossed points indicate the experimental data of [36].

account all numerical values from the basic model, described in Table 2, and the numerical values for the new rate constants $k_4{}'$ and $k_9{}'$ were estimated so that the fit for yfp's complies with the experimental data.

6.4 Model Analysis

Sensitivity analysis. The first analysis approach consisted in estimating the scaled steady state sensitivity coefficients, see [59], of all variables against reaction rate constants and initial concentrations. Given a variable X and a parameter p, the *scaled steady state sensitivity coefficient* of variable X against parameter p is defined by:

$$\lim_{t\to\infty} \frac{\partial ln(X)}{\partial \ln(p)(t)}.$$

The coefficients described above represent the relative variance of the steady state when the model undergoes infinitesimal changes in parameter p. The sensitivity coefficients of all variables against reaction rate constants $k_1{}^-, k_2{}^-, k_3{}^-, k_7$ proved to be all insignificant, suggesting that the reactions corresponding to those rate constants may not be crucial to the global behavior of the model. For this aim, the model was altered so as to exclude the reactions corresponding to the aforementioned kinetic rate constants, namely the backward reactions for dimerization, trimerization, DNA binding and DNA unbinding. The new model attained as such satisfies the validation tests described in Section 6.3. This suggests that hsf dimers and trimers are steady configurations and that non-hsp-mediated DNA unbinding is negligible. While the breaking of trimers (by hsp) does not affect greatly the overall behavior of the model, the reaction describing the breaking of trimers (by hsp) proved to have a substantial impact on the evolution of hsp and mfp.

The variation between the steady state levels of hsp and mfp are correlated, see [53], which is consistent with the biological knowledge that hsp's have a major role in chaperoning mfp's. Table 5 shows the largest sensitivity coefficients for hsp and mfp. The coefficients with respect to ${k_5}^+$ and ${k_5}^-$ are the highest, suggesting that the reaction describing hsf sequestration (forward)/dissipation of $\mathsf{hsp\!:\!hsf}$ ($\mathsf{hsp}+\mathsf{hsf} \leftrightarrows \mathsf{hsp\!:\!hsf}$) is the main feedback loop. The forward direction, hsf sequestration, hereafter compels the ceasing of transcription, inducing an augmentation in the level of mfp and a decrease in that of hsp. The backward direction (dissipation of $\mathsf{hsp\!:\!hsf}$), however, actuates an increase in the level of hsp and hsf and a reduction of mfp. Considering the coefficients in Table 5 in descending order, the next set of coefficients to discuss consists of ${k_1}^+$, ${k_2}^+$ and k_4, corresponding to the forward directions of dimer(trimer) formation and DNA binding respectively, suggesting the augmentation of the transcription level and thereupon the level of hsp. On the contrary, the reactions describing the breaking of dimers, hsp degradation and protein misfolding, diminish the transcription level. The reactions influencing the level of mfp alone are the reactions corresponding to the sequestration of mfp's/dissipation of $\mathsf{hsp\!:\!mfp}$ (see coefficients corresponding to ${k_{11}}^+$ and ${k_{11}}^-$ in Table 5) and protein refolding (same for k_{12}).

Among the sensitivity coefficients of hsp and mfp with respect to the initial concentrations, the one dependent on the initial level of $\mathsf{hsp\!:\!hsf}$ ($\mathsf{hsp\!:\!hsf}(0)$) was the most relevant. On the other hand, the sensitivity coefficients of hsp and mfp with respect to the level of any of the hsf species (monomers, dimers or trimers) were insignificant. This is to be expected since initially the majority of hsf's is sequestrated by hsp's and the initial levels of dimers and trimers are reduced, which is consistent with [30]. Consequently, the sensitivity coefficient with respect to $\mathsf{hsp\!:\!hsf}(0)$ should be conceived as describing a dependency over the total initial amount of hsf.

The sensitivity coefficients with respect to the initial amount of hse were insignificant, which is justified by the consideration of the sensitivity coefficients around the steady state. For instance, for a lower initial amount of hse, the response reaches hereafter the same steady state. A higher level of $\mathsf{hsf}(0)$ brings no change in the evolution of the response. The sensitivity coefficients of hsp and mfp with respect to $\mathsf{hsp}(0)$ were also insignificant.

Model identifiability. Looking into the model identifiability problem, alternative good numerical fits were searched for, using the same fitting data as in the model fitting procedure described above. Several were found, but none of them passed the additional validation tests described in the previous section. Then the *Latin Hypercube Sampling* method was applied to sample the distribution of the fitting score function. The first step was to generate a sample of $N = 100000$ combinations of parameter values, as described in Section 4. For each of them, the initial values were chosen so that they are a steady state of the model at 37°C. Out of these, the analysis was continued only for those combinations that were "responsive", where a model was declared responsive if $\mathsf{hsf_3\!:\!hse}(900) \geq 20$ (note that the experimental data indicated that the peek of the

Table 5. The largest scaled steady state sensitivity coefficients of hsp and mfp. The coefficients are identical for both 37°C and 42°C [53]

Parameter description	p	$\frac{\partial ln(\mathsf{hsp})}{\partial ln(p)}\vert_{t\to\infty}$	$\frac{\partial ln(\mathsf{mfp})}{\partial ln(p)}\vert_{t\to\infty}$
Sequestration of hsf	$k_5{}^+$	−0.50	0.50
Dissipation of hsp: hsf	$k_5{}^-$	0.50	−0.50
Formation of dimers	$k_1{}^+$	0.17	−0.17
Formation of trimers	$k_2{}^+$	0.17	−0.17
Transcription, translation	k_4	0.17	−0.17
Affinity of hsp for hsf$_2$	k_6	−0.17	0.17
Affinity of hsp for hsf$_3$: hse	k_8	−0.17	0.17
Degradation of hsp	k_9	−0.17	0.17
Affinity of hsp for mfp	$k_{11}{}^+$	0.00	−1.00
Dissipation of hsp: mfp	$k_{11}{}^-$	0.00	0.24
Protein refolding	k_{12}	0.00	−0.24
Initial level of hsp: hsf	hsp: hsf(0)	0.50	−0.50

response is reached after 900 time units). The result was interesting: there were only 31506 models satisfying the constraint, already suggesting that finding suitable alternative model fits is a difficult problem. For each of these models we calculated the fit quality as discussed in Section 4; the result is plotted in Figure 6, showing clearly our best fit as an outlier in the fit quality distribution. More details on the identifiability of the heat shock response model can be found in [53]. This suggests that fitting the simple heat shock response model in Table 2 to the experimental data in [36] and to the steady-state condition for the initial values is indeed a difficult numerical problem.

7 Discussion

The focus of our chapter has been on the practical use of modeling with ordinary differential equations in biology. Our choice of topics to discuss has been driven by targeting primarily the computer science community and by the space limitations. This chapter should only be seen as a "teaser" for modeling with ODEs in biology; for a more comprehensive reading on this topic, many excellent textbooks exist, such as [11,32,47,48, 56,58]. We only considered in this chapter reaction-based models and started by discussing how to associate to them an ODE-based model; we presented briefly several laws for biochemical kinetics: mass-action, Michaelis-Menten, Goldbeter-Koshland, Hill, and inhibition. One should note that many other types of models exist, see, e.g., [9]. We then discussed the parameter estimation problem, including model identifiability, measures for fit quality, and fit-preserving model refinement. We then introduced several analysis methods for ODE-based models: steady state analysis, mass conservation, and sensitivity analysis. In addition to some smaller examples discussed throughout the chapter, we dedicated a separate section to a larger case-study on the eukaryotic heat shock response.

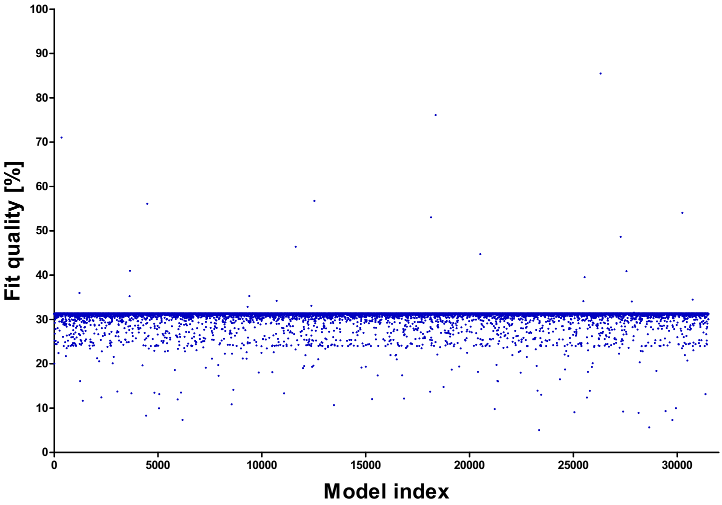


Fig. 6. The distribution of the model fit quality among 31506 model variants obtained through the Latin hypercube sampling method. Most models exhibit a constant level of hsf3:hse, very different from the dynamic behavior in Figure 5; these models yield a numerical value for the fit quality around 30%. The quality of our best fit is around 10^{-30}.

There are many computational benefits that modeling with ODEs brings, including fast numerical simulations, many methods for parameter estimation, several highly useful static and dynamic analysis techniques, such as mass conservation, steady state analysis, flux-balance analysis, metabolic control analysis, sensitivity analysis, etc. At the same time, the ODE-based approach also suffers from several difficulties. The one that is most discussed is the inability to account for stochastic noise in a system, which might be problematic especially in cases where there are relatively small species; a detailed discussion about the physical limitations of the ODE-based approach is in [14, 15]. Another difficulty is in the need for knowing a potentially large number of kinetic parameters; measuring them experimentally is sometimes impossible, while estimating them computationally suffers from model identifiability issues. A partial solution here is the approach based on quantitative model refinement, see [34]. Another partial solution is in terms of static, rather than dynamic analysis, often performed around the steady states; such an approach is modeling based on flux balance analysis, see [51].

The stochastic approach, either in terms of continuous time Markov chains (CTMC) and the chemical master equation, or in terms of higher-level formalisms (such as Petri nets or process algebra) based on a CTMC semantic, is often offered as a solution to the physical limitations of the ODE-based approach. It is important however to understand the limitations of both approaches so that we can take advantage of the benefits of either one, whenever they are applicable. In Table 6 we summarized several aspects about modeling with ODEs and with CTMCs, and placed them in mirror for an easy comparison. It is also important to point out that in the case of very large models, both approaches are insufficient, see Figure 7.

Table 6. Some of the differences between the deterministic and the stochastic modeling approaches

	Deterministic approach	*Stochastic approach*
Fundamental assumptions	the system is well-stirred and at thermodynamical equilibrium	the system is well-stirred and at thermodynamical equilibrium
Modeling goal	it models the average behavior of the system	it models individual runs of the system
Concept	based on the concept of diffusion-like reactions	based on the concept of reactive molecular collisions
Type of model	the time evolution of the model is a continuous process	the time evolution of the model is a random-walk process through the possible states
Math model	governed by a set of ODEs	governed by a single ODE: the chemical master equation
Analytic solution	the system of ODEs is often impossible to solve analytically	the chemical master equation is often impossible to solve
Small populations	conceptual difficulties when small populations are involved	no difficulties with small populations
Numerical simulations	fast	Gillespies algorithm is slow; many runs are needed

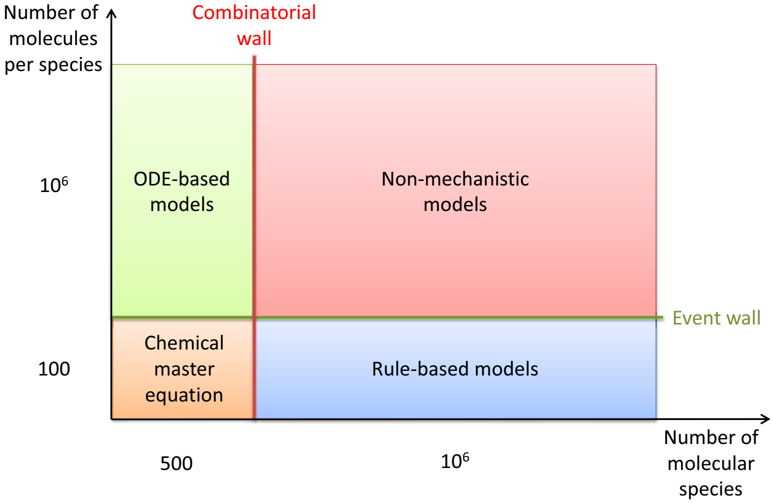


Fig. 7. Modeling limitations depending on the size of the model. Adapted from Walter Fontana `http://fontana.med.harvard.edu/`

The ODE-based approach to computational modeling is (still) arguably the standard choice for biomodelers, especially on the biological side of the community. There are many advantages that it brings, as there are clear limitations. Even in cases where another modeling approach is taken, the corresponding ODE-based model is often also built to serve as comparison to related (ODE-based) models and to make available tools such as parameter estimation or steady state analysis. Moreover, on top of the ODE-based semantic there are many other discrete techniques that can be added to give further insight into the model: Petri net tools, control analysis, network motif identification, etc. In the continuing debate of 'discrete vs. continuous biomodeling' we argue that it is good to retain the advantages of both worlds and use them to their full potential whenever applicable.

References

1. Abravaya, K., Phillips, B., Morimoto, R.I.: Attenuation of the heat shock response in hela cells is mediated by the release of bound heat shock transcription factor and is modulated by changes in growth and in heat shock temperatures. Genes & Development 5(11), 2117–2127 (1991)
2. Ali, M.M., Storey, C., Törn, A.: Application of stochastic global optimization algorithms to practical problems. Journal of Optimization Theory and Applications 95(3), 545–563 (1997)
3. Azimi, S., Gratie, D.-E., Iancu, B., Petre, I.: Three approaches to quantitative model refinement with applications to the heat shock response. Technical report, Turku Centre for Computer Science (2013)
4. Baker, S.M., Schallau, K., Junker, B.H.: Comparison of different algorithms for simultaneous estimation of multiple parameters in kinetic metabolic models. Journal of Integrative Bioinformatics 7(3), 1–9 (2010)
5. Ballew, R.M., Sabelko, J., Gruebele, M.: Direct observation of fast protein folding: the initial collapse of apomyoglobin. Proceedings of the National Academy of Sciences 93(12), 5759–5764 (1996)
6. Boender, G., Romeijn, E.: Stochastic methods. In: Handbook of Global Optimization: nonconvex optimization and its applications, pp. 829–869. Kluwer Academic Publishers (1995)
7. Briggs, G.E., Haldane, J.B.S.: A note on the kinetics of enzyme action. Biochemical Journal 19(2), 338–339 (1925)
8. Chen, W.W., Schoeberl, B., Jasper, P.J., Niepel, M., Nielsen, U.B., Lauffenburger, D.A., Sorger, P.K.: Input-output behavior of erbb signaling pathways as revealed by a mass action model trained against dynamic data. Molecular Systems Biology 5, 239 (2009)
9. Ciobanu, G., Rozenberg, G. (eds.): Modeling in Molecular Biology. Springer (2004)
10. Czeizler, E., Rogojin, V., Petre, I.: The phosphorylation of the heat shock factor as a modulator for the heat shock response. IEEE-ACM Trans. Comp. Biol. Bioinf. 9(5), 1326–1337 (2012)
11. Edelstein-Keshet, L.: Mathematical Models in Biology. McGraw-Hill, New York (1988)
12. Fisher, J., Henzinger, T.A.: Executable cell biology. Nature Biotechnology 25, 1239–1249 (2007)
13. Gessel, I.M., Stanley, R.P.: Algebraic enumeration. Handbook of Combinatorics 2, 1021–1061 (1995)
14. Gillespie, D.T.: A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. Journal of Computational Physics 22(4), 403–434 (1976)
15. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. The Journal of Physical Chemistry 81(25), 2340–2361 (1977)

16. Gillespie, D.T.: A rigorous derivation of the chemical master equation. Physica A: Statistical Mechanics and its Applications 188(1), 404–425 (1992)
17. Giordano, F., Weir, M., Fox, W.: A first course in mathematical modeling, 3rd edn. Thomson (2003)
18. Goldbeter, A., Koshland, D.E.: An amplified sensitivity arising from covalent modification in biological systems. Proceedings of the National Academy of Sciences 78(11), 6840–6844 (1981)
19. Grossmann, I.E.: Global optimization in engineering design. Kluwer Academic Publishers (1996)
20. Guldberg, C.M., Waage, P.: Studies concerning affinity. CM Forhandlinger: Videnskabs-Selskabet i Christiana 35, 92–111 (1864)
21. Guldberg, C.M., Waage, P.: Etudes sur les affinités chimiques. Brøgger & Christie (1867)
22. Haldane, J.B.S.: Enzymes (1930, 1965)
23. Harmer, R.: Rule-based modelling and tunable resolution. EPTCS 9, 65–72 (2009)
24. Heinrich, R., Schuster, S.: The regulation of cellular systems, vol. 416. Chapman & Hall, New York (1996)
25. Heinrich, R., Schuster, S.: The modelling of metabolic systems. structure, control and optimality. Biosystems 47(1), 61–77 (1998)
26. Helton, J.C., Davis, F.J.: Illustration of sampling-based methods for uncertainty and sensitivity analysis. Risk Analysis 22(3), 591–622 (2002)
27. Helton, J.C., Davis, F.J.: Latin hypercube sampling and the propagation of uncertainty in analyses of complex systems. Reliability Engineering and System Safety 81, 23–69 (2003)
28. Henri, V.: Lois générales de l'action des diastases. Librairie Scientifique A. Hermann (1903)
29. Hill, A.V.: A new mathematical treatment of changes of ionic concentration in muscle and nerve under the action of electric currents, with a theory as to their mode of excitation. The Journal of Physiology 40(3), 190–224 (1910)
30. Holmberg, C.I., Tran, S.E.F., Eriksson, J.E., Sistonen, L.: Multisite phosphorylation provides sophisticated regulation of transcription factors. Trends in Biochemical Sciences 27(12), 619–627 (2002)
31. Hoops, S., Sahle, S., Gauges, R., Lee, C., Pahle, J., Simus, N., Singhal, M., Xu, L., Mendes, P., Kummer, U.: Copasi-a complex pathway simulator. Bioinformatics 22(24), 3067–3074 (2006)
32. Hoppenstaedt, F.C., Peskin, C.S.: Modeling and Simulation in Medicine and the Life Sciences. Springer, New York (2002)
33. Horst, R., Tuy, H.: Global optimization: Deterministic approaches. Springer, Berlin (1990)
34. Iancu, B., Czeizler, E., Czeizler, E., Petre, I.: Quantitative refinement of reaction models. International Journal of Unconventional Computing (page to appear, 2013)
35. Jones, C.M., Henry, E.R., Hu, Y., Chan, C.-K., Luck, S.D., Bhuyan, A., Roder, H., Hofrichter, J., Eaton, W.A.: Fast events in protein folding initiated by nanosecond laser photolysis. Proceedings of the National Academy of Sciences 90(24), 11860–11864 (1993)
36. Kline, M.P., Morimoto, R.I.: Repression of the heat shock factor 1 transcriptional activation domain is modulated by constitutive phosphorylation. Molecular and Cellular Biology 17(4), 2107–2115 (1997)
37. Klipp, E., Herwig, R., Kowald, A., Wierling, C., Lehrach, H.: Systems biology in practice: concepts, implementation and application. Wiley-Vch (2005)
38. Kühnel, M., Mayorga, L.S., Dandekar, T., Thakar, J., Schwarz, R., Anes, E., Griffiths, G., Reich, J.: Modelling phagosomal lipid networks that regulate actin assembly. BMC Systems Biology 2, 107–121 (2008)
39. Lotka, A.J.: Elements of Physical Biology. Williams & Wilkins Company (1925)
40. Malthus, T.R.: An Essay on the Principle of Population. 1798

41. McKay, M.D., Beckman, R.J., Conover, W.J.: Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21(2), 239–245 (1979)
42. Mendes, P., Kell, D.: Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. Bioinformatics 14(10), 869–883 (1998)
43. Menten, L., Michaelis, M.: Die kinetik der invertinwirkung. Biochem. Z. 49, 333–369 (1913)
44. Miller, W.G., Alberty, R.A.: Kinetics of the reversible michaelis-menten mechanism and the applicability of the steady-state approximation. Journal of the American Chemical Society 80(19), 5146–5151 (1958)
45. Moles, C.G., Mendes, P., Banga, J.R.: Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome Research 13(11), 2467–2474 (2003)
46. Murphy, E., Danos, V., Féret, J., Krivine, J., Harmer, R.: Rule-based modeling and model refinement. Elements of Computational Systems Biology, 83–114 (2009)
47. Murray, J.D.: Mathematical Biology I: An Introduction. Springer, New York (2002)
48. Murray, J.D.: Mathematical Biology II: Spatial Models and Biomedical Applications. Springer, New York (2002)
49. Nelson, D.L., Cox, M.M.: Lehninger principles of biochemistry. Worth Publishers (2000)
50. Oberguggenberger, M., King, J., Schmelzer, B.: Classical and imprecise probability methods for sensitivity analysis in engineering: A case study. International Journal of Approximate Reasoning 50, 680–693 (2009)
51. Orth, J.D., Thiele, I., Palsson, B.: What is flux balance analysis? Nature Biotechnology 28, 245–248 (2010)
52. Peper, A., Grimbergen, C.A., Spaan, J.A.E., Souren, J.E.M., Van Wijk, R.: A mathematical model of the hsp70 regulation in the cell. International Journal of Hyperthermia 14(1), 97–124 (1998)
53. Petre, I., Mizera, A., Hyder, C.L., Meinander, A., Mikhailov, A., Morimoto, R.I., Sistonen, L., Eriksson, J.E., Back, R.-J.: A simple mass-action model for the eukaryotic heat shock response and its mathematical validation. Natural Computing 10(1), 595–612 (2011)
54. Rabitz, H., Kramer, M., Dacol, D.: Sensitivity analysis in chemical kinetics. Annual Review of Physical Chemistry 34(1), 419–461 (1983)
55. Rieger, T.R., Morimoto, R.I., Hatzimanikatis, V.: Mathematical modeling of the eukaryotic heat-shock response: Dynamics of the hsp70 promoter. Biophysical Journal 88(3), 1646 (2005)
56. Rubinow, S.I.: Introduction to Mathematical Biology. John Wiley, New York (1975)
57. Stewart, W.J.: Probability, Markov chains, queues, and simulation. The mathematical basis of performance modeling. Princeton University Press, Princeton (2009)
58. Taubes, C.: Modeilng Differential Equations in Biology. Prentice Hall, Upper Saddle River (2001)
59. Turányi, T.: Sensitivity analysis of complex kinetic systems. tools and applications. Journal of Mathematical Chemistry 5(3), 203–248 (1990)
60. Volterra, V.: Animal ecology. In: Chapman, R.N. (ed.), pp. 409–448. McGraw-Hill, New York (1926)
61. Wilkinson, D.J.: Stochastic modelling for systems biology. Chapman & Hall/CRC Mathematical Biology and Medicine Series (2006)
62. Zi, Z.: Sensitivity analysis approaches applied to systems biology models. Systems Biology, IET 5(6), 336–346 (2011)

Model Checking of Biological Systems*

Luboš Brim, Milan Češka, and David Šafránek

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

Abstract. Model checking together with other formal methods and techniques is being adapted for applications to biological systems. We present a selection of approaches used for modeling biological systems and formalizing their interesting properties in temporal logics. We also give a brief account of high performance model checking techniques and add a few case studies that demonstrate the use of model checking in computational systems biology. The primary aim is to give a reference for further reading.

1 Introduction

All biological systems, from single pathways to multicellular organisms, can be seen as complex systems of interacting components. Biological systems can also be seen as reactive systems, as they continuously interact with their environment. Systems biology thus studies complex *interactions* in biological systems, with the aim to understand better the processes that happen in such a system, as well as to grasp the emergent properties of such a system as a whole.

Computational systems biology can, by drawing upon mathematical approaches developed in the context of computer science and engineering [87,144], contribute to the creation of powerful simulation, analysis and reasoning tools for working biologists. These tools can be used in devising new experiments and ultimately, for understanding functional properties of genome, proteome, cells, and organisms.

We are experiencing growing collaboration between biologists and computer scientists in the area of systems biology in recent years. This is because it has turned out that formal mathematical approaches to modeling and analysis, that have been developed for distributed computer systems and are referred to as formal methods, are applicable to biological systems as well as both kinds of systems have a lot in common.

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

M. Bernardo et al. (Eds.): SFM 2013, LNCS 7938, pp. 63–112, 2013.

In particular, automated formal verification (model checking) is one of the most promising formal methods that have the potential to be exploited in computational systems biology, because model checking is in principle an excellent methodology to verify/refute interesting biological hypotheses.

In this tutorial review, we would like to briefly describe some of the issues related to the application of model checking to the analysis of biological systems.

2 Setting the Context

2.1 Model Checking of Computer Systems

Model checking is a computer science and engineering technique that grew up from a purely academic research technique to a well-accepted industrial verification method. Nowadays, model checking is widely considered as an enhancement and complement to existing validation and verification techniques such as simulation and testing.

The roots of model checking lay in our never-ending quest to build computer systems that would be bug-free and correct. Our dependency on computer-based applications (both hardware and software) have motivated researchers to develop new techniques to increase our confidence in correctness of developed systems.

Testing is the basic verification technique that is widely used and extremely useful in practise. Another solution is to simulate the behavior of the system on a computer. Simulation does not work directly on the real system, but on a model. A model is an abstract representation of the real system. An advantage of simulation is that one does not need to build the real system and thus it is usually much cheaper than testing.

Both testing and simulation are widespread in industrial applications and their utilization has been shown to be very useful. One drawback, however, is that it is not possible, in general, to simulate or test all the possible scenarios or behaviors of a given system. That is, these techniques are in general not exhaustive and the failure cases may appear among those not tested or simulated.

Formal verification is a technique that complements testing and simulation. Even though the introduction of formal verification is rather costly, it pays off after all as it results in significant reduction in verification time as well as development costs and time-to-market. Attempts are being made to integrate formal verification techniques and tools with other design approaches to support engineering of complex industrial systems.

Model checking is a distinguished technique of formal verification of complex hardware and software designs. Founders of the technique, Edmund M. Clarke jr. (CMU, USA), Allen E. Emerson (Texas at Austin, USA), and Joseph Sifakis (IMAG Grenoble, France), were awarded ACM Turing Award in 2007 *for their roles in developing model checking into a highly effective verification technology, widely adopted in the hardware and software industries.* Unfortunately, the model checking procedure is computationally demanding and memory-intensive in general, hence, its applicability to large and complex systems routinely seen in practise these days is still limited. The major hampering factor is the *state space explosion problem* [61] due to which large industrial models cannot be efficiently handled, unless more sophisticated and scalable methods are used.

A lot of attention has been paid to the development of approaches to fight the state space explosion problem in the field of automated formal verification [139]. Many techniques, such as state compaction [93], compression [107], state space reduction [140,58,81], symbolic state space representation [41], etc., were introduced to reduce the memory requirements needed to handle the verification problem with standard sequential algorithms. These techniques allowed to verify larger systems without the need of increased computing power.

However, for large industrial models, the state space, even if significantly reduced using the above mentioned techniques, does not completely fit into the main memory of a computer and hence the model-checking algorithm becomes very slow as soon as the memory is exhausted and the system starts swapping. A typical approach to dealing with these practical limitations is to increase the computational power (especially the amount of random-access memory) by building a powerful parallel computer as a network (cluster) of workstations. Individual workstations communicate through a message-passing interface such as MPI. Observed from outside, a cluster appears as a single parallel computer with high computing power and a large amount of memory. In recent years, a lot of effort has been invested into using parallel and distributed environments in order to solve the computational and space complexity bottlenecks in model checking and therefore we devote a special section to review some parallel and distributed approaches (Section 3.4).

2.2 On the Role of Model Checking in Systems Biology

There are many ways how we can improve correctness of computer systems. The used methods and techniques are generally classified as *verification* and/or *falsification* approaches. The role of verification techniques, typically theorem proving, is to guarantee there is no bug in the system while the role of falsification techniques, typically testing, is to demonstrate the presence of errors. Model checking is primarily a verification technique which is, however, often used for falsification (as a bug hunting method).

We might tend to a similar position of model checking when applied to biological systems. The situation is, however, different for many reasons (see Fig. 1). The most important difference is that in biology, the system under investigation already exists. It is not the primary role of biology to create life (at least to some extent). On the other hand, computer systems are man-made. In computer engineering the models are used as abstractions that are step-wise transformed into the final system. This contrasts to experimental sciences where models serve as hypotheses. The role of verification in computer engineering is to ensure the system that was constructed from the model has the same behavior as prescribed by the model. If the verification fails, the system has to be corrected. In the case of successful verification, we are done – the engineer has completed his task. On the other hand, in experimental sciences the goal is to show that the hypothesis correctly captures some aspects of the real system. Scientific ideas are tested by generating multiple possible hypotheses, coming up with predictions for each of them, and then designing tests (experiments) by which we can falsify the hypotheses. Typically, we test hypotheses in order to refute them, not to try to support them. In computer engineering the model is thus always correct, while in science the system is.

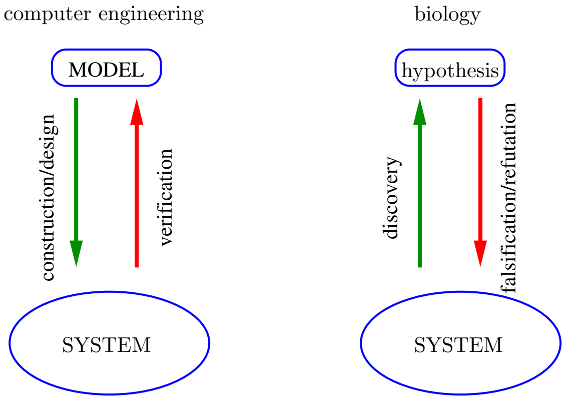


Fig. 1. Knowledge discovery in biology and computer engineering

Now let us have a closer look at the possible position of model checking in computational systems biology. Systems biology can be characterized as an approach to the understanding of life through the study of how the properties of biological systems arise through interactions between the system components [137]. From this point of view, biological systems are similar to complex computer systems. Namely, in both kinds of systems the interaction of components is a source of various emergent system properties that are not explicitly encoded in individual system parts. The common problem related to the analysis of such systems is that the emergent properties are difficult to identify and quite often hard to understand because the causes and effects are not obviously related.

For complex parallel and distributed software and hardware systems the process of detection and analysis of emergent properties relates closely to the process of formal verification. It is often the case that the emergent properties of distributed systems, such as deadlocks or non-progressive cycles, are properties that the designers of the system have not the intention to introduce. Methods of automated formal verification, model checking in particular, can be thus used to detect such properties and to prove their absence.

Going beyond verification and/or falsification of properties of biological systems, there are many other interesting questions having sometimes no direct counter-part in computer systems, that can be solved by application of model checking techniques. An example is the problem of *parameter identification* (also called parameter estimation or model calibration). Parameter identification is a key issue in systems biology, as it represents the crucial step to obtaining better models of biological systems that give more precise predictions. This issue is usually addressed by fitting the model simulations to the observed experimental data. In biological models, the control parameters used to define the behavior of models are kinetic or rate parameters. Some of these parameters usually cannot be experimentally determined which leads to the need to estimate these parameters by computational methods. To this end, model checking provides a promising alternative to fitting – parameters can be identified or fine tuned to satisfy given set of properties. Parameter identification by model checking has been referred to as parameter synthesis [76,13,25].

In [40,147,76], comprehensive parameter exploration techniques are introduced. They are based on the construction (usually approximation) of a landscape function that maps every model parametrization to a value quantitatively characterizing validity of the properties. Landscape function has direct application in robustness analysis. Robustness can be understood as a feature of a system to maintain a property in the face of parameter perturbation.

3 Description of Technique and Tools

In this section we give more technical details about models used in systems biology and their biological properties. We also introduce some parallel and distributed approaches to model checking as high performance techniques to support analysis of complex biological systems.

3.1 Models of Biological Systems

Most of the models currently developed in systems biology focus on complex interactions among system components. State-of-the-art biological knowledge is being reconstructed and organized in the form of *biological networks*. Biological networks are built from biological knowledge databases, experimental data and generally understood principles based on many simplifying assumptions. There are two fundamental types of biological networks – *reaction networks* and *regulatory networks*. Recent network reconstructions typically mix the two. Reaction networks provide a detailed view of underlying biochemical interactions – nodes are chemical species and stoichiometry-labeled (multi-)edges represent elementary chemical reactions. Regulatory networks are higher level and focus on feedbacks among individual system components – nodes are species or abstract biological objects and edges represent positive or negative influence. Gene regulatory networks make a typical example [115].

Computational systems biology studies the dynamics of biological networks, in particular, how a population of components affected by network interactions evolves in time. To this end, *biological model* is defined as a biological network associated with a suitable semantics reflecting the system dynamics at a particular level of abstraction. The semantics fulfils the following tasks:

- Network components are given a mathematical interpretation as *variables* (numbers of molecules or molar concentrations),
- Network interactions are given a mathematical interpretation as *rules* specifying dynamical changes in variables.

Both variables and rules can be understood as model quantities modeled at different levels of abstraction. Variables can be treated as either discrete or continuous. *Discrete-value* semantics can capture either a *microscopic or mesoscopic view* of biological particles (e.g., number of molecules) or abstract qualitative interpretations of selected qualities of modeled components (e.g., absence/presence of a species). *Continuous-value* semantics represents a so-called *macroscopic view* where the modeled objects are expected to appear in large quantities provided that it is inconvenient to distinguish small differences (e.g., molar concentration of a species in a cell).

Quantities that can be associated with rules are time and probability. Since each interaction occurs in time with a specific *rate*, the respective rule is executed with this rate implying the inherent time aspect of the system dynamics. Naturally, time is considered as continuous, dense quantity. When the information on rate is unknown or abstracted out due to simplifications, *discrete-time* semantics is employed. It deals with the shortest (discrete) time step which can represent an arbitrary finite time horizon. Discrete-time abstraction allows to treat qualitative models as *untimed*, i.e., the exact duration of a single time step is left unspecified. It is worth noting that in the most of cases the occurrence of any rule is modeled as instantaneous and it occurs immediately after the conditions for occurrence are satisfied. There exist models that refine these aspects of semantics (e.g., delayed interactions [45] or non-instantaneous interactions [11]).

With respect to execution of interactions, rules can be either *deterministic* or *stochastic*. Deterministic rules represent interactions that occur each time all preconditions are satisfied (e.g., if there is a non-zero amount of all reactants, the reaction occurs). There is no noise affecting the interaction. Stochastic interactions reflect noisy environment by assuming a certain probability with which they occur.

Finally, there is yet another notion of quantity that can enhance the model semantics. In particular, interactions and even variable values can be assigned quantitative costs and *rewards*, e.g., time spent in particular concentration levels, energy consumed by particular reactions, etc. By adding this kind of information (if available), models can be adjusted to provide interesting and detailed quantitative predictions resulting from complex dynamics.

Types of semantics mentioned above can be suitably combined resulting in several classes of models varying in the level of abstraction employed, as is overviewed in Fig. 2. On the right side of the scheme, there are models considering continuous component quantities and deterministic interactions. These inherently quantitative models are currently the most widely used in computational systems biology since they have deep roots in mathematical biology. In fact, from the semantics point of view they are purely denotational [87] and thus we call them *mathematical*. On the left side of the scheme, there are discrete-value models which can be either quantitative (incorporating real-time and/or stochasticity of interactions) or qualitative (abstracting from the timed nature and stochasticity of interactions). These models are closer to computer science or they directly originate from computer science. Fisher and Henzinger [87] classify these models as *executable*, since for any of them the semantics can be considered either denotational or operational. The operational view allows to understand biological systems in the similar way as programs or any formal models in computer engineering. That way these models naturally bring the model checking technique to biology.

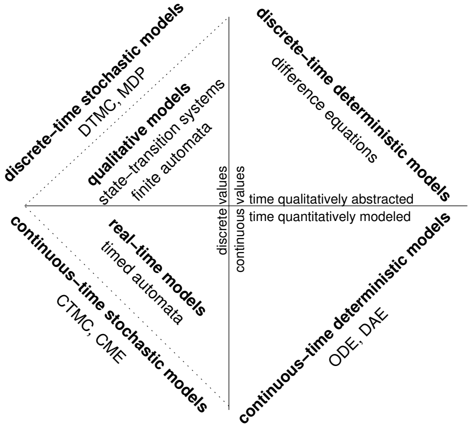

discrete–time stochastic models
DTMC, MDP
qualitative models
state–transition systems
finite automata
discrete–time deterministic models
difference equations
discrete values
continuous values
time qualitatively abstracted
time quantitatively modeled
continuous–time stochastic models
CTMC, CME
real–time models
timed automata
continuous–time deterministic models
ODE, DAE

Fig. 2. Model types sorted according to different level of details captured in their semantics

Mathematical models are used to represent actual quantitative relations between components in the system. Generally, a system of ordinary differential equations (ODEs) [118,110] and/or differential algebraic equations (DAEs) [37] is used to represent the interaction and processes among the various components. Determinism and continuity reflect the modeled phenomena in high chemical species concentrations or large cell populations (the macroscopic level) while completely neglecting the noise and differences in individual components and interactions. These models can be simulated, analyzed, and possibly solved, but require detailed knowledge of the biological system, i.e., *quantitative parameters* identifying the physical aspects of system interactions (e.g., kinetic coefficients of chemical reactions).

On the other hand, executable models employ abstract representations to explain biological phenomena. Examples of widely used formalisms are Boolean networks [159,50,121], Petri Nets [103,32,49,143], timed automata [27,153,97], compact process algebraic representations such as BioPEPA [56], Kappa [68] or suitable adaptations of π-calculus [141,145]. These formalisms have an inherent execution scheme attached to the models, and relate different qualitative configurations (*states*) of model components to each other. The relation among states can be either qualitative or quantitative (with real-time bounded or even stochastic rules). The advantage is the capability to effectively represent the logic behind biological systems dynamics without precise quantitative knowledge about the component interactions. Executable models are inherently discrete provided that the dynamics (execution) occurs in terms of a series of discrete events. In the untimed setting, nondeterminism allows to capture all possible "timings" (orderings) of concurrent events. To quantitatively differentiate among all possible executions in a particular state, rules can be assigned probabilities, resulting in discrete stochastic models [36,163] most typically represented by discrete-time Markov chains (DTMC). When set appropriately, executable models can be used at any level of view of biological systems dynamics.

Stochastic models allow to incorporate noise which causes fluctuations in component quantities and that way affects the biological system dynamics [80,120]. In physics, chemistry and related fields, the probabilistic time-evolution of a system with discrete component quantities is described by so-called master equations. In the case of biological phenomena, the chemical master equation (CME) provides an exact mathematical model for stochastic dynamics [95]. It is formalized as a set of differential equations, providing a denotational representation of component quantities distribution in continuous-time. Gillespie [94,96] has made an important breakthrough in stochastic modeling by introducing techniques for exact simulation of CME. From the computer scientific viewpoint, the CME can be equivalently represented by continuous-time Markov chains (CTMC) which provide operational semantics and allow us to consider continuous-time stochastic models as executable [72].

Although outside the scope of this paper, it is worth mentioning that a significant and general class of models is that of *hybrid models* most typically represented by means of hybrid automata [105] or process algebraic techniques [90,34]. Hybrid models allow to mix discrete-value components with continuous-value components and discrete-time dynamics with continuous-time dynamics. Such a complicated semantics limits the model analysis [106,44]. Hybrid models can be satisfactorily used for modeling and simulation [75] and, when simplifying assumptions are employed (e.g., considering linear dynamics of continuous components), also for a more advanced analysis of biological processes [67,89]. To incorporate noise, stochastic hybrid models [154] (i.e., stochastic hybrid automata) allow both discrete-time and continuous-time dynamics to evolve randomly. Coupling of both kinds of dynamics while keeping their stochasticity complicates analysis even more. To this end, simulation-based (statistical) [70] or fluid-flow approximation techniques [34,65] are typically employed.

Model Simplification. It is important to note that component quantities in biological models most typically do not evolve unlimitedly. In particular, concentration (or number of molecules) is always limited by degradation processes. However, it might not be easy to identify the bounds without a deeper analysis of the model. From the context of an observed phenomenon and the time-scale of relevant model behavior, it can be possible to identify the time-horizon (or the number of steps in the untimed case) for which it is guaranteed that the phenomenon occurs. Even periodically repeating phenomena, e.g., circadian clock, can be approximately detected and analyzed in finite time in the order of an appropriately selected time-scale. Again, a non-trivial analysis has to be performed in some cases to estimate or overapproximate correctly the time horizon. In any case, some hypotheses on maximal (extreme) bounds on component quantities or time can be always considered.

From the computational point of view, there exist many well developed and efficient techniques for exhaustive analysis of models appearing in the top left quadrant of the scheme in Fig. 2. The assumptions stated above imply finite number of model states. However, models in other quadrants incorporate continuous or dense quantities which significantly complicate or even disallow the direct exhaustive analysis. We focus on continuous-time discrete-value models first. Reduction of timed automata into untimed finite automata [2] is the crucial procedure enabling exhaustive analysis for continuous-time discrete-value models. A continuous-time stochastic model represented by a continuous-time Markov chain is reduced to a discrete-time Markov chain and a Poisson process (or a birth process) by uniformization techniques [161,72]. All reductions at this level are exact, provided that no information is lost.

There are techniques to abstract (or approximate) continuous-value models by discrete-value models. Formally defined abstractions allow specific properties to be preserved by means of over-approximation (resp. under-approximation) of model behavior. However, the effect of behavior spuriously added (resp. lost) by the abstraction is usually large. Over-approximative abstractions are conservative in the sense that each execution of the original model is also present in the abstract model but there can appear a new behavior, not present in the original model. Underapproximative abstractions ensure that no execution is added to the abstract model but some can be ignored.

Besides formal abstraction, there are approximations that distort the original behavior rather than adding or removing some. Such approximations do not guarantee preservation of dynamics properties but the deviation of behavior is ensured not to exceed a certain (specified) approximation error.

In the case of continuous-time deterministic models, typically non-linear, the most widely used approximation is provided by numerical simulation (integration) methods. For certain classes of ordinary differential equations, there are also formal abstraction techniques providing a discrete-time discrete-value over-approximation in terms of non-deterministic finite automata [116,30,99,62] or a continuous-time discrete-value over-approximation in terms of timed automata [133]. In the former case, the extent of falsely added executions is usually large whereas the latter case prevents addition of any executions with non-realistic timing and therefore the number of false executions can be reduced.

Some classes of continuous-time deterministic models can be also approximated by continuous-time stochastic models provided that continuous-value variables are approximated by a suitable number of uniformly distributed discrete levels. If calibrated properly, averaged stochastic executions converge to the deterministic solution [42] (see [103] for tutorial).

In reverse direction, some classes of continuous-time stochastic models can be approximated by deterministic continuous-time models (ODEs) by means of fluid-approximation techniques [69]. Based on these techniques, more sophisticated analysis methods for stochastic models, combining fluid-approximation with CTMC analysis have recently been proposed [33]. The advantage of these techniques is that they avoid the state-explosion problem.

In this text we focus on well-established methods developed for discrete- and continuous-time discrete-value models, in particular, we consider techniques targeting the exhaustive analysis of temporal properties of systems dynamics based on qualitative and quantitative model checking. Brief description of concrete techniques is presented in Section 3.3. Properties of biological interest are described in Section 3.2. Examples of models and the application of model checking techniques is presented on several case studies in Section 4.

Model Parameters. Biological model of any type is determined by a fixed topology (biological network) where the interactions (rules) are parametrized. Parameters provide degrees of freedom in which the model dynamics can be adjusted. In contrast to the network topology which stands on common principles, finding a correct model parametrization is a non-trivial task which makes a critical part of the so-called *inverse problem* [82].

Parameters appear in all kinds of models. In the case of qualitative models, a parameter most typically affects the logic behind a rule, i.e., adjusting the effect of the respective interaction on model components. Sets of possible parmeter values (*parametrizations*) for qualitative models are finite and discrete, but can be very large (e.g., the number of parametrizations for setting the dynamics of a gene A in a Boolean model of a gene regulatory network is exponential wrt the number of genes affecting the expression of A). In quantitative models, parameters represent the quantitative aspects associated with the semantics of rules, i.e., how the respective interaction evolves in time. In continuous-time stochastic (resp. deterministic) models the parameters describe the rate (resp. velocity) of respective interactions. In real-time models, the parameters describe time delays (minimal or maximal) between particular interactions. In all these cases, parametrization sets are are uncountable and bounded by laws of physics.

3.2 Biologically Relevant Properties

With respect to the nature of phenomena generated by dynamics of biological processes, typical properties studied on biological models can be organized into six elemental categories: reachability properties, temporal ordering of events, variable correlations, (multi)stability properties, monotonic trends, and oscillation properties. To reason about the model types presented in the previous subsection, properties have to be expressed with different levels of detail.

Qualitative Properties. *Qualitative properties* abstract away from any quantitative information like time aspects or energy costs of targeted biological phenomena. Qualitative properties are in general interpretable on all types of models, especially on untimed discrete-value models. Computer science offers two basic logical formalisms allowing to express qualitative properties of systems dynamics: linear-time temporal logics, interpreted on individual model executions (paths), and branching-time temporal logics, interpreted on trees of (non-deterministically) branching model executions.

The basic linear-time temporal logic is Linear Temporal Logic (LTL) [142]. LTL has been proved to be the basic formal language that is most suitable for qualitatively expressing properties of the six elemental categories. LTL can be interpreted on all kinds of models.

The most basic branching-time logic is Computational Tree Logic (CTL) [57]. In contrast to LTL, CTL allows to reason about non-determinism and, therefore, is used for properties dealing with non-determinism. CTL is also interpretable on all kinds of models, yet, the expressiveness of CTL is limited in the case of deterministic models. Below we give several examples of biologically-relevant properties. Unless otherwise mentioned, these properties will be expressed in LTL.

Qualitative *reachability properties* express reachability of specified concentration levels in given model variables. For example, the formula $\mathbf{F}(2 \leq B \leq 3)$ expresses the property that B reaches the concentration level between 2 and 3 at some point during the progress of the model dynamics. The linear-time formula $\mathbf{F}\varphi$ containing the operator $\mathbf{F}$ (Future) has the intuitive meaning that, on a given path, there must eventually exist a state where φ is satisfied. Note that the property tells nothing regarding the moment at which the event occurs. Reachability properties are useful especially for expressing global bounds of reachable concentration values.

To capture the qualitative temporal patterns in the dynamics of inspected variables, the properties expressing *temporal ordering of events* are used. These properties are based on linear-time operator $\mathbf{U}$ (Until), i.e., the formul $\varphi_1 \mathbf{U} \varphi_2$, with an intuitive meaning that, on a given path, φ_2 must eventually hold in some ith state on the path and for all states from the beginning of the path until the ith state, φ_1 must hold. An example of such property is the formula $(A \leq 2)\ \mathbf{U}\ [(2 < A \leq 5)\ \mathbf{U}\ (A > 5)]$ representing the following temporal pattern: species A is initially kept below 2 until it reaches 5 and finally exceeds 5.

Variable correlations make important observations revealing cooperations and dependencies in biological processes, e.g., co-expression of certain genes. These properties can be expressed by combining several temporal ordering formulae into a single formula using traditional logical operators. Following this approach, mutual dependencies in the dynamics of inspected variables can be captured. For example, the formula $[(A \leq 2)\ \mathbf{U}\ ((2 < A \leq 5)\ \mathbf{U}\ (A > 5))] \Rightarrow [(B \geq 10)\ \mathbf{U}\ ((5 \leq B < 10)\ \mathbf{U}\ (B < 5))]$ expresses the following correlation in concentration of species A and B: if A increases according to the temporal pattern from the previous paragraph then B decreases from a level above 10 to a level below 5.

A specific kind of temporal properties deals with the analysis of presence of stable concentration levels. An example of an elementary *stability property* is the formula $\mathbf{G}(A \leq 2)$ stating that concentration below 2 is stable (attractor) for species A. The formula $\mathbf{G}\varphi$, with the operator $\mathbf{G}$ (Globally), expresses the requirement that φ must hold in each state of a given path, the intuitive meaning is "forever". Stability properties can be effectively combined with reachability properties and relativized with respect to a specific initial condition. For example, the formula $(A \geq 0) \Rightarrow \mathbf{FG}(A \leq 2)$ states that the stable concentration below 2 is reached from any non-negative initial concentration of A. To query for existence of several different stable states (multi-stability), the LTL formula $[(A \leq 5) \Rightarrow \mathbf{G}(A \leq 5)] \wedge [(A > 5) \Rightarrow \mathbf{G}(A > 5)]$ can be employed. It expresses the fact that there are two different stable concentration levels in the dynamics of A: the first is below the level 5 and the second is above 5. Note that this formula expresses only the existence of the two stable attractors, there is nothing specified with respect to reachability of both stable attractors from a particular part of the state space (the so-called basin of attraction). To this end, CTL has to be employed: $\mathbf{EFAG}(A \leq 5) \wedge \mathbf{EFAG}(A \geq 5)$. The branching-time operator $\mathbf{EF}\varphi$ requires the existence of a branch where φ is eventually satisfied, whereas $\mathbf{AG}\varphi$ requires φ to hold in only those states. Therefore, the bistability formula is satisfied in every state from which the execution can eventually branch into both attractors.

Important observations of biological dynamics are *monotonous trends* in system variables [9]. Monotonicity is an indicator of robust increasing or decreasing phases observed in individual species dynamics. In the qualitative setting interpreted on discrete-value models, the non-strict monotonicity can be expressed as a special case of temporal ordering property, e.g., $(A = 1)\ \mathbf{U}\ [(A = 2)\ \mathbf{U}\ (A = 3)]$.

Finally, an interesting dynamics phenomenon appearing in biology is *oscillation*, e.g., circadian rhythms. A simple example of an oscillation property is expressed by the formula $(\mathbf{G}[(A \leq 3) \Rightarrow \mathbf{F}(A > 3)]) \wedge (\mathbf{G}[(A > 3) \Rightarrow \mathbf{F}(A \leq 3)])$ representing a permanent oscillation of A around the concentration level 3. Oscillation properties require linear-time operators, they cannot be expressed in CTL. Finer specification of oscillations can be realized by extending the formula with additional constraints identifying the qualitative aspects of the oscillation, e.g., the maximal and minimal amplitude levels.

Quantitative Properties. Quantitative properties including time aspects, energy consumptions and the stochasticity of a system are essential in the analyses of the dynamics of biological systems. Hence, a wide variety of logical formalisms allowing to reason about quantitative system aspects have been used to study biological systems. These formalisms usually extend the aforementioned logics and can be roughly divided into deterministic and stochastic logics. Deterministic logics are mostly focusing on a quantitative notion of time. The time extension of CTL called Timed Computational Tree Logic (TCTL) has been introduced in [2] and its simplified version is used as a specification language in the tool UPPAAL [29]. The extension allows to specify additional clock constrains, e.g, the TCTL formula $\mathbf{EF}(\varphi \wedge t \leq 3)$, where t is a clock, requires the existence of an execution branch where φ is satisfied within 3 time units. A popular dense time extension of LTL is Metric Interval Logic (MITL) introduced in [3] as a restriction of Metric Temporal Logic (MTL) [123]. It is based on the timed until modality $\mathbf{U}^I$ where the interval I is a nonempty convex subset of $\mathbb{R}_{\geq 0}$. The formula $(A = 1)\ \mathbf{U}^{[a,b]}\ (A = 2)$ is satisfied at any time instant t such that $A = 2$ at some $t' \in [t+a, t+b]$, and $A = 1$ continuously from t to t'. Another time extension of LTL called Timed Propositional Temporal Logic (TPTL) [4] is based on freeze-quantification where extra clocks are used to specify temporal constraints. These clocks can be reset at some point and later we can compare their values to some integers. The TPTL formula $\mathbf{G}[(A = 1) \Rightarrow x.\mathbf{F}(B = 3 \wedge x \leq 5)]$ expresses that whenever the population of species A reaches 1, the population of species B will reach 3 in 5 time units.

Motivated by the application of verification and monitoring techniques to continuous-value and hybrid systems, Signal Temporal Logic (STL) has been introduced [134]. It combines the dense time modalities of MITL with the numerical predicates over real numbers. The predicates are given as a real-value signal describing the evolution of the system, e.g, a function from time to a Cartesian product over reals. The formula $\mathbf{G}^{[0,300]}[(x_1 > 0.7) \Rightarrow F^{[3,5]}(x_2 > 0.7)]$, where x_1, x_2 are some signals, expresses that for each time point $t \in [0.300]$ it holds that if the value of the signal x_1 in t is greater than 0.7 then there exists time $t' \in [t+3, t+5]$ such that the value of the signal x_2 in t' is also greater than 0.7. For example, the tool Breach [76] employs STL to define temporal logic formulae and check whether they are satisfied on simulated trajectories. A version of LTL with constraints over the reals, named LTL($\mathbb{R}$), has been proposed in [5] to express the temporal properties of molecular concentrations and their derivatives. The quantifier free fragment of the first-order extension of LTL($\mathbb{R}$), named QFLTL($\mathbb{R}$) has been considered in [86]. It allows to use free variables in the atomic propositions and, thus, it enables to analyze numerical data time series in temporal logic and to automatically compute LTL($\mathbb{R}$) specifications from experimental traces. The formula $\mathbf{F}(A \geq p)$ expresses the question what threshold p species A attain in the trace. These two extensions of LTL are used in the tool BIOCHAM [84] to formalize numerical temporal properties.

Stochastic logics provides means to specify the probability and performance measures on Markov chains. In the case of DTMCs, a probabilistic extension of CTL, named PCTL, can be employed [100]. The logic is based on the probabilistic operator $\mathbf{P}_{\sim p}[\phi]$ expressing that the probability of the path formula ϕ being satisfied from a given state meets the bound $\sim p$. As a path formula it allows standard bounded and unbounded temporal operators. Note that, PCTL is a discrete-time logic and thus the path formulae are interpreted over discrete time steps. The PCTL formula $\mathbf{P}_{\geq 0.9}[\mathbf{F}^{\leq 5}(A = 3)]$ expresses that the probability that the population of species A will be equal to 3 within 5 time steps is at least 0.9.

To formalize properties of CTMC, Continuous Stochastic Logic (CSL) [6] has been introduced. It is a probabilistic extension of CTL with continuous-time semantics. In contrast to PCTL, the path formulae in CSL uses an interval of non-negative reals, rather than simply an integer upper bound. The CSL formula $\mathbf{P}_{\geq 0.9}[\mathbf{F}^{[1,2]}(A = 3)]$ expresses that the probability that the population of species A will be equal to 3 between 1 and 2 time units is at least 0.9. The logic also includes the steady-state operator **S** describing the steady-state behavior of a CTMC. The CSL formula $\mathbf{S}_{\leq 0.05}[A > 10]$ expresses that the long run probability that the population of species A will be higher than 10 is at most 0.05.

To further broaden the scope of possibly expressible behavior, PCTL and CSL have been extended to allow the specification over reward-based stochastic models, i.e., Markov chains with real-valued rewards/costs attached to states and transitions [126]. The extension enables to express properties such as the expected time a system spends in a specified set of states over a time interval or the expected number of times that a particular reaction occurrs.

The only way to combine temporal operators in PCTL and CSL is to use a nested formula whose meaning can be too subtle. Therefore, a probabilistic extension of LTL has been introduced in [64] allowing to express the probability of more complex events. The semantics of the logic is defined over Markov Decision Processes (MDPs) [71] which are a widely used formalism for modeling systems that exhibit both probabilistic and nondeterministic behavior, see e.g., [88] for more details.

Expressing biological phenomena can require extensions of existing logics. Biologically relevant temporal logic extensions target precise quantitative description of oscillations [74,24] or qualitative properties combining linear-time properties with branching-time [136].

3.3 Model Checking Techniques for Analysis of Biological Systems

Model checking techniques for the analysis of biological systems can be roughly divided into *exhaustive techniques* and *monitoring techniques*. The exhaustive techniques consist of checking whether all executions – state-event sequences – generated by a given system $\mathcal{S}$, satisfy the inspected property described as the formula φ, i.e., they effectively decide the language inclusion $\|\mathcal{S}\| \subseteq \|\varphi\|$ ($\|\varphi\|$ is the set of all executions that satisfy φ). In order to generate all executions, the whole state-space has to be stored and evaluated. This is why the exhaustive techniques generally suffer from the state-space explosion problem. There exist several techniques allowing to reduce this problem, e.g., efficient symbolic representation, state-space reductions or iterative abstraction refinement. For systems which are outside the scope of exhaustive techniques, either due to the incorporation of continuous and/or unbounded values or simply due to the state-space explosion problem, the monitoring techniques are the only feasible validation method. Unlike the inclusion, test the monitoring techniques are based on the membership test $\omega \in \|\varphi\|$ of an individual simulation trace $\omega \in \|\mathcal{S}\|$, where the responsibility for exhaustive coverage is delegated to the procedure that generates the traces. The key observation behind their efficiency is that for large and complex systems, the simulation is generally easier and faster than building a concise representation of global transition systems required for the exhaustive model checking approach. However, since a single simulation generates a single trajectory out of all the possible executions of a system, usually the average values among several simulations need to be considered to achieve the necessary level of confidence in the results obtained.

A possible way to improve the accuracy of monitoring techniques is to employ the statistical model checking that addresses general stochastic systems in terms of statistical inference. It samples the behaviors (simulations) of a model, verifies their conformance with respect to a temporal formula (i.e. performs the membership test), and finally applies a statistical estimation technique to compute an approximate value for the probability that the formula is satisfied. The accuracy of statistical model checking is affected by the accuracy of stochastic simulations techniques that are employed and also by the structure of the model or more precisely by the level of details (initial conditions, parameters, etc.) we have about the system under study.

Exhaustive model checking, statistical model checking and monitoring techniques have been applied to the study of biological systems. They allow researchers to make predictions and test hypotheses on models of different kinds (see Fig. 2). For deterministic models with continuous-value semantics the exhaustive techniques cannot be used due to the infinite number of possible executions. Therefore, advance monitoring techniques for various temporal logics have been designed in order to analyze complex non-linear systems, see [135] for a survey. These techniques have been further extended for application in systems biology. For example, the tool Breach [76] provides a coherent set of simulation-based techniques aimed at the analysis and parameter identification of deterministic models of complex biological and hybrid systems. Its primary features facilitate the computation and the property investigation of a large set of trajectories and also provide information about the sensitivity with respect to parameter perturbations. A successful application of this approach to systems biology has been demonstrated in [77] where a model of the acute inflammatory response to bacterial infection is analyzed.

A similar extension to monitoring techniques has been proposed in [86] where the authors generalize the trace-based model checking algorithm [43] to a constraint solving algorithm for QFLTL($\mathbb{R}$) with numerical constraints over the reals. Given an ODE model and a temporal property to verify within a finite time horizon, the computation of a finite simulation trace by numerical integration provides a linear Kripke structure (each state has a single successor). Afterwards, the QFLTL($\mathbb{R}$) generalization provides the ability to compute those instantiations of a formula that are true in a finite trace, by giving the complete domain of the real-valued variables occurring in the formula for which it is true. This approach has been implemented in the tool BIOCHAM [84]

Techniques for the verification of a temporal logic property against stochastic models can be either exact, based on probabilistic model checking, or approximate, based on statistical model checking using stochastic simulation such as Gillespie's algorithm [94] or Monte Carlo sampling [114,8]. Probabilistic model checking answers quantitative temporal queries by performing an exhaustive exploration of all the possible paths through the system. The probabilistic model checking techniques can be roughly divided into the techniques for discrete-time and forcontinuous-time systems. A discrete-time system is usually described by discrete-time Markov chain (DTMC) where the transitions between the states are governed by a probability distribution. The inspected properties of such systems are mostly specified in PCTL. The model checking algorithm for PCTL over DTMC constructs the parse tree of a given formula Φ and for each node it recursively computes the set of states satisfying the corresponding subformula. For more details, see, e.g., [64].

As mentioned in Section 3.1 a continuous-time system is usually described by a continuous-time Markov chain (CTMC). While each transition between states in a DTMC corresponds to a discrete-time step, transitions in a CTMC occur in real time. The transitions between the states in CTMC are governed by the transition rate matrix. It assigns a rate λ to each pair of states in the CTMC, which are used as parameters of the exponential distribution, i.e., the probability of the transition being triggered within t time-units equals $1-e^{-\lambda\cdot t}$. To reflect the real time aspects, the inspected properties of such systems are mostly specified in CSL. Efficient model checking algorithm for CSL over CTMC has been proposed in [7]. It reduces the model checking problem to the transient analysis, i.e., to the computation of transient probability, having started in state s, of being in state s' at time instant t. The reduction is based on a modification of the rate matrix such that certain states are made absorbing (all outgoing transitions are ignored) according to their validity with respect to the inspected formula. A standard technique for computing transient probabilities is based on uniformization. The key idea is for a given CTMC to construct the uniformized DTMC where all exponential delays in the CTMC are normalized with respect to the fastest transition rate q. Then each step of the uniformized DTMC corresponds to a single exponentially distributed delay with the parameter q. The ith matrix power of the uniformized DTMC gives the probability of jumping between each pair of states in the DTMC in i steps. The transient probability in time t is computed as the sum of the matrix powers weighted by Poisson probabilities giving the probability of i such steps occurring in time t. For more details about the probabilistic model checking techniques, see, e.g., [126].

The exact probabilistic model checking suffers from the state-space explosion problem, which is even more critical than in the non-probabilistic case. Therefore, for systems with too many states (more that 10^{10}) the described techniques become intractable. As result, additional reduction techniques or the statistical model checking have to be used in order to effectively analyze complex biological systems. In [42], the authors consider signal transduction in the RKIP-inhibited ERK pathway. They overcome the state-space explosion problem of probabilistic model checking by rescaling model component quantities to lower numbers of population levels. Probabilistic model checking has also been employed to the analysis of gene regulatory circuits where an automatized translation of models into a CTMC, based on quasi-steady-state approximation (QSSA), has been proposed [132].

The main problem with statistical model checking is caused by rare events, i.e., temporal formulae whose satisfaction probability is very small. When estimating the probability of such formulae, the number of simulations needed to ensure a good estimate becomes unfeasible. In [60], the authors show that the importance sampling, a variance-reduction technique for the Monte Carlo method, and the cross-entropy method, a general Monte Carlo approach to combinatorial and continuous multi-extremal optimization and importance sampling, can efficiently address this problem. They use Bounded Linear Temporal Logic, a variant of LTL where the temporal operators are equipped with time bounds, to reason about biochemical reactions in systems biology.

Both exact and approximate model checking techniques have been implemented in several tools, e.g., PRISM [125], MARCIE [152]. These tools have been successfully employed in the analysis of biological systems, e.g., in [102] the authors apply PRISM to analyze the complex FGF (Fibroblast Growth Factor) signalling pathway, in [151], the authors analyze stochastic Petri nets model of a biological network using efficient state-space representation based on interval decision diagrams. Advanced techniques for exact CSL model checking that allow to reduce the state-space explosion problem for some classes of biological systems have been implemented in the prototype tool SABRE [73].

For real-time models, model checking techniques are based on transforming the uncountable continuous-time model into an equivalent finite discrete structure (the so-called zone automaton). The two main real-time model checking tools, UPPAAL [29] and KRONOS [164], have also been used for the analysis of biological models. In the case of UPPAAL, applications to gene regulatory networks [153,97] and signaling pathways [150] have been realized. KRONOS was applied to gene regulatory networks [27] and to real-time abstractions of continuous-time deterministic models [133].

At the end of this section we briefly introduce model checking techniques for qualitative models of biological systems. These techniques have been extensively studied, see [59] for a good starting point, and there also exist several matured tools providing their efficient implementations.

Application of the qualitative model checking to systems biology is highly-relevant for Boolean models of genetic regulatory networks [50,31] and signaling networks [149,79], provided that symbolic verification techniques are usually employed. In [43], the tool BIOCHAM is used to verify the qualitative properties (specified in CTL) of asynchronous state transitions with Boolean semantics using standard symbolic model checker NuSMV [55].

Explicit model checking techniques are used in [121] where the authors propose new methodology for parameter identification and the analysis of discrete gene networks based on colored LTL model checking [13]. They improve the standard automata-based algorithm for LTL model checking [160] that consists of the following steps. The inspected LTL formula φ is negated and translated into a Büchi automaton $A_{\neg\varphi}$ describing all the executions violating φ. Afterwards, the synchronous product of $A_{\neg\varphi}$ and a finite state automaton describing the system under study is constructed. The system satisfies the formula φ if and only if the language of the product automaton is empty, which is if and only if there is no reachable accepting cycle (cycle containing an accepting state) in the underlying graph of the product automaton. Instead of employing this standard algorithm for each possible parametrization individually, the authors propose a heuristics reducing the computation effort by means of operating on entire parametrization space. A concrete application of these techniques is presented in Section 4.

Another formal method for qualitative analysis of biological systems is presented in [103], where Petri nets have been used to describe the mitogen-activated protein kinase. The authors study general properties (boundedness, liveness, reversibility, invariants), structural properties (reflecting the modeling approach) and also properties specified in temporal logic.

Model checking is also employed to qualitative abstractions of quantitative models, examples are given in Section 4. Techniques for finite discrete abstraction of the continuous state space are used in the tool BioDiVinE [18] to analyze biological models specified in terms of a set of chemical reactions. Chemical reactions are transformed into a system of multi-affine differential equations that are further discretized to a finite state automaton in order to employ the standard LTL model checking techniques including property-driven parameter identification.

3.4 Parallel and Distributed Model Checking

As already stated above model checking is a computationaly demanding procedure and techniques to fight the state explosion problem are an unavoidable ingredient of it. To verify even larger systems, however, no option was left out than to employ combined computing power of multiple computing devices. Attempts to use hard drives or parallel computers for the verification of large systems have appeared in the very early years of the automated formal verification era. However, the inaccessibility of cheap parallel computers with sufficiently fast external memory devices together with the negative theoretical complexity results excluded these approaches from the main stream in formal verification. Moreover, due to the Moore's law, the performance of software tools kept improving continuously for years as the power of a single-cored CPU grew. The situation changed dramatically with the introduction of multi-core CPU chips. The progress in computer design over the past decades had measured several orders of magnitude with respect to various physical parameters such as power consumption, efficiency, physical size or cost. As a result, it became more efficient for chip producers to introduce multiple CPU cores on a single chip rather than to increase the speed of a single core. As the speed of a single core virtually stopped growing, every piece of software that was built upon a serial algorithm could not take the advantage of technological progress anymore. The focus of parallel and distributed-memory computing community shifted away from unique massively parallel systems competing for world records towards smaller and more cost-effective systems built up from small and cheap personal computer parts. Suddenly, the need for parallel processing became rather general and widespread in all science fields relying on complex computation operations, automated formal verification being not an exception. As a matter of fact, the interest in the *platform-dependent* formal verification has been revived.

Unfortunately, some verification techniques cannot preserve their efficiency if adapted to non-sequential models of computation, and therefore an urgent need for new and quite different verification procedures emerged. Many new techniques have been introduced. There were attempts to consider both the symbolic as well as the enumerative techniques, theorem-provers as well as sat-solvers, etc. Some of those approaches are applicable across a broad range of computing platforms, some of them are tailored to the specific capabilities of a particular hardware architectures. Examples include techniques to fight the memory limits with an efficient utilization of external memory devices [156], techniques that introduce cluster-based algorithms to employ the aggregate power of network-interconnected computers [155,129,92], techniques to speed-up the verification process on multi-core processors [109,14,128], etc.

Parallel Algorithms for LTL Model Checking. The need for parallel processing in automated formal verification stemmed from the desire to fight the state space explosion problem by employing the aggregate memory of multiple network interconnected workstations. The crucial problem is how to distribute the work among participating processors in order to take advantage of the aggregate memory and parallel processing at the same time.

Based on a parallel algorithm for state space generation [47], a static partitioning scheme relying on a hash function was introduced [52]. As observed by multiple researchers, the hash-based partitioning yields better space locality if only some parts of the state descriptor are used as the input to the partitioning function. There were considered approaches requiring the user of the tool to specify the concrete parts of the state descriptor to be used for partitioning [52], other approaches employed automated or semi-automated techniques to do it [53]. Techniques for load balancing the set of visited states, also known as re-partitioning techniques, have been suggested [1,130,124] as well as state space generation schemes employing probability aspects [122].

The first known public implementation of a distributed memory tool for the verification of communication protocols was the parallel implementation of the Murφ tool [155]. Murφ's parallel work-flow relied on the standard MPI-like approach to messaging, nevertheless, active messages were later introduced into Murφ to improve its efficiency. The successful story of Murφ was followed by other verification tools: SPIN [130], CADP [92], DiVinE [19], UPPAAL [28], etc. Distributed-memory techniques of automated formal verification also appeared in the context of Petri Nets [52,104], Markov chains [101], and symbolic BDD-based model checkers [98,83].

As a demonstration of distributed-memory approaches to verification we consider explicit state parallel LTL model checking. The LTL model checking problem can be reformulated as a cycle detection problem in an oriented graph and the basic principles behind presented algorithms rely on efficient solutions to detecting cycles in a distributed environment. The best known enumerative sequential algorithms for the detection of accepting cycles are the *Nested DFS* algorithm [63] (implemented, e.g. in the model checker SPIN [107]) and *SCC-based algorithms* originating in Tarjan's algorithm for the decomposition of the graph into strongly connected components (SCCs) [158]. While Nested DFS is more space efficient, SCC-based algorithms produce shorter counterexamples in general. The linear time complexity of both algorithms relies on the postorder as produced by the depth-first search traversal. It is a well known fact that computing depth-first search postorder is P-complete [146], hence probably inherently sequential. This means that none of the two algorithms can be easily adapted to work on a parallel machine. A few fundamentally different cluster-based techniques for accepting cycle detection appeared, though. They typically perform repeated reachability over the graph. Unlike the postorder problem, reachability is a graph problem which can be parallelized, hence the algorithms might be transformed to cluster-based algorithms that work with reasonable increase in time and space.

The algorithms employ specific structural properties of the underlying graphs (often pre-computed in advance from the system specification), use additional data structures to divide the problem into independent sub-problems, or translate the model-checking problem to another one, which admits efficient parallel solution. Several of the algorithms are based on sequentially less efficient but well parallelizable breadth-first exploration of the graph or on placing bounds limiting the size of the graph to be explored.

The first parallel algorithm for LTL model checking employed the so-called dependency structure [17] to record the reachability relation among accepting states of a distributed graph and applied the topological sort algorithm [117] to detect the presence of a self-reachable accepting state. Other parallel algorithms appeared with the time, building upon various ideas. They have differed in theoretic complexity as well as practical efficiency, see [39] for a survey. The two most successful parallel algorithms for LTL model checking are the OWCTY algorithm [48] based on explicit-state implementation of symbolic SCC hull detection and the MAP algorithm [38] based on value propagation.

Distributed-memory processing cannot attack the state space explosion problem alone and must be combined with other techniques. One of the most successful techniques to fight the state space explosion in explicit-state model checking is *Partial Order Reduction* [140]. DiVinE is able to perform this reduction, even though a new topological sort proviso had to be developed in order to maintain efficiency of parallel and distributed-memory processing [16].

Another important algorithmic improvement relates to the classification of LTL formulas. For some classes of LTL formulas (weak LTL), the parallel algorithms may by significantly improved. With this observation the OWCTY algorithm can be improved so that its complexity even meets the complexity of the optimal sequential Nested DFS algorithm and it allows for on-the-fly verification in most verification instances [15].

Parallelism in Distributed and Shared-Memory. The general idea in distributed-memory explicit state model checking is to aggregate the computational power of multiple network interconnected workstations (clusters) in order to facilitate the verification of large model checking instances [17]. The set of vertices of the graph to be processed is partitioned among participating computation nodes using a static partitioning function. When a computation node processes a vertex, it enumerates all its immediate successors and checks them for their ownership. If a newly generated vertex is local according to the partitioning function, it is pushed to the local queue where it waits for further processing. Otherwise a network message containing the vertex is created and sent to the queue of the owning computation node. With this work-flow, a message is generated with every edge connecting vertices from different partitions of the graph.

Message aggregation and buffering are the standard techniques in parallel computing to alleviate the burden of network communication overhead. Therefore, the model checker maintains buffers of messages to be sent to individual computing nodes. A buffer is flushed (messages sent to network) upon one of the following situations: 1) the buffer was explicitly flushed by the executed graph algorithm, 2) the maximal number of messages for the buffer has been reached, and 3) the local computing node was (otherwise) idle.

Most techniques and results known from the distributed-memory setting are straightforwardly applicable to shared-memory architectures. In particular, the graph to be processed is partitioned among individual parallel shared-memory threads in the same way as it would be in the distributed-memory setting. Each individual thread maintains its own hash table and its own pool of vertices to be processed. Vertices belonging to different threads are pushed to their local pools by means of lock-free shared-memory queues [14]. Relative advantages and disadvantages of shared versus private hash tables, within the context of thread-private pools of vertices to be processed, have been discussed in [22].

Nevertheless, the scalability of parallel distributed-memory solutions to shared-memory is often limited. Therefore, shared-memory specific techniques are needed to improve the efficiency and scalability of existing parallel distributed-memory solutions on shared-memory architectures. Examples of successful shared-memory specific techniques include, e.g. shared communication data structures [111,14], specific termination detection techniques [14], dual-core algorithms [109], or quite a unique partitioning scheme [108].

Many-Core Parallelism. After NVIDIA's CUDA technology [66] was introduced, a lot of computational demanding tasks have been accelerated by GPU-aware algorithms. Examples of GPU accelerated procedures include, but are not limited to, sorting [148], sparse matrix-vector multiplication [51], or numerous biological and physical simulations, such as protein folding [113]. As for the graph theory, successful adaptation of general graph traversal algorithms have been reported too [138] demonstrating the tremendous computational power of the CUDA device. On the other hand, graphs to be explored efficiently with a CUDA accelerated algorithm must be encoded explicitly in a compact way.

The CUDA technology as a computing platform, attracted also researches in the field of automated formal verification. The key challenge, for which no satisfactory solution is known yet, is how to accelerate the generation of explicitly encoded state space graph from implicit definition. Preliminary attempts to do so relate to explicit model checking. They suggest to employ a massively parallel check for enabled transitions emanating from the vertices on the frontier of the search and their massively parallel execution [78].

Once the state space is generated and explicitly represented in an appropriate sparse matrix-like structure, many verification tasks can be accelerated using CUDA technology. This has been successfully demonstrated, e.g. on verification of probabilistic systems [35], LTL model checking [23] or the acceleration of strongly connected components decomposition [12].

3.5 Model Checking Tools for Biological Systems

There are several specialized tools for the analysis of biological systems that employ model checking. Some of these tools are well accepted by the community and routinely used in the process of model development. In addition, several model checking tools were experimentally used for the analysis of models in systems biology. In this section we point to three of them that we found to be closest to our own interest in exhaustive model checking. For richer reviews we would like to refer to [46,10,112].

BioCham (Fages et al. [85], see [84] for tutorial)

BioCham stands for BIOCHemical Abstract Machine. The tool provides a modeling environment for systems biology, with some unique features for static analysis or for inferring unknown model parameters from temporal logic constraints. BioCham covers qualitative (Boolean) models as well as quantitative models (continuous-time deterministic and continuous-time stochastic). Models are specified in its native rule-based language. An important feature is that quantitative models specified at the level of reaction networks can be automatically analyzed at the level of qualitative (Boolean) semantics.

Qualitative Models. CTL is employed to formalize the temporal properties of a biological system and validate models with respect to such specifications. Symbolic model checker NuSMV [54] is used to handle this analysis task. Moreover, BioCham has an update component for automatically modifying a network that does not satisfy a given CTL formula. The algorithm of this component is based on the counterexamples computed by NuSMV. Although incomplete (in the sense of sometimes not being able to find the appropriate changes to networks), such a component is useful because of being able to handle large networks [43].

Continuous-time Deterministic Models. BioCham introduces LTL with numerical constraints (LTL($\mathbb{R}$)) to specify properties of numerical simulations of ODE models. Since simulations always produce finite discretely-sampled trajectories bounded by the requested time horizon, there is a natural monitoring algorithm built in. Furthermore, the tool is able to compute the violation degree of a formula. Intuitively, a violation degree is the distance between a particular behavior of a system, given as a path, and the expected behavior, given as a temporal-logic formula [147]. Such a violation measure can be used to estimate a fitness function with evolutionary optimization methods. This is done by finding kinetic parameter values satisfying a set of biological properties formalized in temporal logic. In addition, such a measure can be used to estimate the robustness of a biological model with respect to its temporal specification.

Finally, probabilistic model checking is also provided. BioCham estimates the probability of an LTL formula satisfaction by sampling stochastic simulations.

GNA (de Jong et al. [116])

Genetic Network Analyzer (GNA) provides support for modeling and simulation of genetic regulatory networks using knowledge about regulatory interactions in combination with gene expression data. GNA operates on piece-wise affine models providing a clear relation between quantitative and qualitative semantics. Instead of exact numerical values for the parameters, which are often not available for gene networks, the piece-wise affine models allow to specify inequality constraints. This information is sufficient to generate a state transition graph that describes the qualitative dynamics of the network overapproximating the ODE model.

GNA is able to export the resulting qualitative model to the finite state transition system and check properties by means of standard model-checking tools, either locally installed or accessible through a remote web server. The tool is connected with NuSMV and CADP (Garavel et al. [91]) model checkers. GNA supports an extension of CTL logic, CTRL [136], allowing to express a significant set of biologically relevant properties not expressible in plain CTL.

Additionally, parameter identification techniques have also been introduced for GNA [26]. Based on symbolic model checking, the method avoids enumerating all possible parametrizations in searching for parametrizations satisfying the given temporal specification.

BioDiVinE (Barnat et al.[18,20])

BioDiVinE[1] is a tool-box for automated analysis of biological systems by means of applying model checking to qualitative and quantitative biological models. Emphasis is put on the computational aspects and algorithms are adapted to enable their effective distribution and/or parallelization. Currently, the tool-box contains the following tools:

BioDiVinE 1.0 is a tool created for model checking LTL properties over continuous-time deterministic biological models given by means of multi-affine ODEs ([18], see Section 3.1), which are abstracted by employing the rectangular abstraction [62], translating the continuous model into a finite automaton.

[1] `http://sybila.fi.muni.cz/tools`

The tool makes an evolutionary branch of enumerative LTL model checker DiVinE [19] by adapting the OWCTY [48] and MAP [38] algorithms for distributed analysis of biological models. Properties are specified by means of Büchi automata allowing even more flexibility than is provided by LTL.

Parsybone and PEPMC are tools for property-driven identification of biological models. Parsybone focuses on logical parameters in qualitative models [121], in particular, in gene regulatory networks encoded using the formalism of R. Thomas [159]. PEPMC provides parameter identification for quantitative models [13], in particular, continuous-time deterministic models represented as piecewise multi-affine ODEs (this model class generalizes multi-affine models to capture regulatory dynamics). Both tools are based on colored LTL model checking technique providing a heuristics for effective exploration of models with finite parametrizations. As in BioDiVinE 1.0, specification of the temporal properties is realized by means of Büchi automata.

PARASIM is a tool for approximative analysis of robustness of continuous-time deterministic models. For sets of perturbations of kinetic parameters (or initial conditions) and temporal properties specified in Signal Temporal Logic (STL) [134], the so-called landscape function, giving the property's validity for parametrizations in the required perturbation set, is computed.

Usage of the BioDiVinE toolset is demonstrated in Section 4.

4 Model Checking in Action – Application Examples

In this section we give case studies on qualitative and quantitative representations of two biologically relevant models. The main purpose is to demonstrate the application of selected techniques based on model checking. We focus on techniques implemented in the BioDiVinE tool set, in particular, explicit LTL model checking and parameter identification techniques built on the top of it. Additionally, we provide a demonstration of probabilistic model checking recently extended to property-driven exploration of model parameters.

4.1 *E. coli* Ammonium Transport Model

We consider a simple biological model that describes ammonium transport from the external environment into the cells of *Escherichia Coli.* This simplified model is based on a published model of the *E. Coli* ammonium assimilation system [131].

The model is a typical example of the dynamical models appearing in current computational systems biology. In particular, the model is represented as a reaction network associated with a continuous-time deterministic semantics given in terms of (non-linear) ODEs. Parameters were taken from the literature.

We employ model checking to explore the model dynamics from a global perspective. The term *global* has two meanings here. First, we want to analyze the model dynamics starting at any possible initial concentration of the species, not only at a single initial condition, as allowed by traditionally used simulation methods. Second, we want to explore the model dynamics without restricting ourselves to a given parametrization. In particular, we want to explore how parameters affect the expected (or required) behavior of the model.

As stated in Section 3.1, exhaustive exploration of the system states cannot be directly employed on continuous-time deterministic models due to the uncountability of time and variable domains. In order to allow the model checking analysis, the model must be simplified in terms of Section 3.1. In this particular case, we employ the rectangular abstraction technique [62] that allows to transform an ODE model of a specific class into a finite automaton provided that the dynamics properties are (conservatively) preserved.

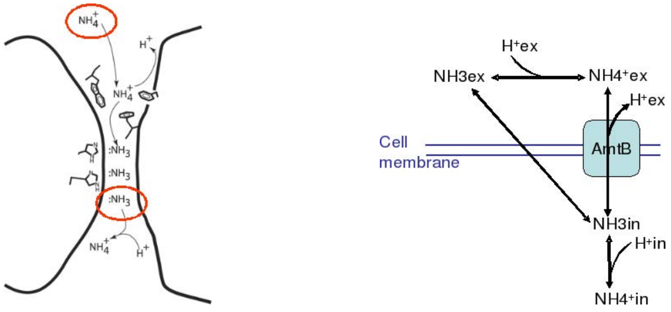


Fig. 3. *E. Coli* ammonium transport mechanism and the respective pathway

Model Description. *E. coli* can express membrane-bound transport proteins for the transportation of small molecules from the environment into the cytoplasm at certain conditions. At normal ammonium concentration, the free diffusion of ammonium can provide enough flux for the growth requirement of nitrogen. When ammonium concentration is very low, E. coli cells express *AmtB* (an *ammonium transporter*) to complement the deficient diffusion process. Three molecules of *AmtB* (trimer) form a channel for the transportation of ammonium. Protein structure analysis revealed that *AmtB* binds $NH4^+$ at the entrance gate of the channel, deprotonates it and conducts NH_3 into the cytoplasm as illustrated in Figure 3 (left) [119]. At the periplasmic side of the channel there is a wider vestibule site capable of recruiting NH_4^+ cations. The recruited cations are passed through the hydrophobic channel where the pKa of NH_4^+ was shifted from 9.25 to below 6, thereby shifting the equilibrium toward the production of NH_3. NH_3 is finally released at the cytoplasmic gate and converted to NH_4^+ because the intracellular pH (7.5) is far below the pKa of NH_4^+.

In addition to the above mentioned *AmtB* mediated transport, the bidirectional free diffusion of the uncharged ammonium through the membrane is also included in the simplified model. The intracellular NH_4^+ is then metabolised by Glutamine Synthetase (GS). The whole model is depicted in Figure 3 (right). The external ammonium is represented in the uncharged and charged forms denoted

Table 1. The model of ammonium transport

$AmtB + NH_4ex \xleftrightarrow{k_1\,k_2} AmtB : NH_4$	$k_1 = 5 \cdot 10^8, k_2 = 5 \cdot 10^3$
$AmtB : NH_4 \xrightarrow{k_3} AmtB : NH_3 + H_{ex}$	$k_3 = 50$
$AmtB : NH_3 \xrightarrow{k_4} AmtB + NH_3in$	$k_4 = 50$
$NH_4in \xrightarrow{k_5}$	$k_5 = 80$
$NH_3in + H_{in} \xleftrightarrow{k_6\,k_7} NH_4in$	$k_6 = 1 \cdot 10^{15}, k_7 = 5.62 \cdot 10^5$
$NH_3ex \xleftrightarrow{k_8\,k_9} NH_3in$	$k_8 = k_9 = 1.4 \cdot 10^4$

NH_3ex and NH_4^+ex. Analogously, the internal ammonium forms are denoted NH_3in and NH_4^+in. The reaction network that combines $AmtB$ transport with NH_3 diffusion is given in Table 1.

The reaction network is assigned a set of ODEs as listed in Table 2 (employing the law of mass action kinetics). It is worth observing that the form of the ODE right-hand sides is in all cases made by polynomials of degree one. Since we are especially interested in how the concentrations of internal ammonium change with respect to the external ammonium concentrations, we employ the following simplifications:

- We do not consider the dynamics of the external ammonium forms, thus we take NH_3ex and NH_4^+ex as constants (the input parameters for the analysis).
- We assume constant intracellular pH (7.5) and extracellular pH (7.0), thus H_{ex} and H_{in} are calculated to be $3 \cdot 10^{-8}$ and 10^{-7}. Based on the extracellular pH and the total ammonium concentration, concentrations of NH_3ex and NH_4^+ex can be calculated.

Without loss of correctness, we simplify the notation of the cation NH_4^+ as NH_4.

Table 2. The mathematical model of ammonium transport

$$
\begin{gathered}
\frac{d[AmtB]}{dt} = -k_1 \cdot [AmtB] \cdot [NH_4ex] + k_2 \cdot [AmtB : NH_4] + k_4 \cdot [AmtB : NH_3] \\
\frac{d[AmtB{:}NH_3]}{dt} = k_3 \cdot [AmtB : NH_4] - k_4 \cdot [AmtB : NH_3] \\
\frac{d[AmtB{:}NH_4]}{dt} = k_1 \cdot [AmtB] \cdot [NH_4ex] - k_2 \cdot [AmtB : NH_4] - k_3 \cdot [AmtB : NH_4] \\
\frac{d[NH_3in]}{dt} = k_4 \cdot [AmtB : NH_3] - k_7 \cdot [NH_3in] + k_6 \cdot [NH_4in] \\
\frac{d[NH_4in]}{dt} = k_5 \cdot [NH_4in] + k_7 \cdot [NH_3in] \cdot [H_{in}] - k_6 \cdot [NH_4in]
\end{gathered}
$$

Model Simplification. The restricted polynomial form of ODEs implies that the model falls into the class of so-called multi-affine systems for which a simplification, the so-called rectangular abstraction, is defined [30] (see [62] for the relation with model checking). Each variable is assigned a set of specific (arbitrarily defined) points, the so-called *thresholds*, expressing concentration levels of

$$A \overset{k_1\ k_2}{\longleftrightarrow} B$$

$$\frac{dA}{dt} = -k_1 \cdot A + k_2 \cdot B$$
$$\frac{dB}{dt} = k_1 \cdot A - k_2 \cdot B$$

thresholds on A: $0, 5, 6, 10$
thresholds on B: $0, 2, 3, 5$

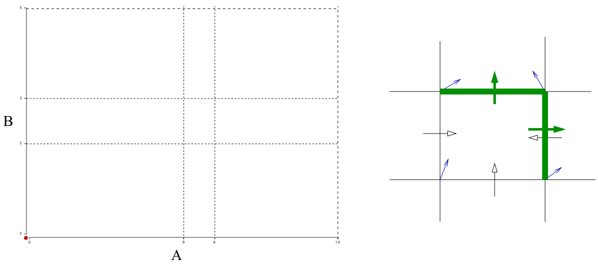


Fig. 4. Example of a rectangular partition of a two-dimensional system (left) and the intuition behind the construction of the abstracted transition system (right)

special interest. This set contains two specific thresholds – the maximal and the zero concentration level (bounding of the state space has been discussed in Section 3.1). The intermediate thresholds then define a partition of the (bounded) continuous state space. The individual regions of the partition are called *rectangles.* An example of a partition is given in Figure 4.

The partition of the system gives us directly the finite discrete abstraction of the dynamic system. In particular, the BioDiVinE tool implements a (discrete) state space generator that constructs a finite automaton representing the rectangular abstraction of the system dynamics. Since the states of the automaton are made by the rectangles in the phase-space partition, the automaton is called *rectangular abstraction transition system* (RATS). The main point is that for each rectangle the exit faces are determined. The intuition is depicted in Figure 4(right). There is a transition from a rectangle to its neighbouring rectangle only if, in the vector field considered in the shared face, there is at least one vector whose particular component agrees with the direction of the transition. The important result is that in a multi-affine system it suffices to consider only the vector field in the vertices of the face. In Figure 4(right), the exit faces of the central rectangle are emphasised by bold lines. In Figure 5 there is depicted the rectangular abstraction transition system constructed for the affine system from Figure 4(left). It is known that the rectangular abstraction is an overapproximation with respect to trajectories of the original dynamic system.

There is one specific issue when considering the time progress of the abstracted trajectories. If there exists a point in a rectangle from which there is no trajectory diverging out through some exit face, then there is a self-transition defined for the rectangle. In particular, this situation signifies an equilibrium inside the rectangle. Such a rectangle is called non-transient. For affine systems, a sufficient and necessary condition is known, that characterizes non-transient rectangles by the vector field in the vertices of those rectangles. However, for multi-affine systems, only the necessary condition is known. Hence, for multi-affine systems BioDiVinE treats as non-transient some states which are not necessarily non-transient.

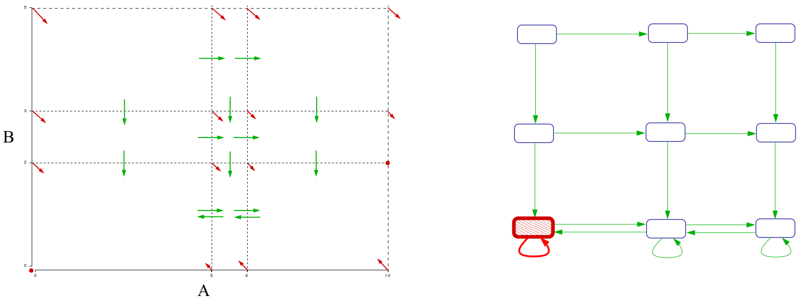


Fig. 5. Example of a rectangular abstraction transition system. The emphasized state and transition make a counterexample contradicting the property $\mathbf{F}(B > 3)$.

We use LTL logic to encode the dynamics properties of the model. Given a dynamic system S with a particular initial state we can then say that S satisfies a formula φ, written $S \models \varphi$, only if the trajectory starting at the initial state satisfies φ. In the context of automata, LTL logic is interpreted universally provided that a formula φ is satisfied by the automaton A, written $A \models \varphi$, only if each execution of the automaton starting from any initial state satisfies φ. The following theorem characterizes the relation between validity of φ in the rectangular abstraction automaton and in the original dynamic system, taken from [62].

Theorem 1. *Consider a dynamic system S and the associated RATS A. If $A \models \varphi$ then $S \models \varphi$.*

The theorem states that when the model checking of a particular property on a RATS returns true, we are sure that the property is satisfied in the original dynamic system. However, when the result is negative, the counterexample returned does not necessarily reflect any trajectory in the original system.

The system in Figure 5 satisfies a formula $\mathbf{FG}(B \leq 3)$ expressing the temporal property stating that whatever the choice of the initial state, the system eventually stabilizes at states where concentration of B is kept below 3. Now let us consider a formula $\mathbf{F}(B > 3)$ expressing the property that whatever the initial settings, the concentration of B will eventually exceed the concentration level 3. In this case the model checking returns one of the counterexamples as emphasized in Figure 5(right) stating that if initially $A < 5$ and $B < 3$ then B is not increased while staying indefinitely long in the emphasised state.

We apply the rectangular abstraction method to the ammonium transport model. We consider the set of states from which we want to explore the dynamics given by the following intervals of concentration values:

$$AmtB \in \langle 0, 1{\cdot}10^{-5}\rangle, \quad AmtB:NH_3 \in \langle 0, 1{\cdot}10^{-5}\rangle, \quad AmtB:NH_4 \in \langle 0, 1{\cdot}10^{-5}\rangle,$$
$$NH_3in \in \langle 1 \cdot 1^{-6}, 1.1 \cdot 10^{-6}\rangle, \quad NH_4in \in \langle 2 \cdot 10^{-6}, 2.1 \cdot 10^{-6}\rangle$$

Table 3. Partitioning of the continuous state space

$AmtB$	$0,\ 10^{-12},\ 10^{-10},\ 9.9 \cdot 10^{-8},\ 10^{-7},\ 5 \cdot 10–6,\ 10^{-5}$
$AmtB : NH_3$	$0,\ 10^{-7},\ 10^{-5}$
$AmtB : NH_4$	$0,\ 10^{-7},\ 10^{-5}$
NH_3in	$0,\ 10^{-6},\ 1.1 \cdot 10^{-6},\ 3 \cdot 10^{-6},\ 8 \cdot 10^{-6},\ 10^{-5}$
NH_4in	$0,\ 2 \cdot 10^{-6},\ 2.1 \cdot 10^{-6},\ 10^{-5},\ 5 \cdot 10^{-4},\ 5.3 \cdot 10^{-4},\ 5.4 \cdot 10^{-4},\ 10^{-3}$

The upper bounds as well as the intervals of internal ammonium forms have been set with respect to the available data obtained from the literature. The partition used for rectangular abstraction has been set by thresholds as given in Table 3.

Model Checking Analysis. From the essence of biophysical laws, it is clear that the maximal reachable concentration level accumulated in the internal ammonium forms directly depends on the ammonium sources available in the environment. However, it is not directly clear what particular maximal level of internal ammonium is achievable at given amount of external ammonium (distributed into the two forms). In the analysis we have focused on just this phenomenon. More precisely, the problem to solve was to analyze how the setting of the model parameters NH_3ex and NH_4^+ex affects the maximal concentration level of NH_3in and NH_4^+in reachable from given initial conditions.

It is very difficult to provide *in vitro* measurements of $AmtB$ concentration (and also the concentration of dimers $AmtB : NH_3$ and $AmtB : NH_4$). This gives a strong motivation to analyze the model globally (with uncertain initial conditions).

We have conducted several model checking experiments in order to determine the maximal reachable concentration levels of NH_3in and NH_4^+in. In particular, we have searched for the lowest α satisfying the property $\mathbf{G}(NH_3in < \alpha)$ and the lowest β satisfying $\mathbf{G}(NH_4in < \beta)$. The property $\mathbf{G}p$ requires that all paths available in the rectangular abstraction from the states specified by the initial condition must satisfy the given proposition p at every state. Note that if the model checking method finds the property $\mathbf{G}p$ false in the model, it also returns a counterexample for that. The counterexample satisfies the negation of the checked formula, which is in this case $\mathbf{F}\neg p$. Interpreting this observation intuitively for the above formulae, we use model checking to find a path on which the species NH_3in (resp. NH_4in) exceeds the level α (resp. β).

The procedure was the following: At the starting point, we substituted for α (resp. β) the upper initial bounds of the respective variables. Then we found the requested values by iteratively increasing and decreasing α (resp. β). The obtained results are summarized in Table 4.

The results have shown that NH_3in does not exceed its initial level no matter how the external ammonium is distributed between NH_3ex and NH_4^+ex. The upper bound concentration considered for both NH_3ex and $NH4^+ex$ has been

Table 4. Experiments on detecting maximal reachable levels of internal ammonium

α	$\mathbf{G}(NH_3in < \alpha)$	# states	Time
$1.1 \cdot 10^{-6}$	true	1081	0.36 s

β	$\mathbf{G}(NH_4in < \beta)$	# states	Time
$1 \cdot 10^{-3}$	true	2161	0.45 s
$5 \cdot 10^{-4}$	false	4753	1.9 s
$6 \cdot 10^{-4}$	true	2161	0.43 s
$5.4 \cdot 10^{-4}$	true	1441	0.27
$5.3 \cdot 10^{-4}$	false	3421	1.2 s

set to $1 \cdot 10^{-5}$ which corresponds to common concentration level of the gas in the cell environment.

In the case of NH_4in we have found that the upper bound to maximal reachable level is in the interval $\beta \in \langle 5.3 \cdot 10^{-4}, 5.4 \cdot 10^{-4} \rangle$. Since the counterexample achieved can be a spurious one due to the overapproximating abstraction, the exact maximal reachable value may be lower. This can be explored by numerical simulation, an important fact is that the range for setting β is now limited to the detected interval.

The results have been achieved by running the OWCTY algorithm on a single computation node. In [18], there is presented a refined variant (a finer partition) of the model leading to 10^5 reachable states. For that variant, distributing of the computation to 36 nodes was needed to achieve good times (in the order of seconds).

Parameter Exploration. Owing to the membrane location of *AmtB*, *in vitro* measuring of the concentration of *AmtB*-based species is impossible and therefore the estimation of kinetic parameters of this model is very difficult.

To identify parameter values computationally, we employ the colored model checking technique [13] implemented in the PEPMC tool of the BioDiVinE toolset (see Section 3.5). If we denote each parametrization by a distinct color and assume that the respective (parametrization-specific) state space has all its transitions marked by this color, we can construct a global state space as a union of all the parametrization-specific state spaces. Since a change in parameter values affects the model dynamics, which is entirely represented by state transitions, the parameter space is completely projected onto the transition relation defined on the universal state space. In this setting, our solution to the parameter identification problem is based on analysis of mono-colored paths in a graph with multi-colored edges. Since in many parametrizations small perturbations in parameter values lead to small locally distributed variations in the transition relation, the respective mono-colored graphs can exhibit significant similarity. For parametrizations amenable to such property, the algorithm achieves good efficiency. In Fig. 6, the basic idea of solving the parameter identification problem by automata-based LTL model checking is illustrated. The automaton

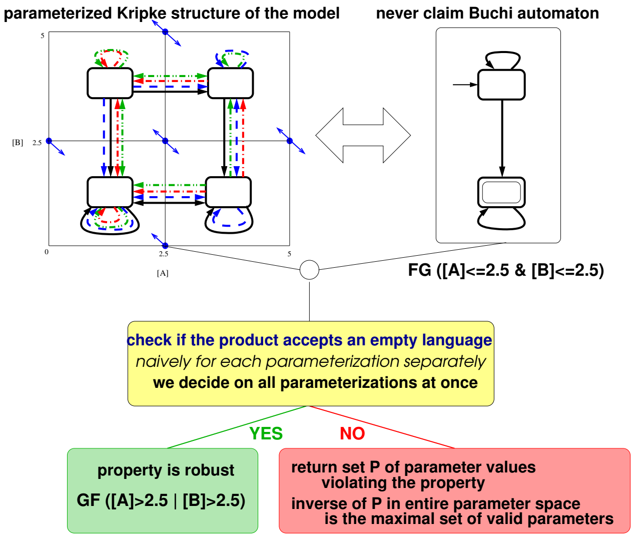
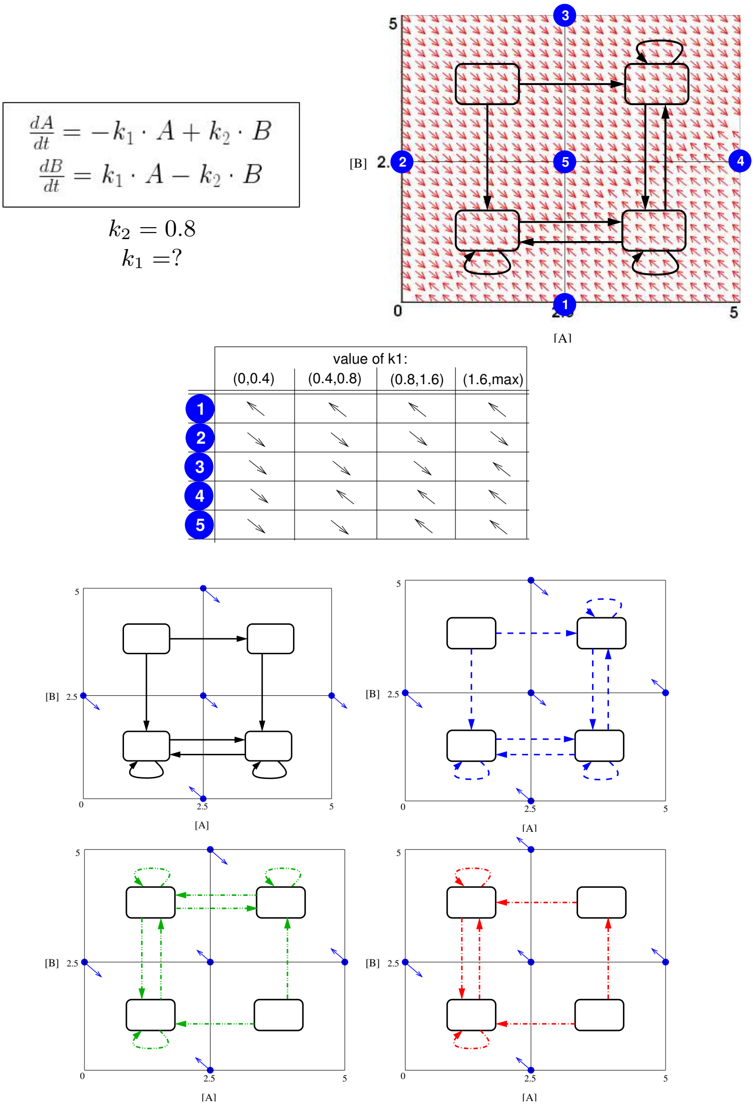

parameterized Kripke structure of the model
never claim Buchi automaton
5
[B] 2.5
0
2.5
5
[A]
FG ([A]<=2.5 & [B]<=2.5)
check if the product accepts an empty language
naively for each parameterization separately
we decide on all parameterizations at once
YES
NO
property is robust
GF ([A]>2.5 | [B]>2.5)
return set P of parameter values violating the property
inverse of P in entire parameter space is the maximal set of valid parameters

Fig. 6. Intuition behind colored model checking

representing the model dynamics can be extended with colored edges, where each color corresponds to a certain parametrization. We expect that transitions are to a large extent shared among individual colors. This allows us to accelerate the computation.

It is important to note that in the considered model the parameters are quantitative and their domain is uncountable (but bounded). However, as shown in [25], the rectangular abstraction partitions the parameter domains into a finite number of intervals where each interval contains parameter values producing an isomorphic state transition system. Intuition behind the application of this result is illustrated in Fig. 7.

In our model, we investigate the effect of different parameter settings to the production of the model output – the internal ammonium forms NH_3in and NH_4in. In particular, we look for perturbations in individual kinetic parameters that lead to an increase of internal ammonium concentration above the standard values. In the terms of LTL model checking, we formulate the negation of this requirement – we check whether the standard value is never exceeded. We formalize the discussed requirement by safety LTL properties $\varphi_1 = \mathbf{G}(NH_3in < 1.1 \cdot 10^6)$ and $\varphi_2 = \mathbf{G}(NH_4in < 2.1 \cdot 10^6)$ stating that NH_3in (resp. NH_4in) never exceeds the given concentration. We performed two sets of parameter identification tasks. In the first group, each single parameter was considered unknown (remaining parameters were set w.r.t. literature [131]). In the second group, we considered a collection of three unknown parameters. In all experiments, the range for every parameter was set to $(1 \cdot 10^{-12}, 1 \cdot 10^{12})$. In Table 5, the most interesting results



$$\frac{dA}{dt} = -k_1 \cdot A + k_2 \cdot B$$
$$\frac{dB}{dt} = k_1 \cdot A - k_2 \cdot B$$

$k_2 = 0.8$
$k_1 = ?$

	value of k1:			
	(0,0.4)	(0.4,0.8)	(0.8,1.6)	(1.6,max)
1	↖	↖	↖	↖
2	↘	↘	↘	↘
3	↘	↘	↘	↖
4	↘	↖	↖	↖
5	↘	↘	↖	↖

Fig. 7. Partitioning of uncountable parameter space into a finite number of intervals. Parameter k_1 is considered to be unknown. Rectangular abstraction of the model is determined by thresholds $0, 2.5, 5$ imposed on both species A and B producing five intersection points. By substituting any of these points into the ODEs while setting the left-hand sides to zero (equilibrium), we can solve the resulting homogeneous system of linear equations for k_1. The solution gives us those values of k_1 where the sign of any of the derivatives changes. By iterating this procedure for each of the points, the domain of k_1 is partitioned into four intervals as can be seen in the table above. Each of the intervals makes a class of equivalence with respect to the derivative sign in a particular intersection point. Accordingly, in this example we get four (qualitatively) different automata abstracting the model dynamics.

Table 5. Parameter exploration experiments

P	prop.	intervals of validity	# states reached	time
k_4	φ_1	$(1 \cdot 10^{-12}, 2.7 \cdot 10^6)$	124580	30 s
k_5	φ_2	$(1.5 \cdot 10^7, 1 \cdot 10^{12})$	3068	0.40 s
k_6	φ_1	$(5.2 \cdot 10^6, 1 \cdot 10^{12})$	67572	22 s
k_6	φ_2	$\emptyset$	6319	1.8 s
k_7	φ_1	$(1 \cdot 10^{-12}, 3.3 \cdot 10^6)$	126458	33 s
k_7	φ_2	$(1.6 \cdot 10^7, 1 \cdot 10^{12})$	12523	3.5 s
k_9	φ_1	$(1 \cdot 10^{-12}, 2.7 \cdot 10^6)$	97495	20 s
k_9	φ_2	$\emptyset$	5779	1.5 s
$k_{1,6,9}$	φ_1	$k_9 \in (1 \cdot 10^{-12}, 2.7 \cdot 10^6) \vee$ $[k_9 \in (2.7 \cdot 10^6, 3.2 \cdot 10^6) \wedge k_6 \in (1 \cdot 10^{-12}, 1.07 \cdot 10^6)]$	202638	51 min
$k_{1,6,10}$	φ_2	$[k_1 \in (1 \cdot 10^{-12}, 1 \cdot 10^7) \wedge k_6 \in (1 \cdot 10^{-12}, 1.4 \cdot 10^5) \wedge k_{10} \in (1.18 \cdot 10^6, 1 \cdot 10^{12})]$ $\vee[k_6 \in (1.4 \cdot 10^5, 1.07 \cdot 10^6) \wedge k_{10} \in (1 \cdot 10^{-12}, 1.18 \cdot 10^5)]$	19473	19 min

are summarized. The presented data show the scanned parameter set, the analyzed property with the computed valid parametrizations, number of reached states, and computation times. Note that if a parameter is not mentioned it led trivially to validity on the entire $(1 \cdot 10^{-12}, 1 \cdot 10^{12})$.

Of special interest are individual scans of k_6 and k_9 for φ_2. In particular, the results show that, in the given parameter value range, there is no perturbation which would satisfy the property. With respect to both parameters, the model is robust in the negative property $\mathbf{F}(NH_4in > 2.1 \cdot 10^6)$ stating that NH_4in eventually exceeds the given concentration. Thus, regardless the setting of k_6, k_9, on each trajectory leading from the range specified by initial conditions, NH_4in must exceed the standard concentration range.

4.2 Gene Regulation of Mammalian Cell Cycle

In the second case study, we focus on parameter identification by model checking for a model of a regulatory network. In particular, we investigate a model representing the central module of the genetic regulatory network governing the G_1/S cell cycle transition in mammalian cells [157]. In particular, the model considers a two-gene network describing interaction of the tumor suppressor protein pRB and the central transcription factor $E2F1$ (see Fig. 8(left)).

This simple model demonstrates the feature of bistability, i.e., the occurrence of two stable states. Bistable networks can drive the systems response to some stimulus: With no stimulus, the system keeps (once reaching it) a certain stable state. In particular, in the stable state, the concentrations of the species does not change, because production and degradation had reached an equilibrium. A stimulus, which is in the form of a change of some protein concentration caused from outside the system, evokes deflection from the stable state. If the deflection is weak enough, the system returns to the previous stable state afterwords. But when it overruns some threshold, the system approaches the other stable state, with no chance of returning to the first one, even after the end of the stimulus. That way the system can decide whether a stimulus is strong enough to permanently switch to a particular mode. The first stable state of our system represents the low level of the $E2F1$ protein concentration. In this state, the cell stays in G_1-phase. When increased enough, the concentration of $E2F1$ grows higher, to the level of the second stable state, which causes the cell approaches to S-phase.

Qualitative Model. First, we consider a Boolean model of the network, we formulate the required properties, and we employ the parameter identification algorithm to find parametrizations of unknown parameters (denoted by question marks in Fig. 8).

We employ the Boolean model of gene regulatory networks as introduced by Thomas [159]. The concrete modeling approach including the parametrization is taken from [13]. The *Boolean model* is determined by the structure (topology) of the GRN and the regulatory logic that controls the network dynamics. The Boolean model is defined as a tuple $\mathcal{B} = \langle G, \sigma, \theta, \rho, L\rangle$ where

- $G = (V, E)$ is a directed graph with vertices $V = \{g_1, ..., g_n\}$ denoting *genes* and set of edges $E \subseteq V \times V$ denoting *regulations.*
- $\sigma(e) \in \{+, -\}$ denotes the type of regulation $e \in E$: positive $(+)$ or negative $(-)$,
- $\theta(e) \in \mathbb{N}_{\geq 1}$ denotes the activation *threshold* of $e \in E$,
- $\rho(g_i) \in \mathbb{N}_{\geq 1}$ denotes the *maximum expression level* of $g_i \in V$ determining the expression domain $\{0, ..., \rho(g_i)\}$,
- L is the *regulatory logic* defined as the set $L = \{K_{i,R} \mid 1 \leq i \leq n, R \subseteq \{v \in V \mid \langle v, g_i\rangle \in E\}\}$ where $K_{i,R}$ denotes the *target expression level* of g_i when regulated by all genes in R, $0 \leq K_{i,R} \leq \rho(g_i)$.

In our example, the model, depicted in Fig. 8, consists of two genes pRB and $E2F1$. To differentiate between the two outgoing regulations from both E2F1 and pRB, we choose the genes maximal activity levels $\rho(\text{pRB}) = \rho(\text{E2F1}) = 2$. Negative and positive interactions together with thresholds are set with respect to data presented in [157]. The regulatory logic is known only for the basal gene activity, in particular, *pRB* under the empty context (no incoming regulation active) has the tendency to attain the expression level 1. A significant role for the model behavior has the positive autoregulation of E2F1. In order to become active (i.e., the resource for E2F1, since E2F1 is not a target of any other positive regulation), we need to set $K_{\text{E2F1},\emptyset} = 2$. We do not know target levels for other regulatory contexts of both genes, therefore we consider them as parameters. Note that since the expression levels of both genes is bounded by the maximal activity levels, the number of possible parametrizations is finite.

Once the regulatory logic is set (all parameters are assigned), the semantics in terms of a finite automaton capturing the dynamics of a network $\mathcal{B}$ can be defined as the tuple $BTS(\mathcal{B}) = \langle S, T, S_0\rangle$ where $S = \prod_{i=1}^{n}\{0, ..., \rho(g_i)\}$ is set of states with $S_0 \subseteq S$ initial states and $T \subseteq S \times S$ is the transition relation defined as follows.

First we denote the level of g_i in the state $s \in S$ by $l_i(s)$. Assume there is a regulation $e = \langle g_i, g_j \rangle \in E$ between genes g_i, g_j. We say that g_i is a *resource* for g_j in s if $\sigma(e) = +$ and $l_i \geq \theta(e)$, or $\sigma(e) = -$ and $l_i < \theta(e)$. Let $Re(s, g_i)$ denote the set of all resources of g_i in s. There is a transition $s \to s'$ according to the following rules:

- If there exists u such that $K_{u,Re(s,g_u)} > l_u$ then $l_u(s') = l_u(s) + 1$.
- If there exists u such that $K_{u,Re(s,g_u)} < l_u$ then $l_u(s') = l_u(s) - 1$.

The presented semantics requires that the level of at most one gene can be affected in a single transition. This represents the so-called *asynchronous semantics* [159] that models all possible time-orderings of individual expression level updates by the means of non-determinism (an update on a single gene is considered an atomic operation).

The formulae specifying behavior of E2F1 are built over the following atomic propositions:

$$AP = \{\text{E2F1} < x \mid 1 \leq x \leq \rho(\text{E2F1})\}$$

For the purpose of our analysis, we establish the set of initial states S_0 as those satisfying $l_{\text{pRB}} = 0$ and $l_{\text{E2F1}} \in \{0, 1, 2\}$.

To filter out certain trivial executions, the concept of explicitly stated accepting states in the Büchi automaton (BA) representing the LTL property is used. Such a restriction is called a *fairness constraint* as is frequently used in formal verification by model checking [59]. From the above regulatory logic settings follows the selection of accepting states with $l_{\text{pRB}} \geq 1$, denoted F_1. A more restricting filter may be created by choosing states satisfying $l_{\text{pRB}} = 2$, denoted F_2. The former fairness constraint can be formulated in LTL as a formula $\mathbf{GF}(\text{pRB} \geq 1)$, the latter one as $\mathbf{GF}(\text{pRB} = 2)$.

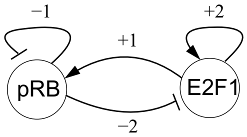
We understand bistability as the following set of properties (described in the form of LTL formulae). These properties are used to detect certain paths in the model state space with respect to the behavior observed *in vitro*. In the following we assume $\theta \in \{1, 2\}$:

- expression of E2F1 begins below the threshold and does not exceed it ($\mathbf{G}(\text{E2F1} < \theta)$)
- expression of E2F1 begins above the threshold and remains such ($\mathbf{G}(\text{E2F1} \geq \theta)$)
- expression of E2F1 begins under the threshold and at certain moment exceeds it and stays above the exceeded level ($(\text{E2F1} < \theta) \to \mathbf{FG}(\text{E2F1} \geq \theta)$)

We used the colored model checking algorithm implemented in the Parsybone tool of the BioDiVinE toolset (see Section 3.5) to identify admissible parametrizations for these properties with the setting of accepting states F_1. In particular, the properties listed above have been used as an observer (BA) that makes a witness for the respective dynamic phenomenon. Only a small parameter restriction was synthesized by employing the observer (BA) for both settings

$\theta = 1$ and $\theta = 2$: $K_{\mathrm{pRB},\{\mathrm{E2F1}\}} \neq 0$. When employing the accepting states F_2, the observer demanded $K_{\mathrm{pRB},\{\mathrm{pRB},\mathrm{E2F1}\}} = 2$ as well and, additionally, $K_{\mathrm{pRB},\{\mathrm{pRB}\}} = 2$ for $\theta = 1$. Apparently, the regulatory network is considerably robust regarding the choice of θ and the setting of regulatory logic on E2F1 (i.e. $K_{\mathrm{E2F1},R}$ where $R \subseteq \{\mathrm{E2F1}, \mathrm{pRB}\}$).

Continuous-Time Deterministic Model. Now we consider a quantitative model of the gene regulatory network traditionally formalized by means of so-called Hill kinetics that abstracts from unknown elementary reactions occurring during processes of protein identification and its regulation. However, from the perspective of computer analysis, Hill kinetics introduces rational polynomial functions into the right-hand sides if ODEs. Course of the Hill function modeling positive regulation is shown in Fig. 9.



$$\begin{array}{ll} K_{pRB}, \emptyset = 1 & K_{E2F1}, \emptyset = 2 \\ K_{pRB,\{pRB\}} = ? & K_{E2F1,\{E2F1\}} = ? \\ K_{pRB,\{E2F1\}} = ? & K_{E2F1,\{pRB\}} = ? \\ K_{pRB,\{E2F1,pRB\}} = ? & K_{E2F1,\{E2F1,pRB\}} = ? \end{array}$$

$$\frac{d[pRB]}{dt} = k_1 \frac{[E2F1]}{0.5+[E2F1]} \frac{0.5}{0.5+[pRB]} - \gamma_{pRB}[pRB]$$

$$\frac{d[E2F1]}{dt} = k_p + k_2 \frac{a^2+[E2F1]^2}{16+[E2F1]^2} \frac{5}{5+[pRB]} - \gamma_{E2F1}[E2F1]$$

Fig. 8. (left) Genetic regulatory network controlling the G_1/S transition. (right) Regulatory logic employed for the qualitative model. (bottom) The original ODE model system that makes the quantitative model of the network.

To analyze the model at the level of quantitative kinetics, we again need to simplify the continuous-time deterministic model. Similarly as in the previous case study, we translate the model into the discrete-time discrete-value domain. To this end, we employ the piece-wise multi-affine abstraction (PMA) of the non-linear ODE model shown in Fig. 8. This abstraction has two consecutive steps:

- transforming the non-linear ODE model into a piece-wise multi-affine (PMA) model and
- performing rectangular abstraction to transform the PMA model into a finite automaton (the so-called rectangular abstraction transition system)

The first step has been defined in [25], the main idea is to get rid of the rational polynomial functions appearing in right-hand sides of ODEs. As illustrated in Fig. 9, this is done by approximating each of them by a so-called ramp function defined in the following way:

$$r^+(x_i, \theta_i, \theta_i') = \begin{cases} 0, & \text{if } x_i \le \theta_i, \\ \frac{x_i - \theta_i}{\theta_i' - \theta_i}, & \text{if } \theta_i < x_i < \theta_i', \\ 1, & \text{if } x_i \ge \theta_i'. \end{cases}$$

The ramp function approximates the non-linear sigmoid function by a piece-wise affine curve. A sum of scaled ramp functions approximating individual segments of the original non-linear curve can be employed.

Second, the PMA model is abstracted by using the rectangular abstraction as introduced in Section 4.1. Since Theorem 1 extends to piece-wise multi-affine models with no restrictions [25], the abstraction procedure is the same as in the case of multi-affine systems. Again, the rectangular abstraction partitions the domain of every unknown parameter domain into a finite number of intervals.

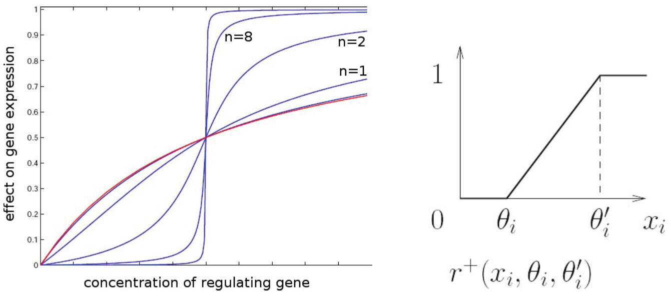


Fig. 9. Sigmoid (Hill) function for positive regulation abstracted by a corresponding ramp function. The steepness is affected by the exponent appearing in Hill functions, here denoted n.

The parameters in the original ODE model have been estimated by employing the bifurcation analysis [157]. We show how our alternative method based on model checking can be employed for identification of parametrizations satisfying the required specification. Our PMA abstraction of this system is shown in Fig. 10. Each function $\varrho_i(x)$ is defined as a sum of several ramp-functions that gradually approximate the respective regulatory Hill curve by a polyline.

Since we detected bistability by using the qualitative model above, it follows to find how this phenomenon is affected by the setting of (quantitative) kinetic parameters. As shown in [157], the steady behavior of this system is strongly influenced by the degradation coefficient γ_{pRB}. Fig. 11 shows the vector field of the above system for two different values of γ_{pRB}.

Similarly to the previous case, to express dynamical properties of paths in rectangular abstraction of an n-dimensional model $\mathcal{M}$ we employ traditional Linear Temporal Logic (LTL) built over atomic propositions AP:

$$AP = \left\{ x_i \odot \theta_j^i \mid 1 \le i \le n, 1 \le j \le \zeta_i \}, \odot \in \{<, >\} \right\}.$$

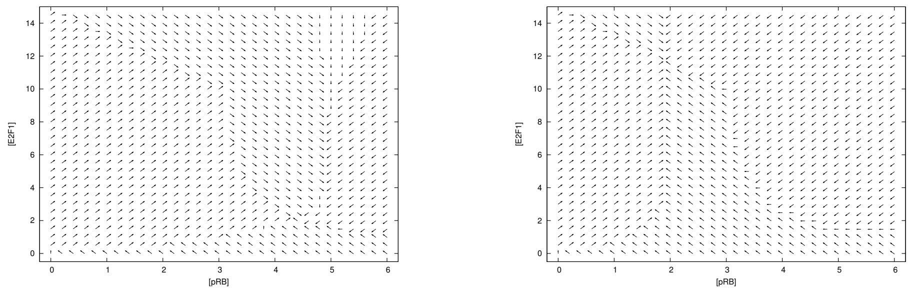
$$\begin{aligned}\frac{d[pRB]}{dt} &= k_1\varrho_1(pRB, E2F1) - \gamma_{pRB}[pRB]\\ \frac{d[E2F1]}{dt} &= k_p + k_2\varrho_2(pRB, E2F1) - \gamma_{E2F1}[E2F1]\end{aligned}$$

$$\begin{aligned}\varrho_1(pRB, E2F1) \;=\;& (0.85r^+(E2F1, 0, 3) + 0.1r^+(E2F1, 3, 10) + 0.05r^+(E2F1, 10, 50))\\ &\cdot(0.85r^-(pRB, 0, 2) + 0.1r^-(pRB, 2, 5) + 0.05r^-(pRB, 5, 80))\\ \varrho_2(pRB, E2F1) \;=\;& (0.85r^+(E2F1, 0, 10) + 0.1r^+(E2F1, 10, 20) + 0.05r^+(E2F1, 20, 80))\\ &\cdot(0.5r^-(pRB, 0, 5) + 0.2r^-(pRB, 5, 10) + 0.15r^-(pRB, 10, 30) + 0.15r^-(pRB, 30, 130))\end{aligned}$$

Fig. 10. Piece-wise multi-affine abstraction (PMA model) for the G_1/S transition regulatory network

Our goal is to determine the set of parameters in the range $[0.01, 1]$ for which the concentration of $E2F1$ is greater than 8 in the stable state. This phenomenon can be specified in terms of an LTL formula $\varphi = \mathbf{FG}([E2F1] > 8)$.

We executed the colored model checking algorithm implemented in the PEPMC tool (see Section 3.5) for the model described above and the property φ. Since the abstraction technique has the overapproximative character, some of counterexamples found by model checking can be false-positive paths. Therefore the result is an under-approximated set of parameter valuations under which the property φ is satisfied. Owing to the fact that the property φ is a liveness property, many of the counterexamples can be paths on which the time does not really proceed (the so-called time-convergent paths). In [21] we have shown a way of how the model checking procedure for this specific model can be elaborated to avoid unwanted time-convergent paths. By applying the algorithm to the model described above we were able to prove that for $\gamma_{pRB} > 0.053$, the system stabilizes with $E2F1 > 8$.



(a) $\gamma_{pRB} = 0.01$, the system stabilizes with $E2F1 < 3$

(b) $\gamma_{pRB} = 0.1$, the system stabilizes with $E2F1 > 11$

Fig. 11. Vector field of the liveness model

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

(a)

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

(b)

Fig. 12. (a) Stochastic mass action model of the G_1/S regulatory circuit – A denotes the protein pRB, B denotes E2F1, a, b represent genes, aA, aB, bA, bB represent transcription factor-gene promoter complexes (b) Computation results

Continuous-Time Stochastic Model. We have translated the original ODE model into the framework of stochastic mass action kinetics [94]. The resulting reactions are shown in Fig. 12a. Since the detailed knowledge of elementary chemical reactions occurring in the process of transcription and translation is incomplete, we use the simplified form as suggested in [80]. In the minimalistic setting, the reformulation requires addition of rate parameters describing the transcription factor–gene promoter interaction while neglecting cooperativeness of transcription factors activity. Our parametrization is based on time-scale orders known for the individual processes [162]. Moreover, we assume the numbers of A and B are bounded by 10 molecules. The bound is calibrated with respect to the original ODE model and reflects the character of the two steady states. All other species are bounded by the initial number of DNA molecules (genes a and b) which is conserved and set to 1. The model is translated into a CTMC which has 1078 states and 5919 transitions.

An interesting biologically relevant problem is to predict how the population of cells implements this regulatory circuit in reaction to mitogenic stimulation and under presence of noise. Low molecular numbers typical for DNA and proteins molecules make the gene regulation highly sensitive to noise. Since mitogenic stimulation influences the degradation rate of A, our goal is to study the population distribution around the low and high steady state.

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

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

For the fixed valuations of parameters, quantitative CSL model checking can be used to answer the above mentioned questions. For this purpose, PRISM provides the most suitable tool [125]. The papers on applying PRISM to biological models [102,127], the book [112], and the tutorials available at the tool webpage provide a good source for this topic.

Here we focus on analysing the above stated properties with respect to the potential uncertainty in parameters. In particular, we explore the effect of the parameter γ_A on the probability of the properties. According to [157], we consider the parameter space $\gamma_A \in [0.005, 0.5]$.

The technique employed for the parameter exploration is described in [40], it is implemented on the top of PRISM. The basic notion is the *landscape function* that for each parameter point from the inspected parameter space returns the quantitative model checking result for the respective CTMC determined by the parameter point and the given property. Computation of the landscape function is based on automatic decomposition of the given parameter space with respect to how it influences the model dynamics. The computation is approximative and provides the result within a required absolute error bound.

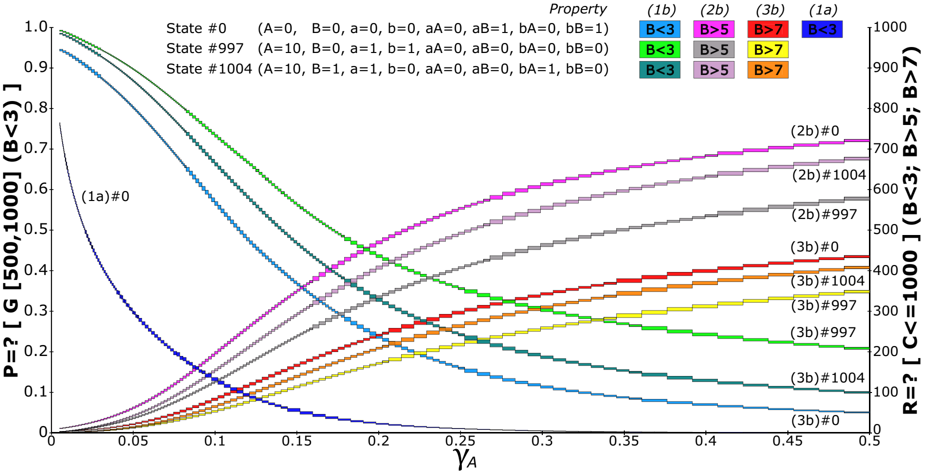


Fig. 13. Landscape functions of properties *(1a,1b,2b,3b)* for parameter $\gamma_A \in [0.005, 0.5]$ and initial states #0, #997 and #1004. The left Y-axis scale corresponds to *(1a)*, the right to *(1b,2b,3b)*.

Since the stochastic noise causes molecules to repeatedly escape the requested bound, the resulting probability is significantly lower than expected. Namely, in cases (2a) and (3a) the resulting probability is close to 0 for the whole parameter space. Moreover, the selection of an initial state has only a negligible impact on the result. Therefore, in Fig. 13 only the resulting probability for case (1a) and a single selected initial state is visualized.

Second, we use a cumulative reward property [126] to capture the fraction of the time the system has the required number of molecules within the time interval

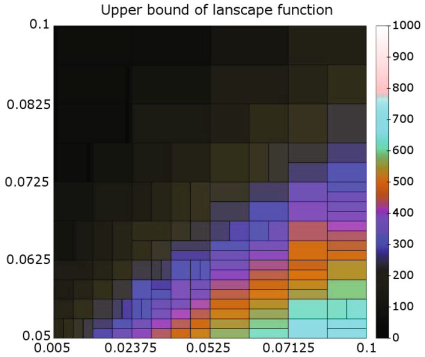


Fig. 14. Landscape function for property (3*b*), initial state #0 ($A = 0, B = 0, a = 0, b = 0, aA = 0, aB = 1, bA = 0, bB = 1$) and two-dimensional parameter space $(\gamma_A, \gamma_B) \in [0.005, 0.1] \times [0.05, 0.1]$ (represented by X and Y axes, respectively). The upper bound of the landscape function is illustrated.

$[0, 1000]$: (1*b*) $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B < 3)$, (2*b*) $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B > 5)$, (3*b*) $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B > 7)$ where $t = 1000$ and $\mathsf{R}_{\sim ?}[\mathsf{C}^{\leq t}](B \sim X)$ denotes that state reward ρ is defined such that $\forall s \in \mathbb{S}.\rho(s) = 1$ iff $B \sim X$ in s. The result is visualized for three selected initial states in Fig. 13.

Fig. 13 also illustrates inaccuracy of our approach with respect to the absolute error bound $\textsc{Err} = 0.01$ by means of small rectangles depicting approximations of the resulting probabilities and expected rewards. The analyses predict that the distribution of the low steady mode interferes with the distribution of the high steady mode. It confirms bistability predicted in [157] but in contrast to ODE analysis our method shows how the population of cells distributes around the two stable states. Results of computations including the number of iterations performed during parametrized uniformization, numbers of resulting subspaces and execution times in hours, are presented in Fig. 12b.

Finally, to see how degradation rates of A and B cooperate in affecting property (3*b*), we explore two-dimensional parameter space $(\gamma_A, \gamma_B) \in [0.005, 0.1] \times [0.05, 0.1]$. Fig. 14 illustrates the computed upper bound of the landscape function for initial state #0. The result predicts antagonistic relation between the degradation rates which is in agreement with the ODE model [157].

References

1. Allmaier, S., Dalibor, S., Kreische, D.: Parallel Graph Generation Algorithms for Shared and Distributed Memory Machines. In: Parallel Computing Conference (PARCO). LNCS, vol. 1253, pp. 207–218. Springer (1997)
2. Alur, R., Courcoubetis, C., Dill, D.: Model-checking in dense real-time. Information and Computation 104, 2–34 (1993)
3. Alur, R., Feder, T., Henzinger, T.A.: The benefits of relaxing punctuality. J. ACM 43(1), 116–146 (1996)

4. Alur, R., Henzinger, T.A.: A really temporal logic. J. ACM 41(1), 181–203 (1994)
5. Antoniotti, M., Policriti, A., Ugel, N., Mishra, B.: Model building and model checking for biochemical processes. Cell Biochemistry and Biophysics 38, 271–286 (2003)
6. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
7. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model Checking Continuous-Time Markov Chains by Transient Analysis. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 358–372. Springer, Heidelberg (2000)
8. Ballarini, P., Forlin, M., Mazza, T., Prandi, D.: Efficient parallel statistical model checking of biochemical networks. In: Parallel and Distributed Methods in verifiCation (PDMC). EPTCS, vol. 14, pp. 47–61 (2009)
9. Ballarini, P., Guerriero, M.L.: Query-based verification of qualitative trends and oscillations in biochemical systems. Theor. Comput. Sci. 411(20), 2019–2036 (2010)
10. Ballarini, P., Guido, R., Mazza, T., Prandi, D.: Taming the complexity of biological pathways through parallel computing. Briefings in Bioinformatics 10(3), 278–288 (2009)
11. Barbuti, R., Caravagna, G., Maggiolo-Schettini, A., Milazzo, P., Tini, S.: Foundational aspects of multiscale modeling of biological systems with process algebras. Theor. Comput. Sci. 431, 96–116 (2012)
12. Barnat, J., Bauch, P., Brim, L., Češka, M.: Computing Strongly Connected Components in Parallel on CUDA. In: International Parallel & Distributed Processing Symposium (IPDPS), pp. 541–552. IEEE Computer Society (2011)
13. Barnat, J., Brim, L., Krejci, A., Streck, A., Safranek, D., Vejnar, M., Vejpustek, T.: On Parameter Synthesis by Parallel Model Checking. IEEE/ACM Transactions on Computational Biology and Bioinformatics 9(3), 693–705 (2012)
14. Barnat, J., Brim, L., Ročkai, P.: Scalable Multi-core LTL Model-Checking. In: Bošnački, D., Edelkamp, S. (eds.) SPIN 2007. LNCS, vol. 4595, pp. 187–203. Springer, Heidelberg (2007)
15. Barnat, J., Brim, L., Ročkai, P.: A Time-Optimal On-the-Fly Parallel Algorithm for Model Checking of Weak LTL Properties. In: Breitman, K., Cavalcanti, A. (eds.) ICFEM 2009. LNCS, vol. 5885, pp. 407–425. Springer, Heidelberg (2009)
16. Barnat, J., Brim, L., Ročkai, P.: Parallel Partial Order Reduction with Topological Sort Proviso. In: Software Engineering and Formal Methods (SEFM), pp. 222–231. IEEE Computer Society (2010)
17. Barnat, J., Brim, L., Stříbrná, J.: Distributed LTL Model-Checking in SPIN. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 200–216. Springer, Heidelberg (2001)
18. Barnat, J., Brim, L., Černá, I., Dražan, S., Fabriková, J., Láník, J., Šafránek, D., Ma, H.: BioDiVinE: A Framework for Parallel Analysis of Biological Models. In: Computational Models for Cell Processes (COMPMOD). EPTCS, vol. 6, pp. 31–45 (2009)
19. Barnat, J., Brim, L., Černá, I., Moravec, P., Ročkai, P., Šimeček, P.: DiVinE – A Tool for Distributed Verification. In: Ball, T., Jones, R.B. (eds.) CAV 2006. LNCS, vol. 4144, pp. 278–281. Springer, Heidelberg (2006)
20. Barnat, J., Brim, L., Šafránek, D.: High-Performance Analysis of Biological Systems Dynamics with the DiVinE Model Checker. Briefings in Bioinformatics 11(3), 301–312 (2010)

21. Barnat, J., Brim, L., Šafránek, D., Vejnár, M.: Parameter Scanning by Parallel Model Checking with Applications in Systems Biology. In: Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology (HiBi/PDMC 2010), pp. 95–104. IEEE Computer Society (2010)
22. Barnat, J., Ročkai, P.: Shared Hash Tables in Parallel Model Checking. In: Parallel and Distributed Methods in verifiCation (PDMC). ENTCS, vol. 198, pp. 79–91 (2008)
23. Barnat, J., Bauch, P., Brim, L., Češka, M.: Designing fast LTL model checking algorithms for many-core GPUs. Journal of Parallel and Distributed Computing 72(9), 1083–1097 (2012)
24. Bartocci, E., Corradini, F., Merelli, E., Tesei, L.: Detecting synchronisation of biological oscillators by model checking. Theoretical Computer Science 411(20), 1999–2018 (2010)
25. Batt, G., Belta, C., Weiss, R.: Model checking genetic regulatory networks with parameter uncertainty. In: Bemporad, A., Bicchi, A., Buttazzo, G. (eds.) HSCC 2007. LNCS, vol. 4416, pp. 61–75. Springer, Heidelberg (2007)
26. Batt, G., Page, M., Cantone, I., Goessler, G., Monteiro, P., de Jong, H.: Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. Bioinformatics 26(18), 603–610 (2010)
27. Batt, G., Ben Salah, R., Maler, O.: On timed models of gene networks. In: Raskin, J.-F., Thiagarajan, P.S. (eds.) FORMATS 2007. LNCS, vol. 4763, pp. 38–52. Springer, Heidelberg (2007)
28. Behrmann, G., Hune, T., Vaandrager, F.: Distributed Timed Model Checking — How the Search Order Matters. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 216–231. Springer, Heidelberg (2000)
29. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (eds.) SFM-RT 2004. LNCS, vol. 3185, pp. 200–236. Springer, Heidelberg (2004)
30. Belta, C., Habets, L.: Controlling a class of nonlinear systems on rectangles. IEEE Transactions on Automatic Control 51(11), 1749–1759 (2006)
31. Bernot, G., Comet, J.P., Richard, A., Guespin, J.: Application of formal methods to biological regulatory networks: extending thomas' asynchronous logical approach with temporal logic. Journal of Theoretical Biology 229(3), 339–347 (2004)
32. Bonzanni, N., Krepska, E., Feenstra, K.A., Fokkink, W., Kielmann, T., Bal, H.E., Heringa, J.: Executing multicellular differentiation: quantitative predictive modelling of *C.elegans* vulval development. Bioinformatics 25(16), 2049–2056 (2009)
33. Bortolussi, L., Hillston, J.: Fluid model checking. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 333–347. Springer, Heidelberg (2012)
34. Bortolussi, L., Policriti, A.: Hybrid systems and biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 424–448. Springer, Heidelberg (2008)
35. Bošnački, D., Edelkamp, S., Sulewski, D.: Efficient Probabilistic Model Checking on General Purpose Graphics Processors. In: Păsăreanu, C.S. (ed.) SPIN 2009. LNCS, vol. 5578, pp. 32–49. Springer, Heidelberg (2009)
36. Bošnački, D., ten Eikelder, H.M.M., Steijaert, M.N., de Vink, E.P.: Stochastic analysis of amino acid substitution in protein synthesis. In: Heiner, M., Uhrmacher, A.M. (eds.) CMSB 2008. LNCS (LNBI), vol. 5307, pp. 367–386. Springer, Heidelberg (2008)
37. Brenan, K.E., Campbell, S.L., Petzold, L.R.: Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations. SIAM (1987)

38. Brim, L., Černá, I., Moravec, P., Šimša, J.: Accepting predecessors are better than back edges in distributed LTL model-checking. In: Hu, A.J., Martin, A.K. (eds.) FMCAD 2004. LNCS, vol. 3312, pp. 352–366. Springer, Heidelberg (2004)
39. Brim, L., Barnat, J.: Platform Dependent Verification: On Engineering Verification Tools for 21st Century. In: Parallel and Distributed Methods in verifiCation (PDMC). EPTCS, vol. 72, pp. 1–12 (2011)
40. Brim, L., Česka, M., Dražan, S., Šafránek, D.: Exploring parameter space of stochastic biochemical systems using quantitative model checking. Tech. rep., Faculty of Informatics, Masaryk University (2013), `http://sybila.fi.muni.cz/TR-01-2013.pdf`
41. Burch, J.R., Clarke, E.M., McMillan, K.L., Dill, D.L., Hwang, L.J.: Symbolic model checking: 10^{20} states and beyond. Information and Computation 98(2), 142–170 (1992)
42. Calder, M., Vyshemirsky, V., Gilbert, D., Orton, R.: Analysis of signalling pathways using continuous time markov chains. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 44–67. Springer, Heidelberg (2006)
43. Calzone, L., Chabrier-Rivier, N., Fages, F., Soliman, S.: Machine learning biochemical networks from temporal logic properties. In: Priami, C., Plotkin, G. (eds.) Transactions on Computational Systems Biology VI. LNCS (LNBI), vol. 4220, pp. 68–94. Springer, Heidelberg (2006)
44. Campagna, D., Piazza, C.: Hybrid automata in systems biology: How far can we go? In: From Biology to Concurrency and Back (FBTC). ENTCS, vol. 229, pp. 93–108 (2009)
45. Caravagna, G., Hillston, J.: Modeling biological systems with delays in Bio-PEPA. In: Proceedings Fourth Workshop on Membrane Computing and Biologically Inspired Process Calculi 2010. EPTCS, vol. 40, pp. 85–101 (2010)
46. Carrillo, M., Góngora, P.A., Rosenblueth, D.A.: An overview of existing modeling tools making use of model checking in the analysis of biochemical networks. Front Plant Sci. 3(155), 1–13 (2012)
47. Caselli, S., Conte, G., Marenzoni, P.: Parallel state space exploration for GSPN models. In: DeMichelis, G., Díaz, M. (eds.) ICATPN 1995. LNCS, vol. 935, pp. 181–200. Springer, Heidelberg (1995)
48. Černá, I., Pelánek, R.: Distributed explicit fair cycle detection (Set based approach). In: Ball, T., Rajamani, S.K. (eds.) SPIN 2003. LNCS, vol. 2648, pp. 49–73. Springer, Heidelberg (2003)
49. Chaouiya, C.: Petri net modelling of biological networks. Briefings in Bioinformatics 8(4), 210–219 (2007)
50. Chaouiya, C., Remy, E., Mossé, B., Thieffry, D.: Qualitative analysis of regulatory graphs: A computational tool based on a discrete formal framework. In: Benvenuti, L., De Santis, A., Farina, L. (eds.) Positive Systems. LNCIS, vol. 294, pp. 830–832. Springer, Heidelberg (2003)
51. Che, S., Li, J., Sheaffer, J., Skadron, K., Lach, J.: Accelerating Compute-Intensive Applications with GPUs and FPGAs. In: IEEE Symposium on Application Specific Processors (SASP), pp. 101–107. IEEE Computer Society (2008)
52. Ciardo, G., Gluckman, J., Nicol, D.: Distributed state-space generation of discrete-state stochastic models. INFORMS J. Comp. 10(1), 82–93 (1998)
53. Ciardo, G.: Automated parallelization of discrete state-space generation. J. Parallel Distrib. Comput. 47, 153–167 (1997)
54. Cimatti, A., Clarke, E., Giunchiglia, F., Roveri, M.: NuSMV: a new symbolic model checker. J. Softw. Tools Technol. Transf. 2, 410–425 (2000)

55. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: An OpenSource Tool for Symbolic Model Checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
56. Ciocchetta, F., Hillston, J.: Bio-PEPA: A framework for the modelling and analysis of biological systems. Theor. Comput. Sci. 410(33-34), 3065–3084 (2009)
57. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. 8(2), 244–263 (1986)
58. Clarke, E.M., Enders, R., Filkorn, T., Jha, S.: Exploiting symmetry in temporal logic model checking. Form. Methods Syst. Des. 9(1-2), 77–104 (1996)
59. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press (1999)
60. Clarke, E.M., Zuliani, P.: Statistical Model Checking for Cyber-Physical Systems. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA 2011. LNCS, vol. 6996, pp. 1–12. Springer, Heidelberg (2011)
61. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Progress on the State Explosion Problem in Model Checking. In: Wilhelm, R. (ed.) Informatics: 10 Years Back, 10 Years Ahead. LNCS, vol. 2000, pp. 176–194. Springer, Heidelberg (2001)
62. Collins, P., Habets, L.C., van Schuppen, J.H., Černá, I., Fabriková, J., Šafránek, D.: Abstraction of biochemical reaction systems on polytopes. In: Proceedings of the 18th IFAC World Congress, vol. 18, pp. 14869–14875 (2011)
63. Courcoubetis, C., Vardi, M., Wolper, P., Yannakakis, M.: Memory-Efficient Algorithms for the Verification of Temporal Properties. Formal Methods in System Design 1, 275–288 (1992)
64. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM 42(4), 857–907 (1995)
65. Crudu, A., Debussche, A., Radulescu, O.: Hybrid stochastic simplifications for multiscale gene networks. BMC Systems Biology 3(1), 89 (2009)
66. NVIDIA CUDA Compute Unified Device Architecture - Programming Guide Version 2.0, (2009), http://www.nvidia.com/object/cuda_develop.html
67. Dang, T., Guernic, C.L., Maler, O.: Computing reachable states for nonlinear biological models. Theor. Comput. Sci. 412(21), 2095–2107 (2011)
68. Danos, V., Laneve, C.: Formal molecular biology. Theor. Comput. Sci. 325(1), 69–110 (2004)
69. Darling, R., Norris, J.: Differential equation approximations for markov chains. Probab. Surveys 5, 37–79 (2008)
70. David, A., Du, D., Larsen, K.G., Legay, A., Mikucionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. In: Hybrid Systems and Biology (HSB). EPTCS, vol. 92, pp. 122–136 (2012)
71. Derman, C.: Finite State Markovian Decision Processes. Academic Press, Inc., Orlando (1970)
72. Didier, F., Henzinger, T.A., Mateescu, M., Wolf, V.: Fast Adaptive Uniformization for the Chemical Master Equation. In: Parallel and Distributed Methods in Verification and High Performance Computational Systems Biology (HiBi/PDMC 2009), pp. 118–127. IEEE Computer Society (2009)
73. Didier, F., Henzinger, T.A., Mateescu, M., Wolf, V.: Sabre: A tool for stochastic analysis of biochemical reaction networks. CoRR abs/1005.2819 (2010)
74. Dluhoš, P., Brim, L., Šafránek, D.: On expressing and monitoring oscillatory dynamics. In: Hybrid Systems and Biology (HSB). EPTCS, vol. 92, pp. 73–87 (2012)

75. Doi, A., Fujita, S., Matsuno, H., Nagasaki, M., Miyano, S.: Constructing Biological Pathway Models with Hybrid Functional Petri Nets. In Silico Biology 4(3), 271–291 (2004)
76. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010)
77. Donzé, A., Clermont, G., Langmead, C.J.: Parameter synthesis in nonlinear dynamical systems: Application to systems biology. Journal of Computational Biology 17(3), 325–336 (2010)
78. Edelkamp, S., Sulewski, D.: Parallel State Space Search on the GPU (2009), symposium on Combinatorial Search (SoCS)
79. Eker, S., Knapp, M., Laderoute, K., Lincoln, P., Meseguer, J., Sonmez, K.: Pathway logic: Symbolic analysis of biological signaling. In: Pacific Symposium on Biocomputing, pp. 400–412 (2002)
80. El Samad, H., Khammash, M., Petzold, L., Gillespie, D.: Stochastic Modelling of Gene Regulatory Networks. Int. J. of Robust and Nonlinear Control 15(15), 691–711 (2005)
81. Emerson, E.A., Sistla, A.P.: Symmetry and model checking. Form. Methods Syst. Des. 9(1-2), 105–131 (1996)
82. Engl, H.W., Flamm, C., Kügler, P., Lu, J., Müller, S., Schuster, P.: Inverse problems in systems biology. Inverse Problems 25(12), 123014 (2009)
83. Ezekiel, J., Lüttgen, G., Ciardo, G.: Parallelising symbolic state-space generators. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 268–280. Springer, Heidelberg (2007)
84. Fages, F., Soliman, S.: Formal cell biology in Biocham. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 54–80. Springer, Heidelberg (2008)
85. Fages, F., Soliman, S., Rivier, C.N.: Modelling and querying interaction networks in the biochemical abstract machine BIOCHAM. Journal of Biological Physics and Chemistry 4(2), 64–73 (2004)
86. Fages, F., Rizk, A.: On temporal logic constraint solving for analyzing numerical data time series. Theor. Comput. Sci. 408(1), 55–65 (2008)
87. Fisher, J., Henzinger, T.A.: Executable cell biology. Nature Biotechnology 25(11), 1239–1249 (2007)
88. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011)
89. Fromentin, J., Eveillard, D., Roux, O.: Hybrid modeling of biological networks: mixing temporal and qualitative biological properties. BMC Systems Biology 4(1), 79 (2010)
90. Galpin, V., Hillston, J., Bortolussi, L.: HYPE Applied to the Modelling of Hybrid Biological Systems. ENTCS 218, 33–51 (2008)
91. Garavel, H., Mateescu, R., Lang, F., Serwe, W.: CADP 2006: A toolbox for the construction and analysis of distributed processes. In: Damm, W., Hermanns, H. (eds.) CAV 2007. LNCS, vol. 4590, pp. 158–163. Springer, Heidelberg (2007)
92. Garavel, H., Mateescu, R., Smarandache, I.: Parallel State Space Construction for Model-Checking. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 217–234. Springer, Heidelberg (2001)
93. Geldenhuys, J., de Villiers, P.J.A.: Runtime efficient state compaction in SPIN. In: Dams, D.R., Gerth, R., Leue, S., Massink, M. (eds.) SPIN 1999. LNCS, vol. 1680, pp. 12–21. Springer, Heidelberg (1999)

94. Gillespie, D.T.: Exact Stochastic Simulation of Coupled Chemical Reactions. Journal of Physical Chemistry 81(25), 2340–2381 (1977)
95. Gillespie, D.T.: A rigorous derivation of the chemical master equation. Physica A: Statistical Mechanics and its Applications 188(1-3), 404–425 (1992)
96. Gillespie, D.T.: Stochastic Simulation of Chemical Kinetics. Annual Review of Physical Chemistry 58(1), 35–55 (2007)
97. Goethem, S.V., Jacquet, J.M., Brim, L., Šafránek, D.: Timed modelling of gene networks with arbitrary expression level discretization. In: Interactions between Computer Science and Biology. ENTCS. Elsevier (in press, 2013)
98. Grumberg, O., Heyman, T., Schuster, A.: A work-efficient distributed algorithm for reachability analysis. In: Hunt Jr., W.A., Somenzi, F. (eds.) CAV 2003. LNCS, vol. 2725, pp. 54–66. Springer, Heidelberg (2003)
99. Habets, L., van Schuppen, J.H.: A control problem for affine dynamical systems on a full-dimensional polytope. Automatica 40(1), 21–35 (2004)
100. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6, 512–535 (1994)
101. Haverkort, B.R., Bell, A., Bohnenkamp, H.C.: On the efficient sequential and distributed generation of very large Markov chains from stochastic Petri nets. In: Petri Net and Performance Models (PNPM), pp. 12–21. IEEE Computer Society Press (1999)
102. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theoretical Computer Science 319(3), 239–257 (2008)
103. Heiner, M., Gilbert, D., Donaldson, R.: Petri nets for systems and synthetic biology. In: Bernardo, M., Degano, P., Zavattaro, G. (eds.) SFM 2008. LNCS, vol. 5016, pp. 215–264. Springer, Heidelberg (2008)
104. Heljanko, K., Khomenko, V., Koutny, M.: Parallelisation of the Petri Net Unfolding Algorithm. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 371–385. Springer, Heidelberg (2002)
105. Henzinger, T.: The theory of hybrid automata. In: Logic in Computer Science (LICS), pp. 278 –292. IEEE Computer Society (1996)
106. Henzinger, T.A., Kopke, P.W., Puri, A., Varaiya, P.: What's decidable about hybrid automata? In: Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, pp. 373–382. ACM (1995)
107. Holzmann, G.J.: The Spin Model Checker: Primer and Reference Manual. Addison-Wesley (2003)
108. Holzmann, G.J.: A Stack-Slicing Algorithm for Multi-Core Model Checking. Electonic Notes in Theoretical Computer Science 198(1), 3–16 (2008)
109. Holzmann, G.J., Bosnacki, D.: The design of a multicore extension of the spin model checker. IEEE Trans. Software Eng. 33(10), 659–674 (2007)
110. Horn, F., Jackson, R.: General mass action kinetics. Archive for Rational Mechanics and Analysis 47, 81–116 (1972), doi:10.1007/BF00251225
111. Inggs, C.P., Barringer, H.: CTL* Model Checking on a Shared-Memory Architecture. Electronic Notes in Theoretical Computer Science 128(3), 107–123 (2005)
112. Iyengar, M.S.: Symbolic Systems Biology: Theory and Methods. Jones & Bartlett Publishers (2010)
113. Jayachandran, G., Vishal, V., Pande, V.S.: Using massively parallel simulations and Markovian models to study protein folding: examining the villin head-piece. Journal of Chemical Physics 124(6), 903–914 (2006)

114. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
115. de Jong, H.: Modeling and Simulation of Genetic Regulatory Systems: A Literature Review. Journal of Computational Biology 9(1), 67–103 (2002)
116. de Jong, H., Gouzé, J., Hernandez, C., Page, M., Sari, T., Geiselmann, J.: Qualitative simulations of genetic regulatory networks using piecewise linear models. Bull. Math. Biol. 66, 301–340 (2004)
117. Kahn, A.B.: Topological sorting of large networks. Commun. ACM 5(11), 558–562 (1962)
118. Keener, J.P., Sneyd, J.: Mathematical Physiology. Springer (1998)
119. Khademi, S., O'Connell III, J., Remis, J., Robles-Colmenares, Y., Miercke, L., Stroud, R.: Mechanism of ammonia transport by Amt/MEP/Rh: Structure of AmtB at 1.35. Science 305(5690), 1587–1594 (2004)
120. Kholodenko, B.N.: Cell-signalling dynamics in time and space. Nature Molecular Cell Biology 7, 165–176 (2006)
121. Klarner, H., Streck, A., Šafránek, D., Kolčák, J., Siebert, H.: Parameter identification and model ranking of thomas networks. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, vol. 7605, pp. 207–226. Springer, Heidelberg (2012)
122. Knottenbelt, W., Mestern, M., Harrison, P., Kritzinger, P.: Probability, parallelism and the state space exploration problem. In: Puigjaner, R., Savino, N.N., Serra, B. (eds.) TOOLS 1998. LNCS, vol. 1469, pp. 165–179. Springer, Heidelberg (1998)
123. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems 2, 255–299 (1990)
124. Kumar, R., Mercer, E.G.: Load Balancing Parallel Explicit State Model Checking. In: Parallel and Distributed Methods in Verification (PDMC). ENTCS, vol. 128, pp. 19–34. Elsevier (2005)
125. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
126. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 220–270. Springer, Heidelberg (2007)
127. Kwiatkowska, M.Z., Norman, G., Parker, D.: Using probabilistic model checking in systems biology. SIGMETRICS Performance Evaluation Review 35(4), 14–21 (2008)
128. Laarman, A., van de Pol, J., Weber, M.: Boosting Multi-Core Reachability Performance with Shared Hash Tables. In: Formal Methods in Computer-Aided Design (FMCAD), pp. 247–255. IEEE Computer Science (2010)
129. Lerda, F., Sisto, R.: Distributed-memory Model Checking with SPIN. In: Dams, D.R., Gerth, R., Leue, S., Massink, M. (eds.) SPIN 1999. LNCS, vol. 1680, pp. 22–39. Springer, Heidelberg (1999)
130. Lerda, F., Visser, W.: Addressing Dynamic Issues of Program Model Checking. In: Dwyer, M.B. (ed.) SPIN 2001. LNCS, vol. 2057, pp. 80–102. Springer, Heidelberg (2001)
131. Ma, H., Boogerd, F., Goryanin, I.: Modelling nitrogen assimilation of *Escherichia coli* at low ammonium concentration. Journal of Biotechnology 144(3), 175–183 (2009)

132. Madsen, C., Myers, C., Roehner, N., Winstead, C., Zhang, Z.: Utilizing Stochastic Model Checking to Analyze Genetic Circuits. In: IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp. 379–386 (2012)
133. Maler, O., Batt, G.: Approximating continuous systems by timed automata. In: Fisher, J. (ed.) FMSB 2008. LNCS (LNBI), vol. 5054, pp. 77–89. Springer, Heidelberg (2008)
134. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
135. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. In: Avron, A., Dershowitz, N., Rabinovich, A. (eds.) Pillars of Computer Science. LNCS, vol. 4800, pp. 475–505. Springer, Heidelberg (2008)
136. Mateescu, R., Monteiro, P.T., Dumas, E., de Jong, H.: CTRL: Extension of CTL with regular expressions and fairness operators to verify genetic regulatory networks. Theoretical Computer Science 412(26), 2854–2883 (2011)
137. Melham, T., Bard, J., Werner, E., Noble, D.: Conceptual foundations of systems biology. Prog. Biophys. Mol. Biol. (2012)
138. Merrill, D., Garland, M., Grimshaw, A.: Scalable GPU Graph Traversal. In: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 117–128. ACM (2012)
139. Pelánek, R.: Fighting State Space Explosion: Review and Evaluation. In: Cofer, D., Fantechi, A. (eds.) FMICS 2008. LNCS, vol. 5596, pp. 37–52. Springer, Heidelberg (2009)
140. Peled, D.: Ten years of partial order reduction. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 17–28. Springer, Heidelberg (1998)
141. Phillips, A., Cardelli, L.: Efficient, correct simulation of biological processes in the stochastic π-calculus. In: Calder, M., Gilmore, S. (eds.) CMSB 2007. LNCS (LNBI), vol. 4695, pp. 184–199. Springer, Heidelberg (2007)
142. Pnueli, A.: The temporal semantics of concurrent programs. Theoretical Computer Science 13(1), 45–60 (1981)
143. Popova-Zeugmann, L., Heiner, M., Koch, I.: Time Petri Nets for Modelling and Analysis of Biochemical Networks. Fundam. Inform. 67(1-3), 149–162 (2005)
144. Priami, C.: Algorithmic systems biology. Commun. ACM 52(5), 80–88 (2009)
145. Regev, A., Silverman, W., Shapiro, E.Y.: Representation and Simulation of Biochemical Processes Using the π-Calculus Process Algebra. In: Pacific Symposium on Biocomputing, pp. 459–470 (2001)
146. Reif, J.: Depth-first Search is Inherently Sequential. Information Proccesing Letters 20(5), 229–234 (1985)
147. Rizk, A., Batt, G., Fages, F., Soliman, S.: A general computational method for robustness analysis with applications to synthetic gene networks. Bioinformatics 25(12) (2009)
148. Satish, N., Harris, M., Garland, M.: Designing efficient sorting algorithms for manycore gpus. In: IEEE International Parallel & Distributed Processing Symposium (IPDPS), pp. 1–10. IEEE Computer Society (2009)
149. Schaub, M., Henzinger, T., Fisher, J.: Qualitative networks: a symbolic approach to analyze biological signaling networks. BMC Systems Biology 1(1), 4 (2007)

150. Schivo, D.S., Scholma, J., Wanders, B., Urquidi Camacho, R., van der Vet, P., Karperien, H., Langerak, R., van de Pol, J., Post, J.: Modelling biological pathway dynamics with timed automata. In: IEEE International Conference on Bioinformatics and Bioengineering (ICBB), pp. 447–453. IEEE Computer Society (2012)
151. Schwarick, M., Heiner, M.: CSL model checking of biochemical networks with interval decision diagrams. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 296–312. Springer, Heidelberg (2009)
152. Schwarick, M., Rohr, C., Heiner, M.: MARCIE - Model checking and Reachability analysis done effiCIEntly. In: Quantitative Evaluation of SysTems (QEST 2011), pp. 91–100. IEEE Computer Society (2011)
153. Siebert, H., Bockmayr, A.: Incorporating time delays into the logical analysis of gene regulatory networks. In: Priami, C. (ed.) CMSB 2006. LNCS (LNBI), vol. 4210, pp. 169–183. Springer, Heidelberg (2006)
154. Singh, A., Hespanha, J.P.: Stochastic hybrid systems for studying biochemical processes. Physical and Engineering Sciences 368(1930), 4995–5011 (2010)
155. Stern, U., Dill, D.L.: Parallelizing the murφ verifier. In: Grumberg, O. (ed.) CAV 1997. LNCS, vol. 1254, pp. 256–267. Springer, Heidelberg (1997)
156. Stern, U., Dill, D.L.: Using Magnetic Disk Instead of Main Memory in the Murφ Verifier. In: Vardi, M.Y. (ed.) CAV 1998. LNCS, vol. 1427, pp. 172–183. Springer, Heidelberg (1998)
157. Swat, M., Kel, A., Herzel, H.: Bifurcation analysis of the regulatory modules of the mammalian G1/S transition. Bioinformatics 20(10), 1506–1511 (2004)
158. Tarjan, R.: Depth First Search and Linear Graph Algorithms. SIAM Journal on Computing 1(2), 146–160 (1972)
159. Thomas, R.: Regulatory networks seen as asynchronous automata: A logical description. Journal of Theoretical Biology 153(1), 1–23 (1991)
160. Vardi, M.Y., Wolper, P.: An Automata-Theoretic Approach to Automatic Program Verification. In: IEEE Symposium on Logic in Computer Science (LICS), pp. 332–344. IEEE Computer Society Press (1986)
161. Stewart, W.J.: Introduction to the Numerical Solution of Markov Chains. Princeton University Press (1995)
162. Yang, E., van Nimwegen, E., Zavolan, M., Rajewsky, N., Schroeder, M., Magnasco, M., Darnell, J.E.: Decay Rates of Human mRNAs: Correlation With Functional Characteristics and Sequence Attributes. Genome Research 13(8), 1863–1872 (2003)
163. Yang, H.T., Ko, M.S.H.: Stochastic modeling for the expression of a gene regulated by competing transcription factors. PLoS ONE 7(3), e32376 (2012)
164. Yovine, S.: Kronos: a verification tool for real-time systems. International Journal on Software Tools for Technology Transfer 1, 123–133 (1997)

Checking Individual Agent Behaviours in Markov Population Models by Fluid Approximation

Luca Bortolussi[1] and Jane Hillston[2]

[1] Department of Mathematics and Geosciences
University of Trieste, Italy
CNR/ISTI, Pisa, Italy
luca@dmi.units.it

[2] Laboratory for the Foundations of Computer Science,
School of Informatics, University of Edinburgh, UK
jane.hillston@ed.ac.uk

Abstract. In this chapter, we will describe, in a tutorial style, recent work on the use of fluid approximation techniques in the context of stochastic model checking. We will discuss the theoretical background and the algorithms working out an example.

This approach is designed for population models, in which a (large) number of individual agents interact, which give rise to continuous time Markov chain (CTMC) models with a very large state space. We then focus on properties of individual agents in the system, specified by Continuous Stochastic Logic (CSL) formulae, and use fluid approximation techniques (specifically, the so called fast simulation) to check those properties. We will show that verification of such CSL formulae reduces to the computation of reachability probabilities in a special kind of time-inhomogeneous CTMC with a small state space, in which both the rates and the structure of the CTMC can change (discontinuously) with time. In this tutorial, we will discuss only briefly the theoretical issues behind the approach, like the decidability of the method and the consistency of the approximation scheme.

Keywords: Stochastic model checking, fluid approximation, mean field approximation, reachability probability, time-inhomogeneous Continuous Time Markov Chains.

1 Introduction

In recent years there has been a growing interest in the use of mean field or fluid approximation techniques in the analysis of large scale models of dynamic behaviour. In particular there have been a number of attempts to integrate these mathematical approximations with formal modelling techniques originating in theoretecial computer science. Specifically, fluid approximation techniques are cross-fertilising with quantitative formal methods, mainly Stochastic Process

M. Bernardo et al. (Eds.): SFM 2013, LNCS 7938, pp. 113–149, 2013.

Algebras (SPA), giving birth to a new class of quantitative analysis techniques for large scale population models, described by continuous time Markov chains (CTMC) [7,8,31,27,16,9].

Whereas process algebra models can be viewed as agent-based descriptions in which the behaviour of each entity in the system is described in detail, fluid or mean field methods focus instead on the more abstraction population view of the system. In order to move from the process algebra description to a mean field approximation, individuality of agents must be abstracted and a collective or counting abstraction employed [27]. This leads to an aggregated state representation, in which the system is described by variables counting the number of agents in each possible state. Then, the discrete state space of the CTMC is approximated by a continuous one, and the stochastic dynamics of the process is approximated by a deterministic one, meaning that states no longer evolve through discrete jumps based on the interleaved state changes of individuals. Instead the state variables are assumed to be subject to continuous change expressed by means of a set of differential equations. The correctness of this approach is guaranteed by limit theorems [34,21,22], showing the convergence of the CTMC to the fluid ODE for systems of increasing population size.

On the one hand, fluid approximation has been used in the context of stochastic process algebras mainly to approximately compute the average transient dynamics, or to approximate the average steady state, when possible [49,48]. It has also been used to estimate fluid passage times [25,26].

On the other hand, stochastic process algebra models can also be analysed using quantitative model checking. These techniques have a long tradition in computer science and provide powerful ways of querying a model and extracting information about its behaviour. In the case of stochastic model checking, there are some consolidated approaches, principally based on checking Continuous Stochastic Logic (CSL) formulae [5,4,45], and these are supported by software tools which are in widespread use [36,37]. All these methods, however, suffer (in a more or less relevant way) from the curse of state space explosion, which severely hampers their practical applicability particularly for population models.

One possibility to mitigate the state space explosion problem is to combine fluid approximation techniques with stochastic model checking, obtaining efficient approximate algorithms for checking formulae against population models. In this tutorial, we will present a first attempt in this direction [12,13], in which mean field approximation is used to carry out approximate model checking of behaviours of individual agents in large population models, specified as CSL formulae. This is made possible by a corollary of the fluid convergence theorems, known by the name of *fast simulation* [24,22], which characterises the limit behaviour of a single agent in terms of the solution of the fluid equation: the agent senses the rest of the population only through its “average” evolution, as given by the fluid equation. The idea of [12,13] is to check individual properties in this limit model, rather than on the full model with N interacting agents. In fact, extracting metrics from the description of the global system can be extremely expensive from a computational point of view. Fast simulation, instead, is a very compact abstraction of the system and the evolution of a single agent (or of a subset of agents) can be computed efficiently, by decoupling its evolution from the evolution of its environment and hence reducing the dimensionality of the state space by several orders of magnitude. A central feature of the abstraction based on fluid approximation is that the limit model of an individual agent is a time-inhomogeneous CTMC (ICTMC). This introduces some additional complexity in the approach, as model checking of ICTMC is far more difficult than the homogeneous-time counterpart. Nevertheless we can learn from the previous techniques for model-checking CSL properties on time-homogeneous CTMC, and developed suitable approaches for ICTMCs.

In this tutorial, we will present the work of [12,13] in detail, using a network epidemic model as a simple running example. We will start by setting the context by presenting a simple modelling language for population CTMC (Section 2), discussing CSL and properties for individual agent (Section 3), and introducing some fundamental concepts about fluid approximation and fast simulation (Section 4). We will then turn to explain in detail the model checking procedure for ICTMC, considering first non-nested properties (Section 5), and then turning to nested CSL formulae (Section 7). The difficulty in this case is that the truth of a formula depends on the time at which the formula is evaluated, hence we need algorithms to compute this functional dependence (Section 6). The algorithms to model check nested formulae are presented only informally, by means of the running example (Section 7). We also discuss briefly two theoretical aspects of the work in [12,13], namely decidability of the model checking algorithm for ICTMC and convergence of the truth value of CSL formulae for an individual in a system with a finite population level to the truth for the limit individual model (Section 8). We discuss the related work in Section 9 and finally, we sketch some conclusions in Section 10.

2 Population Models

In this section, we will introduce a simple language to construct Markov models of populations of interacting agents. We will consider models of processes evolving in continuous time, although a similar theory can be considered for discrete-time models (see, for instance, [14]). In principle, we can have different classes of agents, and many agents for each class in the system. Furthermore, the number of agents can change at runtime, due to birth or death events. Models of this kind include computer networks, where each node (e.g. server, client) of the network is an agent [38], biological systems (in which molecules are the agents) [47], ecological systems (in which individual animals are the agents) [11], and so on. However, to keep notation simple, we will assume here that the number of agents is constant and equal to N (making a closed world assumption). Furthermore, in the notation we do not distinguish between different classes of agents.

In particular, let us assume that each agent is a finite state machine, with internal states taken from a finite set S, and labeled by integers: $S = \{1, 2, \ldots, n\}$. We have a population of N agents, and denote the state of agent i at time t, for $i = 1, \ldots, N$, by $Y_i^{(N)}(t) \in S$. Note that we made explicit the dependence on N, the total population size.

A configuration of a system is thus represented by the tuple $(Y_1^{(N)}, \ldots, Y_N^{(N)})$. This represention is based on treating each agent as a distinct individual with identity conferred by the position in the vector. However, when dealing with population models, it is customary to assume that single agents in the same internal state cannot be distinguished, hence we can move from the *individual representation* to the *collective representation* by introducing n variables counting how many agents are in each state. Hence, we define

$$X_j^{(N)} = \sum_{i=1}^{N} \mathbf{1}\{Y_i^{(N)} = j\}. \tag{1}$$

Note that the vector $\mathbf{X}^{(N)} = (X_1^{(N)}, \ldots, X_n^{(N)})$ has a dimension independent of N, and will be equivalently referred to as the collective, population, or counting vector. The domain of each variable $X_j^{(N)}$ is obviously $\{0, \ldots, N\}$, and, by the closed world assumption, it holds that $\sum_{j=1}^{n} X_j^{(N)} = N$. Let us denote with $\mathcal{S}^{(N)}$ the subset of vectors of $\{1, \ldots, N\}^n$ that satisfy such constraint.

Up to now, we just described the state space of our population models. In order to capture their dynamics, we will specify a set of possible events, or transitions, that can change the state of the system. Each such event will involve just a small, fixed, number of agents, usually one or two, but we will in any case describe it from the perspective of the collective system.

Events are stochastic, and take an exponentially distributed time to happen, with a rate depending on the current global state of the system. Hence, each event will be specified by a rate function, and by a set of update rules, telling us how many and which agents are involved and how they will change state.

The set of events, or transitions, $\mathcal{T}^{(N)}$, is made up of elements $\tau \in \mathcal{T}^{(N)}$, which are pairs $\tau = (R_\tau, r_\tau^{(N)})$. More specifically, R_τ is a *multi-set* of *update rules* of the form $i \to j$, specifying that an agent changes state from i to j when the event fires. As R_τ is a multiset, we can describe events in which two or more agents in state i synchronise and change state to j. The exact number of agents synchronising can be extracted looking at the multiplicity of rule $i \to j$ in R_τ; let us denote such a number by $m_{\tau,i\to j}$. Note also that R_τ is independent of N, so that each transition involves a finite and fixed number of individuals.

In order to model the effect of event τ on the population vector, we will construct from R_τ the *update vector* $\mathbf{v}_\tau$ in the following way:

$$\mathbf{v}_{\tau,i} = \sum_{(i\to j)\in R_\tau} m_{\tau,i\to j}\mathbf{e}_j - \sum_{(i\to j)\in R_\tau} m_{\tau,i\to j}\mathbf{e}_i,$$

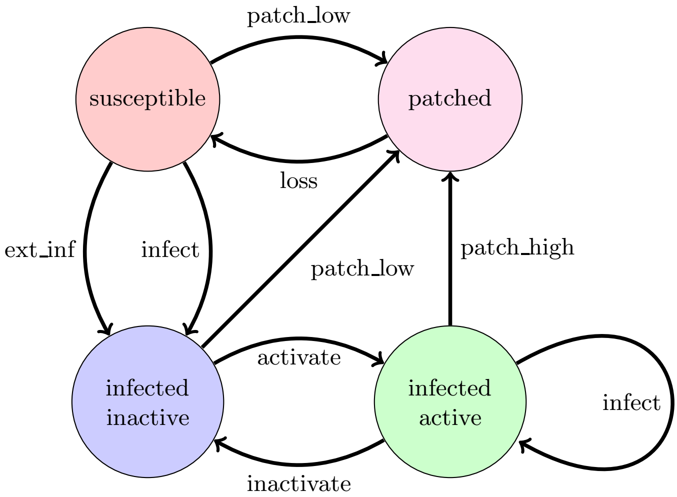


Fig. 1. States and transitions of a single computer in the p2p network epidemic model

where $\mathbf{e}_i$ is the vector equal to one in position i and zero elsewhere. Then, event τ changes the state from $\mathbf{X}^{(N)}$ to $\mathbf{X}^{(N)} + \mathbf{v}_\tau$.

The other component of event τ is the rate function $r_\tau^{(N)} : \mathcal{S}^{(N)} \to \mathbb{R}_{\geq 0}$, which depends on the current state of the system, and specifies the speed of the corresponding transition. It is assumed to be equal to zero if there are not enough agents available to perform a τ transition, and it is required to be *Lipschitz continuous* (when interpreted as a function on real numbers).

All these bits of information are collected together in the population model $\mathcal{X}^{(N)} = (\mathbf{X}^{(N)}, \mathcal{T}^{(N)}, \mathbf{x}_0^{(N)})$, where $\mathbf{x}_0^{(N)}$ is the initial state. Given such a model, it is straightforward to construct the CTMC $\mathbf{X}^{(N)}(t)$ associated with it, exhibiting its infinitesimal generator matrix. First, its state space is $\mathcal{S}^{(N)}$, while its infinitesimal generator matrix $Q^{(N)}$ is the $|\mathcal{S}^{(N)}| \times |\mathcal{S}^{(N)}|$ matrix defined by

$$q_{\mathbf{x},\mathbf{x}'} = \sum \{r_\tau(\mathbf{x}) \mid \tau \in \mathcal{T},\ \mathbf{x}' = \mathbf{x} + \mathbf{v}_\tau\}.$$

Remark 1. We note here that in this formalism we can still easily model multiple classes of agents. This can be done by partitioning the state space S into subsets, and allowing state changes only within a single subset. Furthermore, the rule set can be easily modified to include the possibility of birth and death events: we just need to add rules of the form $\emptyset \to i$ (birth of an agent in state i) or $i \to \emptyset$ (death of an agent in state i). Most of the theory presented below works for open models as well, see [13] for further details, but here we stick to the closed world assumption to simplify the presentation.

2.1 Running Example: A Worm Epidemic in a P2P Network

We introduce now the running example of this tutorial, which will be used to discuss the main ideas and algorithms. We consider a model of a worm epidemic in a peer-to-peer (P2P) network, which is comprised of N computers. For simplicity, we ignore new connections and disconnections, so that the number of nodes is constant (thus keeping the closed world assumption). Initially, nodes are vulnerable to the infection (susceptible S), and they can be infected by the worm in two ways, either by external infection (ext_inf), for instance by receiving an infected email message, or by the malicious action of an active infected node (infect). Infected nodes can themselves be in two states: active and inactive. An inactive infected node remains dormant and does not spread infection. In this way, the worm is harder to detect. An active node, instead, spreads the infection by sending messages to other computers in the network. The worm policy is to alternate between active and inactive states (activate and deactivate), to minimise the chances of being patched. All newly infected nodes start in the inactive state. Patching happens in all computers of the network (patch_s, patch_d, and patch_i), but we assume that the patching rate is higher for active nodes (patch_i), due to their anomalous activity in the P2P network. Patched nodes are immune, and cannot be infected by the worm. However, after some time the worm mutates, and immunity is lost (loss). A diagrammatic view of a network node is given in Figure 1.

To describe this system in the modelling language of this section, we need to specify the collective variables, which in this case are four: X_s, for susceptible nodes, X_d, for infected and dormant nodes, X_i for infected and active nodes, and X_p for patched nodes. Furthermore, we need 8 transitions or events whose rate and rule sets are described below:

ext_inf:	$R_{\mathtt{ext\_inf}} = \{s \to d\}$,	$r^{(N)}_{\mathtt{ext\_inf}} = k_{ext} X_s$;
infect:	$R_{\mathtt{infect}} = \{s \to d, i \to i\}$,	$r^{(N)}_{\mathtt{infect}} = \frac{k_{inf}}{N} X_s X_i$;
activate:	$R_{\mathtt{activate}} = \{d \to i\}$,	$r^{(N)}_{\mathtt{activate}} = k_{act} X_d$;
deactivate:	$R_{\mathtt{deactivate}} = \{i \to d\}$,	$r^{(N)}_{\mathtt{deactivate}} = k_{deact} X_i$;
patch_s:	$R_{\mathtt{patch\_s}} = \{s \to p\}$,	$r^{(N)}_{\mathtt{patch\_s}} = k_{low} X_s$;
patch_d:	$R_{\mathtt{patch\_d}} = \{d \to p\}$,	$r^{(N)}_{\mathtt{patch\_d}} = k_{low} X_d$;
patch_i:	$R_{\mathtt{patch\_i}} = \{i \to p\}$,	$r^{(N)}_{\mathtt{patch\_i}} = k_{high} X_i$;
loss:	$R_{\mathtt{loss}} = \{p \to s\}$,	$r^{(N)}_{\mathtt{loss}} = k_l X_p$;

In the previous list, the symbols $k.$ appearing in the rate functions are model parameters that describe the rate of an action involving a single object or a single pair of objects (for infect). Note that the parameter for the internal infection rate is divided by N. This corresponds to the classical *density dependent* assumption for epidemic models: each infected node sends messages to other random network nodes with rate k_i, thus hitting a susceptible node with probability X_s/N. The total rate of infection is then obtained by multiplying $k_i X_s/N$ by the number of infected nodes X_i.

3 Individual Agents Properties

We turn now to discuss the class of properties we are interested to check. As announced in the introduction, we will focus on individual agents, asking questions about the behaviour of an arbitrary individual agent in the system. These properties are quite common in performance models and in network epidemics [26], whenever we are interested in checking some aspect of the system from the point of view of a single user. For instance, in client/server systems, we may be interested in quality-of-service metrics, like the expected service time [38]. In network epidemics, instead, we may be interested in properties connected with the probability of a single node being infected in a certain amount of time, or in the probability of being patched before being infected [31]. Other classes of systems can be naturally queried from the perspective of a single agent, including ecological models [46] (survival chances of an individual or foraging patterns), single enzyme kinetics in biochemistry [43] (performance of an enzyme), but also crowd models [39] or public transportation models in a smart city.

Running Example. Some properties of interest of individual nodes in the worm epidemic model are listed below:

- What is the probability of a node being infected within T units of time?
- Is the probability of a single node remaining infected for T units of time smaller than p_1?
- Is the probability of a node being patched before getting infected larger than p_2?
- What is the probability of being patched within time T_1, and then remaining uninfected with probability at least p_3 for T_2 units of time?

What is shared by all those properties is the fact that they can be expressed in Continuous Stochastic Logic (CSL) [5], a well known extension to the stochastic setting of the non-deterministic Computational Tree Logic [20], which is also supported by the probabilistic model checker PRISM [37]. We will now introduce CSL, and then show how the previous properties can be expressed in this language.

3.1 Continuous Stochastic Logic

In this section we consider generic labelled stochastic processes [4,5]. A labelled stochastic process is a random process $Z(t)$, with state space S and a labelling function $L : S \to 2^{\mathcal{P}}$, associating with each state $s \in S$, a subset of atomic propositions $L(s) \subset \mathcal{P} = \{a_1, \ldots, a_k \ldots\}$ true in that state: each atomic proposition $a_i \in \mathcal{P}$ is true in s if and only if $a_i \in L(s)$. We require that all subsets of paths considered are measurable[1].

[1] Measurability is a technical condition that guarantees that the probability of a set is defined.

This is a very general definition, and encompasses all the cases we will encounter in the rest of the paper: CTMC, time-inhomogeneous CTMC, projections of CTMC on a subset of variables. In particular, the condition on measurability will always be satisfied. From now on, we always assume we are working with labelled stochastic processes.

A path of $Z(t)$ is a sequence $\sigma = s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} \ldots$, such that the probability of going from s_i to s_{i+1} at time $t_\sigma[i] = \sum_{j=0}^{i} t_j$, is greater than zero. For CTMCs, this condition is equivalent to $q_{s_i,s_{i+1}}(t_\sigma[i]) > 0$, where $Q = (q_{ij})$ is the infinitesimal generator matrix. We denote by $\sigma@t$ the state of σ at time t, with $\sigma[i]$ the i-th state of σ, and with $t_\sigma[i]$ the time of the i-th jump in σ.

A time-bounded CSL formula φ is defined by the following syntax:

$$\begin{aligned}\varphi &::= \mathit{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid P_{\bowtie p}(\psi)\\ \psi &::= \mathbf{X}^{[T_1,T_2]}\varphi \mid \varphi_1 \mathbf{U}^{[T_1,T_2]}\varphi_2\end{aligned}$$

where a is an atomic proposition, $p \in [0,1]$ and $\bowtie \in \{<,>,\leq,\geq\}$. φ are known as *state formulae* and ψ are *path formulae.*

The satisfiability relation of φ with respect to a labelled stochastic process $Z(t)$ is given by the following rules:

- $s, t_0 \models a$ if and only if $a \in L(s)$;
- $s, t_0 \models \neg\varphi$ if and only if $s, t_0 \not\models \varphi$;
- $s, t_0 \models \varphi_1 \wedge \varphi_2$ if and only if $s, t_0 \models \varphi_1$ and $s, t_0 \models \varphi_2$;
- $s, t_0 \models P_{\bowtie p}(\psi)$ if and only if $\mathbb{P}\{\sigma \mid \sigma,\, t_0 \models \psi\} \bowtie p$.
- $\sigma, t_0 \models \mathbf{X}^{[T_1,T_2]}\varphi$ if and only if $t_\sigma[1] \in [T_1, T_2]$ and $\sigma[1], t_0 + t_\sigma[1] \models \varphi$.
- $\sigma, t_0 \models \varphi_1 \mathbf{U}^{[T_1,T_2]}\varphi_2$ if and only if $\exists \bar{t} \in [t_0 + T_1, t_0 + T_2]$ s.t. $\sigma@\bar{t}, \bar{t} \models \varphi_2$ and $\forall t_0 \leq t < \bar{t}$, $\sigma@t, t \models \varphi_1$.

Note that the restriction to the time-bounded fragment of CSL means that we do not consider the steady state operator and we allow only time-bounded properties. This last restriction is connected with the nature of the fluid approximation (see Theorems 1 and 2 below), which hold only on finite time horizons. See [12,13] and Section 7 for further details.

Running Example. Consider the informal properties of the network epidemic model listed at the beginning of this section. We can easily rephrase them as CSL formulae (either state or path formulae), using the following atomic propositions, interpreted in the obvious way on the states of Figure 1: $a_{infected}$, $a_{patched}$. In the following, we use the convention that path formulae are denoted by ψ and state formulae are denoted by φ.

- $\psi_1 = F^{[0,T]} a_{infected}$ (a node is infected within T units of time);
- $\varphi_1 = P_{<p_1}(G^{[0,T]} a_{infected})$ (the probability of a single node remaining infected for T units of time is smaller than p_1);

- $\varphi_2 = P_{>p_2}(\neg a_{infected}\mathbf{U}^{[0,T]}a_{patched})$ (the probability of a node being patched before getting infected is larger than p_2);
- $\psi_2 = F^{[0,T_1]}(a_{patched} \wedge P_{\geq p_3}(G^{[0,T_2]}\neg a_{infected}))$ (a node is patched within time T_1, and then remains not infected with probability at least p_3 for T_2 units of time).

Model checking of CSL formulae for time-homogeneous CTMC proceeds bottom-up on the parse tree of the formula [5]. Checking atomic propositions and boolean operators is trivial. The difficult part is to compute the probability of path formulae. Then this probability is compared with the threshold in the quantifier operator to establish the truth of quantified path formulae. Computing the probability of a next CSL formula $P_{\bowtie p}(\mathbf{X}^{[T_1,T_2]}\varphi)$ is usually reduced to the computation of the integral, based on the probability of making a jump in the time interval and the probability that the new state satisfies φ. For a time-homogeneous CTMC this can be solved analytically. Dealing with until CSL formula $P_{\bowtie p}(\varphi_1\mathbf{U}^{[T_1,T_2]}\varphi_2)$ is more complex. For a time-homogeneous CTMC $Z(t)$, it can be reduced to the computation of two reachability problems, which themselves can be solved by transient analysis [5]. In particular, consider the sets of states $U = [\![\neg\varphi_1]\!]$ and $G = [\![\varphi_2]\!]$ and compute the probability of going from state $s_1 \notin U$ to a state $s_2 \notin U$ in T_1 time units, in the CTMC in which all U-states are made absorbing, $\pi^1_{s_1,s_2}(T_1)$. Furthermore, consider the modified CTMC in which all U and G states are made absorbing, and denote by $\pi^2_{s_2,s_3}(T_2 - T_1)$ the probability of going from a state $s_2 \notin U$ to a state $s_3 \in G$ in $T_2 - T_1$ units of time in such a CTMC. Then the probability of the until formula in state s can be computed as $P_s(\varphi) = \sum_{s_3\in G, s_2\notin U} \pi^1_{s_1,s_2}(T_1)\pi^2_{s_2,s_3}(T_2 - T_1)$. The probabilities π^1 and π^2 can be computed using standard methods for transient analysis (e.g. by uniformisation [28] or by solving the Kolmogorov equations [42]).

4 Fluid Approximation

In this section we will introduce some concepts of fluid approximation and mean field theory. The basic idea is to approximate a CTMC by an Ordinary Differential Equation (ODE), which can be interpreted in two different ways: it can be seen as an approximation of the average of the system (usually a first order approximation, see [15,50]), or as an approximate description of system trajectories for large populations. We will focus on this second interpretation, which corresponds to a functional version of the law of large numbers: instead of having a sequence of random variables, like the sample mean, converging to a deterministic value, like the true mean, in this case we have a sequence of CTMC (which can be seen as random trajectories in $\mathbb{R}^n$) for increasing population size, which converge to a deterministic trajectory, the solution of the fluid ODE.

In order to properly speak about convergence, we need to formally define the sequence of CTMC to be considered. The collective model of Section 2 depends on the total population N, yet models of different population sizes cannot be directly compared, as it would not make sense to compute a distance between a population of the size of hundreds with a population of the size of billions: the distance will be astronomically large because of the difference in population sizes. Hence, to make the comparison meaningful, we normalise the populations, by dividing each variable for the total population N. In this way, the normalised population variables $\hat{\mathbf{X}}^{(N)} = \frac{\mathbf{X}^{(N)}}{N}$, or population densities, will always range between 0 and 1 (for the closed world models we consider here), and so the behaviour for different population sizes can be compared. In the case of a constant population, normalised variables are usually referred to as the *occupancy measure*, as they represent the fraction of agents in each state.

When we perform the normalisation, we need to impose an appropriate scaling to update vectors, initial conditions, and rate functions of the normalised models. Let $\mathcal{X}^{(N)} = (\mathbf{X}^{(N)}, \mathcal{T}^{(N)}, \mathbf{X_0}^{(N)})$ be the non-normalised model with total population N and $\hat{\mathcal{X}}^{(N)} = (\hat{\mathbf{X}}^{(N)}, \hat{\mathcal{T}}^{(N)}, \hat{\mathbf{X}}_{\mathbf{0}}^{(N)})$ the corresponding normalised model. We require that:

- initial conditions scale appropriately: $\hat{\mathbf{X}}_{\mathbf{0}}^{(N)} = \frac{\mathbf{X_0}^{(N)}}{N}$;
- for each transition $(R_\tau, r_\tau^{(N)}(\mathbf{X}))$ of the non-normalised model, define $\hat{r}_\tau^{(N)}(\hat{\mathbf{X}})$ to be the rate function expressed in the normalised variables (obtained from $r_\tau^{(N)}$ by a change of variables). The corresponding transition in the normalised model is $(R_\tau, \hat{r}_\tau^{(N)}(\hat{\mathbf{X}}))$, with update vector equal to $\frac{1}{N}\mathbf{v}_\tau$.

We further assume, for each transition τ, that there exists a bounded and Lipschitz continuous function $f_\tau(\hat{\mathbf{X}}) : E \to \mathbb{R}^n$ on normalised variables (where E contains all domains of all $\hat{\mathcal{X}}^{(N)}$), independent of N, such that $\frac{1}{N}\hat{r}_\tau^{(N)}(\mathbf{x}) \to f_\tau(\mathbf{x})$ *uniformly* on E. In accordance with the previous subsection, we will denote the state of the CTMC of the N-th non-normalised (resp. normalised) model at time t as $\mathbf{X}^{(N)}(t)$ (resp. $\hat{\mathbf{X}}^{(N)}(t)$).

Running Example. Consider again the network epidemic model, which is easily seen to satisfy all the assumptions before. The conditions for the rate functions are easily verified. They hold trivially for linear rate functions, for instance $k_{ext}X_s = Nk_{ext}\frac{X_s}{N}$, and they also hold for the non-linear rate function modelling internal infections, due to the density dependent scaling of the rate constant with respect to the total population N, i.e. $\frac{k_{inf}}{N}X_sX_i = Nk_{inf}\frac{X_s}{N}\frac{X_i}{N}$.

4.1 Deterministic Limit Theorem

In order to present the "classic" deterministic limit theorem, consider a sequence of normalised models $\hat{\mathcal{X}}^{(N)}$ and let $\mathbf{v}_\tau$ be the (non-normalised) update vectors. The *drift* $F^{(N)}(\hat{\mathbf{X}})$ of $\hat{\mathcal{X}}$, which is formally the mean instantaneous increment of model variables in state $\hat{\mathbf{X}}$, is defined as

$$F^{(N)}(\hat{\mathbf{X}}) = \sum_{\tau \in \hat{\mathcal{T}}} \frac{1}{N}\mathbf{v}_\tau \hat{r}_\tau^{(N)}(\hat{\mathbf{X}}) \qquad (2)$$

Furthermore, let $f_\tau : E \to \mathbb{R}^n$, $\tau \in \hat{\mathcal{T}}$ be the limit rate functions of transitions of $\hat{\mathcal{X}}^{(N)}$. The *limit drift* of the model $\hat{\mathcal{X}}^{(N)}$ is therefore

$$F(\hat{\mathbf{X}}) = \sum_{\tau \in \hat{\mathcal{T}}} \mathbf{v}_\tau f_\tau(\hat{\mathbf{X}}), \tag{3}$$

and $F^{(N)}(\mathbf{x}) \to F(\mathbf{x})$ uniformly, as easily checked. The fluid ODE is

$$\frac{d\mathbf{x}}{dt} = F(\mathbf{x}), \quad with\ \mathbf{x}(0) = \mathbf{x_0} \in S.$$

Given that F is Lipschitz in E (since all f_τ are), this ODE has a unique solution $\mathbf{x}(t)$ in E starting from $\mathbf{x_0}$. Then, one can prove the following theorem:

Theorem 1 (Deterministic approximation [34,21]). *Let the sequence $\hat{\mathbf{X}}^{(N)}(t)$ of Markov processes and $\mathbf{x}(t)$ be defined as before, and assume that there is some point $\mathbf{x_0} \in S$ such that $\hat{\mathbf{X}}^{(N)}(0) \to \mathbf{x_0}$ in probability. Then, for any* finite *time horizon $T < \infty$, it holds that:*

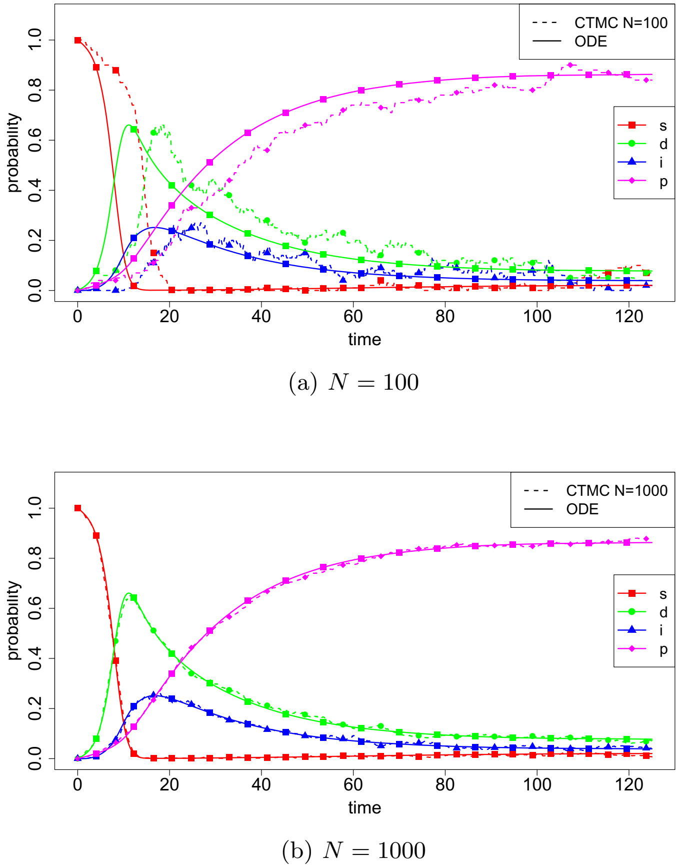
$$\mathbb{P}\left\{ \sup_{0 \le t \le T} ||\hat{\mathbf{X}}^{(N)}(t) - \mathbf{x}(t)|| > \varepsilon \right\} \to 0.$$

Running Example. Focus on the internal infection `infect`. It can be easily seen that the update vector associated with the rule set $R_{\mathtt{infect}}$ is $\mathbf{v}_{\mathtt{infect}} = (-1, 1, 0, 0)$, given the ordering (s, d, i, p) of S. Similarly for each of the other transitions. Hence, we obtain the following set of fluid ODEs, whose solution is compared with single trajectories of the CTMC, for different populations, in Figure 2:

$$\begin{cases} \dfrac{dx_s(t)}{dt} = -k_{ext}x_s - k_{inf}x_s x_i - k_{low}x_s + k_{loss}x_p \\ \dfrac{dx_d(t)}{dt} = k_{ext}x_s + k_{inf}x_s x_i - k_{act}x_d - k_{low}x_d + k_{deact}x_i \\ \dfrac{dx_i(t)}{dt} = k_{act}x_d - k_{deact}x_i - k_{high}x_i \\ \dfrac{dx_p(t)}{dt} = k_{low}x_s + k_{low}x_d + k_{high}x_i - k_{loss}x_p \end{cases} \tag{4}$$

4.2 Fast Simulation

We now consider what happens to a single individual in the population when the population size goes to infinity. Even if the collective behaviour tends to a deterministic process, an individual agent will still behave randomly. However, the fluid limit theorem implies that the dynamics of a single agent, in the limit, becomes independent of other agents, and it will sense them only through the



(a) $N = 100$

(b) $N = 1000$

Fig. 2. Comparison between the limit fluid ODE and a single stochastic trajectory of the network epidemic example, for total populations $N = 100$ and $N = 1000$. Parameters of the model are $k_{ext} = 0.01$, $k_{inf} = 5$, $k_{act} = 0.1$, $k_{deact} = 0.1$, $k_{low} = 0.005$, $k_{high} = 0.1$, $k_{loss} = 0.005$, while initial conditions are $\bar{X}_s(0) = 1$, $\bar{X}_d(0) = 0$, $\bar{X}_i(0) = 0$, and $\bar{X}_p(0) = 0$.

collective system state, described by the fluid limit. This asymptotic decoupling allows us to find a simple, time-inhomogenous, Markov chain for the evolution of the single agent, a result often known as *fast simulation* [22,24].

Let us explain this point formally. We focus on a single individual $Y_h^{(N)}(t)$, which is a (Markov) process on the state space $S = \{1, \dots, n\}$, conditional on the global state of the population $\hat{\mathbf{X}}^{(N)}(t)$. Denote by $Q^{(N)}(\mathbf{x})$ the infinitesimal generator matrix of $Y_h^{(N)}$, described as a function of the normalised state of the population $\hat{\mathbf{X}}^{(N)} = \mathbf{x}$, i.e.

$$\mathbb{P}\{Y_h^{(N)}(t+dt) = j \mid Y_h^{(N)}(t) = i, \hat{\mathbf{X}}^{(N)}(t) = \mathbf{x}\} = q_{i,j}^{(N)}(\mathbf{x})dt.$$

We stress that $Q^{(N)}(\mathbf{x})$ describes the exact dynamics of $Y_h^{(N)}$, conditional on $\hat{\mathbf{X}}^{(N)}(t)$, and that this process is *not independent* of $\hat{\mathbf{X}}^{(N)}(t)$. In fact, the marginal distribution of $Y_h^{(N)}(t)$ is not a Markov process.

This means that in order to capture its evolution in a Markovian setting, one has to consider the whole Markov chain $(Y_h^{(N)}(t), \hat{\mathbf{X}}^{(N)}(t))$.

The rate matrix $Q^{(N)}(\mathbf{x})$ can be constructed from the rate functions of global transitions by computing the fraction of the global rate seen by an individual agent that can perform it. To be more precise, let $r_\tau^{(N)}(\mathbf{X})$ be the rate function of transition τ, and suppose $i \to j \in R_\tau$ (and each update rule in R_τ has multiplicity one). Then, transition τ will contribute to the ij-entry $q_{ij}^{(N)}(\mathbf{X})$ of the matrix $Q^{(N)}(\mathbf{X})$ with the term $\frac{1}{X_i} r_\tau^{(N)}(\mathbf{X}) = \frac{1}{\hat{X}_i} \hat{r}_\tau^{(N)}(\hat{\mathbf{X}})$, which converges to $\frac{1}{\hat{X}_i} f_\tau(\hat{\mathbf{X}})$. Additional details about this construction (taking multiplicities properly into account) can be found in [12,13], see also the example below. From the previous discussion, it follows that the local rate matrix $Q^{(N)}(\mathbf{x})$ converges uniformly to a rate matrix $Q(\mathbf{x})$, in which all rate functions $\hat{r}_\tau^{(N)}$ are replaced by their limit f_τ. We now define two processes:

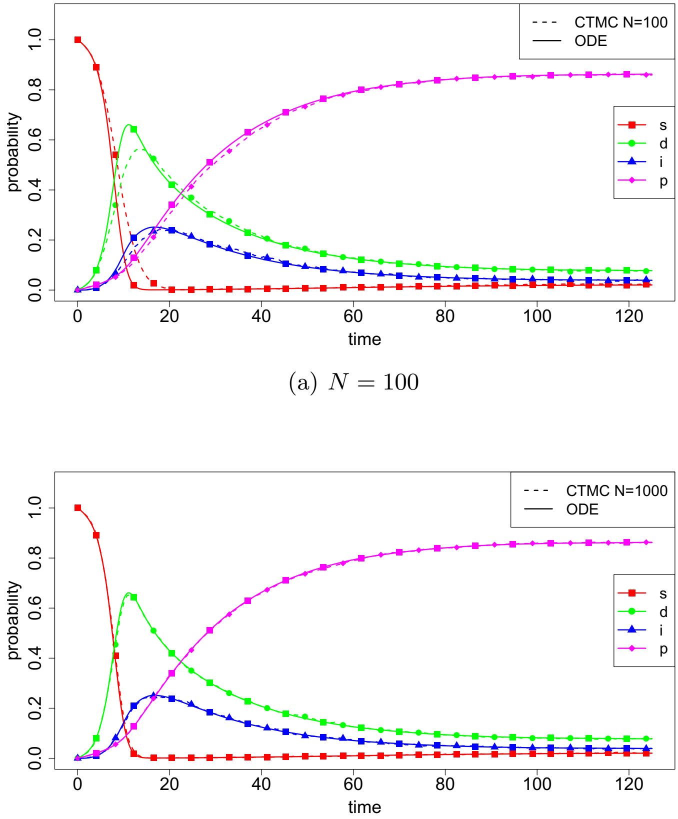
- $Z^{(N)}(t)$, which is the stochastic process describing the state of a random individual $Y_h^{(N)}(t)$ in a population of size N, marginalised with respect to the collective state $\hat{\mathbf{X}}^{(N)}(t)$.
- $z(t)$, which is a time-inhomogeneous CTMC (ICTMC), on the same state space S of $Z^{(N)}$, with time-dependent rate matrix $Q(\hat{\mathbf{x}}(t))$, where $\hat{\mathbf{x}}(t)$ is the solution of the fluid equation.

The following theorem can be proved [22]:

Theorem 2 (Fast simulation theorem). *For any $T < \infty$, $\mathbb{P}\{Z^{(N)}(t) \neq z(t),$ for some $t \leq T\} \to 0$, as $N \to \infty$.*

This theorem states that, in the limit of an infinite population, each agent will behave independently from all the others, sensing only the mean state of the global system, described by the fluid limit $\mathbf{x}(t)$. This *asymptotic decoupling* of the system, which can be generalised to any subset of $k \geq 1$ agents, is also known in the literature under the name of *propagation of chaos* [9].

Running Example. Consider again the worm epidemic, and focus on a single node in the network. In order to construct the local rate matrix $Q^{(N)}(\mathbf{x})$, we need to consider each single transition and compute the portion of rate function seen from a single node, dividing by the population variable corresponding to the local state involved in the transition. With reference to Figure 1, we obtain the following local rate functions:



(a) $N = 100$

(b) $N = 1000$

Fig. 3. Comparison between the solution of the Kolmogorov equations for the limit model of an individual agent and an estimate of the solution for an individual agent in a finite population, of size $N = 100$ or $N = 1000$. The estimate for the finite population has been obtained by statistical means, taking the sample average of the indicator functions of each local state, for a grid of 1000 time points. Averages have been taken from 10000 samples. Parameters of the model are as in Figure 2.

$$\begin{array}{lll}
\texttt{ext\_inf}: & s \rightarrow d, & \dfrac{1}{X_s} r^{(N)}_{\texttt{ext\_inf}} = k_{ext}; \\
\texttt{infect}: & s \rightarrow d, & \dfrac{1}{X_s} r^{(N)}_{\texttt{infect}} = \dfrac{1}{N} k_{inf} X_i = k_{inf} \hat{X}_i; \\
\texttt{activate}: & d \rightarrow i, & \dfrac{1}{X_d} r^{(N)}_{\texttt{activate}} = k_{act}; \\
\texttt{deactivate}: & i \rightarrow d, & \dfrac{1}{X_i} r^{(N)}_{\texttt{deactivate}} = k_{deact}; \\
\texttt{patch\_s}: & s \rightarrow p, & frac1X_s r^{(N)}_{\texttt{patch\_s}} = k_{low};
\end{array}$$

$$\texttt{patch\_d}: d \to p,\ \frac{1}{X_d} r^{(N)}_{\texttt{patch\_d}} = k_{low};$$

$$\texttt{patch\_i}: i \to p,\ \frac{1}{X_i} r^{(N)}_{\texttt{patch\_i}} = k_{high};$$

$$\texttt{loss}: \quad p \to s,\ \frac{1}{X_p} r^{(N)}_{\texttt{loss}} = k_{loss};$$

Note that all the local rate functions are independent of N, and so $Q^{(N)}(\mathbf{x}) = Q(\mathbf{x})$. Ordering S as (s, d, i, p), it follows that

$$Q(\mathbf{x}) = \begin{pmatrix} -k_{ext} - k_{inf}x_i - k_{low} & k_{ext} + k_{inf}x_i & 0 & k_{low} \\ 0 & -k_{act} - k_{low} & k_{act} & k_{low} \\ 0 & k_{deact} & -k_{deact} - k_{high} & k_{high} \\ k_{loss} & 0 & 0 & -\, k_{loss} \end{pmatrix}$$

Therefore, the limit ICTMC of an individual agent depends on the solution of the fluid equation only via the fraction of infected and active nodes, $x_i(t)$. A numerical comparison of the transient probability for the limit individual agent $z(t)$ and an individual agent $Z^{(N)}(t)$ in a finite population model (for $N = 100$ and $N = 1000$) is shown in Figure 3. For $N = 1000$, it is almost impossible to distinguish between the two transient probabilities.

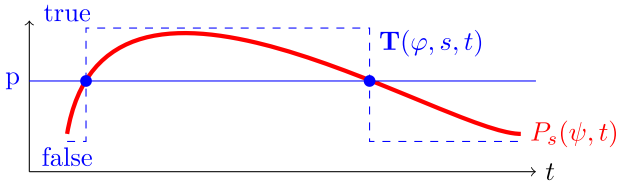
5 Checking CSL Properties for Individual Agents

CSL model checking is computationally expensive and can become prohibitively so for large CTMC models, such as population models. The same is true of transient analysis of CTMCs but fluid approximation has provided a highly efficient means to obtain high quality approximations for population models in this case. Therefore it is natural to consider whether fluid approximation can also be exploited to find good, efficient approximations in CSL model checking. For properties that relate to a single arbitrary agent in a population model, we will demonstrate that this is indeed the case if we exploit the fast simulation property established in Theorem 2.

In the fluid approximation of a CSL property φ for an arbitrary individual agent $Z^{(N)}(t)$ in a population of size N we exploit Theorem 2 and replace $Z^{(N)}(t)$ by its fluid limit $z(t)$, checking φ on $z(t)$. The essential advantage in doing this is that, in order to properly compute the satisfaction of CSL formulae for $Z^{(N)}(t)$ we need to take into account the whole population model $\hat{\mathbf{X}}^{(N)}(t)$. This results in a huge state space that is out of reach of CSL model checkers. Simulation-based methods, like statistical model checking [29], may be exploited for this purpose for moderately sized populations (this is what we do to compare our approximate method with the results for the proper stochastic model), but simulation becomes extremely costly when the population increases.

As we will show in the rest of the chapter, checking properties on $z(t)$ is much more efficient, and the error seems to remain small. In addition to experimental validation, Theorem 2 provides us with the means of formally showing that the result of checking CSL properties on $z(t)$ and on $Z^{(N)}(t)$ will be the same, provided N is sufficiently large.

In replacing $Z^{(N)}(t)$ with $z(t)$, however, we have to face the fact that $z(t)$ is a time-inhomogeneous CTMC. It turns out that working with ICTMC is much more complex, because of the dependency of the satisfaction of a formula on time. In fact, if we inspect the definition of CSL semantics in Section 3.1, we can observe that the satisfaction relation depends on a state of the model and on the time at which the formula is evaluated. This particularly affects the computation of the probabilities of path formulae. In the case of time-homogeneous CTMC, time dependency is not an issue, as rates are constant, hence starting the system at time $t_0 > 0$ is the same as starting it at time 0. But when rates depend on time, this is no longer the case. What can happen is schematically depicted in the figure below.



In this figure, we show a hypothetical scenario in which the probability of a path formula ψ is plotted against the time t at which the formula is evaluated. When we compare this function with the threshold p in the probability operator of $\varphi = P_{\bowtie p}(\psi)$, it can happen that this function is above p for some time instants and below it for some other time instants. It follows that the truth $\mathbf{T}(\varphi, s, t)$ of a CSL temporal formula in a state $s \in S$, can itself depend on the time at which the formula is evaluated. This makes CSL model checking for time-inhomogeneous CTMC a much more delicate business: we need to compute not a single probability for a path formula, but its probability as a function of time, and we further need to take into account the time-dependent truth of CSL formulae when checking nested formulae.

In the rest of the chapter, we will first discuss how to compute next state and reachability probabilities (the two main building blocks of CSL algorithms) when the next-state set or the goal/unsafe sets are independent of time (Sections 5.1 and 5.2), facing also the problem of computing the dependency of such probabilities on the initial time (Section 6). Then, we will move to nested CSL formulae, and give an intuition on how to compute path probabilities in the case of nested temporal operators (Section 7). Finally, in Section 8 we will briefly discuss theoretical properties, like decidability of the algorithms and convergence of CSL truth as population increases.

5.1 Next State Probabilities

In this section, we will show how to compute the probability that the next state in which an agent jumps belongs to a given set of states $S_0 \subseteq S$, constraining the jump to happen between time $[t_0 + T_1, t_0 + T_2]$, where t_0 is the current time. This is clearly at the basis of the computation of the probability of next path formulae.

Let $Z(t)$ be a CTMC with state space S and infinitesimal generator matrix $Q(t)$. We indicate with $P_{next}(Z, t_0, T_1, T_2, S_0)[s]$ the probability of the set of trajectories of Z jumping into a state in S_0, starting at time t_0 in state s, within time $[t_0 + T_1, t_0 + T_2]$. Hence, $P_{next}(Z, t_0, T_1, T_2, S_0)$ is a vector of probabilities, one for each state $s \in S$.

For any fixed t_0, the probability $P_{next}(Z, t_0, T_1, T_2, S_0)[s]$ is given by the following integral [51,30]

$$P_{next}(Z, t_0, T_1, T_2, S_0)[s] = \int_{t_0+T_1}^{t_0+T_2} q_{s,S_0}(t) \cdot e^{-\Lambda(t_0,t)[s]} dt, \tag{5}$$

where $\Lambda(t_0, t)[s] = \int_{t_0}^{t} -q_{s,s}(\tau) d\tau$ is the cumulative exit rate of state s from time t_0 to time t, and $q_{s,S_0}(t) = \sum_{s' \in S_0, s' \neq s} q_{s,s'}(t)$ is the rate of jumping from s to a state $s' \in S_0$, $s' \neq s$, at time t.

Equation 5 can be explained as follows: first of all, remember that for an exponential distribution, the probability density of the first jump happening at time t, given that we are in state s at time t_0, is $\Lambda(t_0, t)[s]e^{-\Lambda(t_0,t)[s]}$. For a time homogeneous CTMC, it holds that $\Lambda(t_0, t)[s] = \lambda_s(t - t_0)$, where λ_s is the exit rate from state s, hence the density takes the more common form. Furthermore, if we jump at time t, then the probability of landing in a state of S_0 is $q_{s,S_0}(t)/\Lambda(t_0, t)[s]$. Multiplying this for the probability density above, we obtain the probability density of jumping in a state of S_0 at time t, which is the argument of the integral (5). Then we only need to integrate it from time $t_0 + T_1$ to time $t_0 + T_2$ to compute the desired quantity.

In order to practically compute $P_{next}(Z, t_0, T_1, T_2, S_0)[s]$ for a given t_0, we can either numerically compute the integral, or transform it into a differential equation, and integrate the so-obtained ODE with standard numerical methods. This second method is preferrable, as it can be extended to compute the next probability as a function of the initial time, see Section 6. More specifically, we can introduce two variables, P (giving the desired probability) and L (giving the cumulative rate Λ), initialise $P(t_0 + T_1) = 0$ and $L(t_0 + T_1) = \Lambda(t_0, t_0 + T_1)$, and then integrate the following two ODEs from time $t_0 + T_1$ to time $t_0 + T_2$:

$$\begin{cases} \dfrac{d}{dt}P(t) = q_{s,S_0}(t) \cdot e^{-L(t)} \\ \dfrac{d}{dt}L(t) = -q_{s,s}(t) \end{cases} \tag{6}$$

Running Example. We consider the path formula $\psi = \mathbf{X}^{[T_1,T_2]} a_{infected}$, which expresses the fact that a node of the network will change state between time $t_0 + T_1$ and $t_0 + T_2$, and it will become (or remain) infected. In Figure 4, we show the probability of the path formula for the fluid limit model of a single node in the network, initially in the susceptible state s, for $t_0 = 0$ and $T_1 = 0$, as a function of $T_2 = T$.

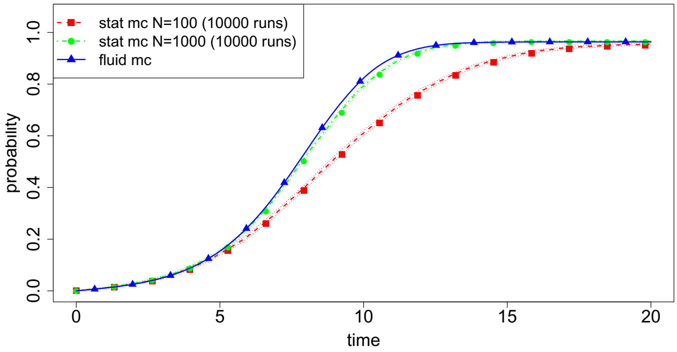


Fig. 4. Path probability of $\mathbf{X}^{[0,T]} a_{infected}$, as a function of T, starting in state s at time 0. The fluid approximation (continuous line) is compared with the statistical estimate (computed using statistical model checking out of an ensemble of 10000 runs) for population levels of $N = 100$ and $N = 1000$. Binomial confidence intervals are reported in the plot (they are quite narrow) and parameters of the model are as in Figure 2.

5.2 Reachability Probabilities

We now turn to the computation of reachability probabilities of an individual agent. Essentially, we want to compute the probability of the set of traces reaching some goal state $G \subseteq S$ within T units of time, given that we are in state $s \in S$ at time t_0, and avoiding unsafe states $U \subseteq S$, which will be denoted by $P_{reach}(Z, t_0, T, G, U)[s]$, where $Z(t)$ is a ICTMC on S, with rate matrix $Q(t)$ and initial state $Z(0) = Z_0 \in S$.

The computation of reachability probabilities is the key operation needed to compute probabilities of until formulae. In fact, the probability of the path formula $\varphi_1 \mathbf{U}^{[0,T]} \varphi_2$ is the probability of reaching a goal set $G = \{s \mid s \models \varphi_2\}$, avoiding unsafe states $U = \{s \mid s \models \neg\varphi_1\}$. Here we are assuming the satisfaction of φ_1 and φ_2 does not depend on time. We will solve this reachability problem in a standard way, by reducing it to the computation of transient probabilities in a modified ICTMC [5], similarly to [18].

Let $\Pi(t_1, t_2)$ be the probability matrix of $Z(t)$, in which entry $\pi_{s_1,s_2}(t_1, t_2)$ gives the probability of being in state s_2 at time t_2, given that Z was in state s_1 at time t_1. The *Kolmogorov forward and backward equations* [42] describe the time evolution of $\Pi(t_1, t_2)$ as a function of t_2 and t_1, respectively. More precisely, the forward equation is

$$\frac{\partial \Pi(t_1, t_2)}{\partial t_2} = \Pi(t_1, t_2) Q(t_2),$$

while the backward equation is

$$\frac{\partial \Pi(t_1,t_2)}{\partial t_1} = -Q(t_1)\Pi(t_1,t_2).$$

The probability $P_{reach}(Z,t_0,T,G,U)$, for a given initial time t_0, can be solved integrating the forward Kolmogorov equation (with initial value given by the identity matrix) in the ICTMC $Z'(t)$ in which all unsafe states and goal states are made absorbing [5]. The infinitesimal generator matrix $Q'(t)$ of $Z'(t)$ is obtained from $Q(t)$ by setting $q'_{s_1,s_2}(t) = 0$, for each $s_1 \in G \cup U$.

In particular, $P_{reach}(Z,t_0,T,G,U) = \Pi'(t_0,t_0+T)\mathbf{e}_G$, where $\mathbf{e}_G$ is an $n \times 1$ vector equal to 1 if $s \in G$ and 0 otherwise, and Π' is the probability matrix of the modified ICTMC Z'.[2] We emphasise that, in order for the initial value problem defined by the Kolmogorov forward equation to be well posed, the infinitesimal generator matrix $Q(t)$ has to be sufficiently regular (e.g. bounded and integrable).

Running Example. Consider again the running example, and the path formula $\psi_1 = F^{[0,T]}a_{infected}$, expressing the fact that a node will be infected within T units of time, starting from time t_0. The path probability of ψ_1 can be recast into the computation of a reachability probability, with goal set $G = \{d,i\}$ and unsafe set $U = \emptyset$. Applying the method discussed above, we just need to compute the transient probability for the ICTMC with rate matrix

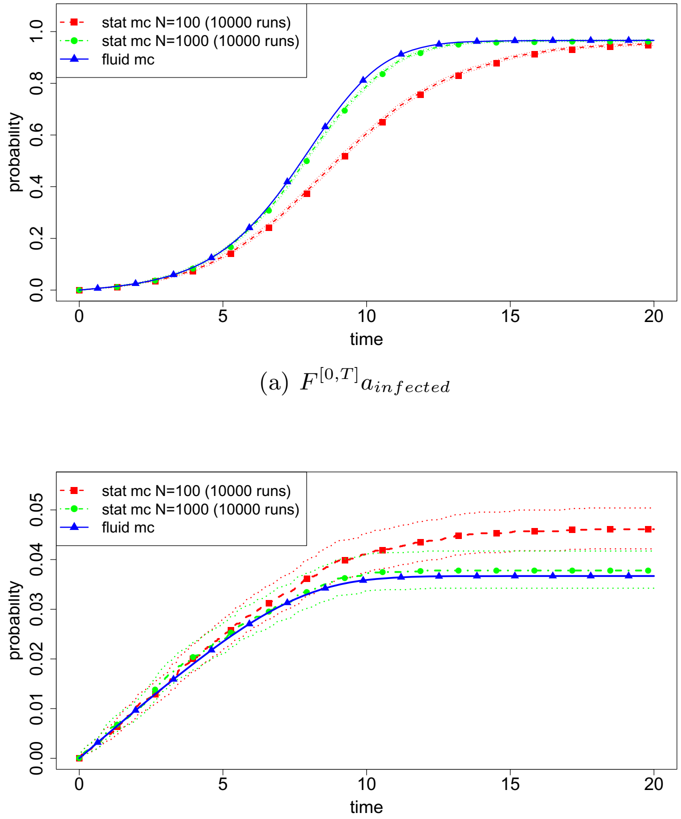
$$Q'(\mathbf{x}) = \begin{pmatrix} -k_{ext} - k_{inf}x_i - k_{low} & k_{ext} + k_{inf}x_i & 0 & k_{low} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k_{loss} & 0 & 0 & -k_{loss} \end{pmatrix}$$

in which states d and i are made absorbing. The result, starting from $t_0 = 0$ and as a function of T, is shown in Figure 5(a).
In Figure 5(b), instead, we show the result of computing the probability of the path formula $\psi_3 = a_{not\ infected}\mathbf{U}^{[0,T]}a_{patched}$, for $t_0 = 0$, as a function of T. Here, we just need to compute the reachability probability for the goal set $G = \{p\}$ and unsafe set $U = S \setminus \{s,p\} = \{d,i\}$.

6 Time Dependent Path Probabilities

In this section we will present a method to compute next state and reachability probabilities as a function of the time t_0 at which the property is evaluated. As we have already discussed, this is the crucial step to check nested CSL formulae. In fact, the satisfaction of a CSL formula depends on the time at which the formula is evaluated. This is particularly the case for a quantified path formula like $\varphi = P_{\leq p}(\psi)$, where ψ can be an until or a next formula. In this case, the

[2] Clearly, alternative ways of computing the transient probability, like uniformization for ICTMC [3], could also be used. However, we stick to the ODE formulation in order to deal with dependency on the initial time t_0.



(a) $F^{[0,T]}a_{infected}$

(b) $\neg a_{infected}\mathbf{U}^{[0,T]}a_{patched}$

Fig. 5. Path probability of the formulas $F^{[0,T]}a_{infected}$ and $\neg a_{infected}\mathbf{U}^{[0,T]}a_{patched}$, as a function of T, starting in state s at time 0. The fluid approximation (continuous line) is compared with the statistical estimate (computed using statistical model checking out of an ensemble of 10000 runs) for population levels of $N = 100$ and $N = 1000$. Binomial confidence intervals are reported in the plot, and parameters of the model are as in Figure 2.

probability of the path formula ψ from state $s \in S$, $P(t_0, \psi)[s]$, depends on the initial time t_0 at which we start evaluating such a formula. Hence, $P(t_0, \psi)[s]$ is a function of t_0, and when we evaluate the inequality $P(t_0, \psi)[s] \leq p$, needed to establish the truth of φ, we may find that the inequality holds for some t_0 and is falsified for other t_0 (see again the figure in Section 5).

Hence, we need a way to compute the probability of a path formula as a function of time. The starting point for this will be the formulation of next state and reachability probabilities as solutions of a differential equation. In fact, we will derive other differential equations whose solution will return the probability of path formulae as a function of the initial time.

Before presenting the derivation of the ODE more formally, let us comment on the choice of using ODE-based methods to compute transient probabilities, rather than more standard unifomisation-based algorithms. The first and more fundamental reason is precisely connected with the dependency of path probabilities on the initial time: uniformisation algorithms do not generalise easily to such scenarios. Moreover, the size of the state space of a single individual is usually very small, even when the collective system has a huge state space. Hence, numerical solvers for ODEs will work fine and will be efficient. Additionally, the fluid limit for an individual agent depends on the solution of the fluid ODE, so in any case we need to resort to ODE solvers. Coupling all the ODEs together allows us to exploit adaptive solvers [17] in order to control and reduce the global error.

Time-Dependent Next Probabilities. We will start by showing how to compute the next-state probability $P_{next}(Z, t_0, T_1, T_2, S_0)[s]$ as a function of t_0: $\bar{P}_s(t_0) = P_{next}(Z, t_0, T_1, T_2, S_0)[s]$. Computing the integral (5) for any t_0 is obviously not feasible. However, we can exploit the differential formulation of the problem, and define a set of ODEs with the initial time t_0 as an independent variable. First, we can compute the derivative of $\bar{P}_s(t_0)$ with respect to t_0 and obtain

$$\begin{aligned}\frac{d}{dt_0}\bar{P}_s(t_0) &= q_{s,S_0}(t_0+T_2)\cdot e^{-\Lambda(t_0,t_0+T_2)} - q_{s,S_0}(t_0+T_1)\cdot e^{-\Lambda(t_0,t_0+T_1)} \\ &\quad + \int_{t_0+T_1}^{t_0+T_2} \frac{\partial}{\partial t_0} q_{s,S_0}(t)\cdot e^{-\Lambda(t_0,t)}dt \\ &= q_{s,S_0}(t_0+T_2)\cdot e^{-\Lambda(t_0,t_0+T_2)} - q_{s,S_0}(t_0+T_1)\cdot e^{-\Lambda(t_0,t_0+T_1)} \\ &\quad - q_{s,s}(t_0)\bar{P}_s(t_0)\end{aligned}$$

Consequently, we can compute the next-state probability as a function of t_0 by solving the following set of ODEs:

$$\begin{cases}\frac{d}{dt}\bar{P}_s(t) = q_{s,S_0}(t+T_2)\cdot e^{-L_2(t)} - q_{s,S_0}(t+T_1)\cdot e^{-L_1(t)} - q_{s,s}(t)\bar{P}_s(t) \\ \frac{d}{dt}L_1(t) = q_{s,s}(t) - q_{s,s}(t+T_1) \\ \frac{d}{dt}L_2(t) = q_{s,s}(t) - q_{s,s}(t+T_2)\end{cases} \tag{7}$$

where $L_1(t) = \Lambda(t, t+T_1)$ and $L_2(t) = \Lambda(t, t+T_2)$.
Initial conditions are $P_s(t_0) = P_{next}(Z, t_0, T_1, T_2, S_0)[s]$, $L_1(t_0) = \Lambda(t_0, t_0+T_1)$, and $L_2(t_0) = \Lambda(t_0, t_0+T_2)$, and are computed solving the equations (6).

Running Example. We consider again the next path formula $\psi = \mathbf{X}^{[0,T]}a_{infected}$, fix $T = 7.5$, and compute its path probability $\bar{P}_s(t_0)$, from the susceptible state s, as a function of t_0. In order to do this, we need to first solve the ODEs (6) for

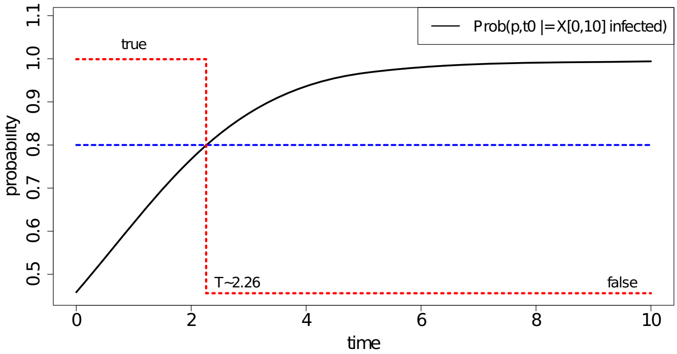


Fig. 6. Fluid estimate of the path probability of the formula $\mathbf{X}^{[0,T_2]}a_{infected}$, for $T_2 = 7.5$, as a function of the initial time t at which the formula is evaluated. We assume we start in state s at time t. The red dotted line represents the time-dependent truth value of the formula $P_{\leq 0.8}(\mathbf{X}^{[0,7.5]}a_{infected})$ in state s. Parameters of the model are as in Figure 2.

$t_0 = 0$ and $T = 7.5$, in order to obtain the initial conditions of the ODEs (7). Then, we need to solve the following ODE system:

$$\begin{cases} \frac{d}{dt}\bar{P}_s(t) = (k_{ext} + k_{inf}x_i(t+T))e^{-L_2(t)} - (k_{ext} + k_{inf}x_i(t)) + \dots \\ \qquad\qquad + (k_{ext} + k_{inf}x_i(t) + k_{low})\bar{P}_s(t) \\ \frac{d}{dt}L_2(t) = k_{inf}(x_i(t+T) - x_i(t)) \\ L_1(t) = 0 \end{cases}$$

Its solution is shown in Figure 6, together with the truth of the formula $P_{\leq 0.8}(\mathbf{X}^{[0,T_2]}a_{infected})$ in state s, for $t_0 \in [0, 10]$. As we can see, the formula is initially true and then, at time $t = 2.26$, it changes truth status and becomes false.

Time-Dependent Reachability. We now turn to the problem of computing $P(t) = P_{reach}(Z, t, T, G, U)$ as a function of $t \in [t_0, t_1]$. Recall that in Section 5.2 we reduced the computation of $P(t)$, for a fixed initial time t, to the solution of the Kolmogorov forward equation of the modified ICTMC, in which goal and unsafe sets are made absorbing. Stated otherwise, we used the forward equation to compute $\mathit{\Pi}(t, t')$, from $t' = t$ to $t' = t + T$. To compute the reachability probability as a function of the initial time $t \in [t_0, t_1]$, for T fixed, we need to compute $\mathit{\Pi}(t, t+T)$ for $t \in [t_0, t_1]$. We can do this using the chain rule and combining the forward and the backward Kolmogorov equation to obtain the following ODE for $\Pi(t, t+T)$:

$$\begin{aligned}\frac{d\Pi(t,t+T)}{dt} &= \frac{\partial\Pi(t,t+T)}{\partial t} + \frac{\partial\Pi(t,t+T)}{\partial(t+T)}\frac{d(t+T)}{dt} \\ &= -Q(t)\Pi(t,t+T) + \Pi(t,t+T)Q(t+T).\end{aligned} \tag{8}$$

Here the initial condition is $\Pi(t_0, t_0+T)$, and it can be computed by solving the Kolmogorov forward equation. Using a numerical ODE solver, we can integrate this equation and obtain $\Pi(t, t+T)$ for $t \in [t_0, t_1]$. This gives the basic algorithm to compute probabilities $P_s(\psi, t)$ of until path formulae like $\psi = \varphi_1 \mathbf{U}^{[0,T]} \varphi_2$ from a state s: we just need to compute the reachability probability $\pi_{s,s'}(t, t+T)$ and add it over states $s' \in G$.[3] Once the until probability is computed, we can check if state s satisfies $P_{\bowtie p}(\psi)$ at time t by comparing the value $P_s(t)$ with the threshold p. Doing this for all times $t \in [t_0, t_1]$ requires us to find all zeros of the function $P_s(t) - p$. This can be done during the integration of the ODEs, using event detection routines, provided the number of zeros is finite and the function $P_s(t) - p$ always changes sign around a zero. This does not hold in all cases, and further restrictions on the rate functions and the thresholds p have to be made, see [12,13] and Section 8 below for more details.

Running Example. Consider the formula $\psi_4 = G^{[0,T_2]} \neg a_{infected}$, fix T_2 to 10, and evaluate it as a function of the initial time. In order to apply the reachability algorithm above, we need to turn the always operator into an until. This is done in the standard way, as $G^{[0,T_2]} \neg a_{infected} \equiv \neg F^{[0,T_2]} a_{infected} \equiv \neg(true \mathbf{U}^{[0,T_2]} a_{infected})$. Hence, we need to compute the reachability probability for the goal set $G = \{i, d\}$ and the unsafe set $U = \emptyset$, and then compute 1 minus this probability. The result for the patched state p is shown in Figure 7, for the initial time varying between 0 and 150.

If we now consider the state formula $P_{\geq 0.97}(\psi_4)$ and focus again on the patched state p, then we see in Figure 7 that the formula is false from time 0 to time $T \approx 81.8$ and then becomes true.

7 Nested CSL Formulae

Computing the truth of nested CSL formulae for time-homogeneous CTMC is the same as for non-nested formulae. For an until or next temporal operator, we first solve the model checking problem for its subformulae, hence establish if a state satisfies or falsifies them, and then we use this information in the standard algorithms (e.g. the reachability algorithm based on transient analysis for the until case). Unfortunately, this simple recipe does not work for time-inhomogeneous

[3] More general until formulae $\varphi_1 \mathbf{U}^{[T_1,T_2]} \varphi_2$, for $T_1 > 0$, can be dealt by a minor modification of the approach, see [12,13] for further details.

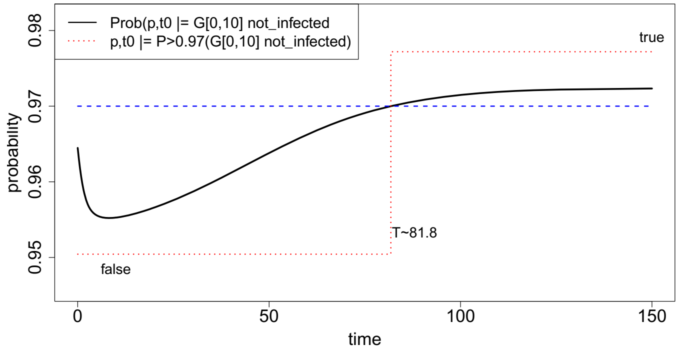
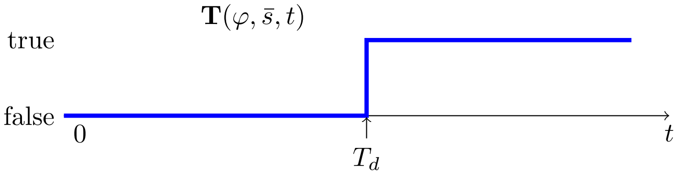


Fig. 7. Fluid estimate of the path probability of the formula $G^{[0,T_2]}\neg a_{infected}$, for $T_2 = 10$, as a function of the initial time t at which the formula is evaluated, starting from the patched state p at time t (continuous black line). Then the path probability is compared with the threshold $p = 0.97$, and the truth value of the CSL state formula $P_{\geq 0.97}(G^{[0,10]}\neg a_{infected})$ is computed, as a function of the time t at which it is evaluated. The time-dependent truth is depicted in the red dotted line. Parameters of the model are as in Figure 2.

CTMC. The problem is that a state satisfies a subformula containing a temporal operator depending on the time in which the formula is evaluated. This fact introduces an extra dimension of complexity into the algorithms.

To give a flavour of the problems involved, let us discuss the nested path formula $\psi = F^{[0,T]}(P_{\geq p}(\varphi_1 \mathbf{U}^{[0,T_1]}\varphi_2))$, where φ_1 and φ_2 are boolean combinations of atomic propositions, so that their truth in a state does not depend on the time at which we evaluate them. Clearly, using the procedure put forward in the previous section, we can compute, for each state $s \in S$, the probability $P_s(t)$ of the path formula $\varphi_1 \mathbf{U}^{[0,T_a]}\varphi_2$, as a function of the time at which we evaluate it. Therefore, we can compute the time-dependent truth function $\mathbf{T}(\varphi, s, t)$ of the state formula $\varphi = P_{\geq p}(\varphi_1 \mathbf{U}^{[0,T_a]}\varphi_2)$ by finding the zeros of $P_s(t) - p$, for each state $s \in S$. Specifically, we obtain that $\mathbf{T}(\varphi, s, t) = true$ if and only if $s, t \models \varphi$, and $false$ otherwise. In general, this function will keep jumping between 0 ($false$) and 1 ($true$), and we can represent it by identifying and storing in a list all time instants T_d at which a change in the truth of $s, t \models \varphi$ happens.

Fix now a state $\bar{s}$, and assume that at time T_d the function $\mathbf{T}(\varphi, \bar{s}, t)$ has a discontinuity, and that φ was false in $\bar{s}$ before time T_d and it is true afterwards. Now, focus the attention on the path formula $\psi = F^{[0,T]}(\varphi)$. To compute its path probability, we need to compute the probability of reaching a state satisfying φ within T time units. This is done by making φ-states absorbing, and computing the transient probability in the so-modified Markov chain. If $T_d < T$, then $\bar{s}$ is not a goal state from time 0 to time T_d, and becomes a goal state at time T_d, as shown in the figure below.



The first consequence of this fact is that the modified Markov chain in which goal states are made absorbing has a structure that changes in time, according to the truth of formula φ. In particular, $\bar{s}$ is not absorbing from time 0 to T_d, and becomes absorbing at time T_d.

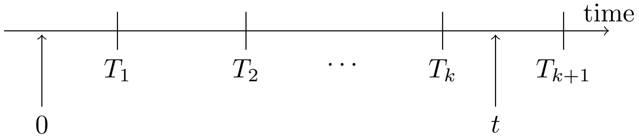
Now fix a non-goal state s' (remaining non-goal for the whole time interval $[0, T]$) and focus on the reachability probability starting from this state. In particular, there can be a non-null probability to go from s' to $\bar{s}$ in T_d units of time, starting at time 0. This means that the probability of the set of trajectories starting at s' at time 0 and being in $\bar{s}$ at time T_d, without passing from a goal state, has non-null probability. Pick one such trajectory, which clearly does not contribute to the reachability probability form s'. At time T_d, however, the structure of the modified CTMC changes, and this trajectory suddenly satisfies the reachability condition. In particular, this holds at time T_d for all the trajectories that are in $\bar{s}$. Hence, at time T_d the probability $\pi_{s',\bar{s}}(0, T_d)$ has to be added to the reachability probability $P_{s'}(T_d^-)$, as computed before $\bar{s}$ becomes a goal state.

Stated otherwise, a change in the truth status of the formula φ not only forces us to *change the topology* of the modified CTMC, by altering the set of absorbing states, but it may also *induce a discontinuity* in the path probability.

Computing Reachability Probabilities for Time-Varying Sets. We will quickly sketch now an algorithmic procedure to compute reachability probabilities when the goal and unsafe sets vary with time. This will provide the key procedure to compute path probabilities for nested until formulae of the form $\varphi_1 \mathbf{U}^{[0,T]} \varphi_2$, for general CSL formulae φ_1 and φ_2. For more general until formulae $\varphi_1 \mathbf{U}^{[T_a,T_b]} \varphi_2$, and for next path formulae, we refer the reader to [12,13].

We first discuss the case in which the unsafe set U is always empty, corresponding to eventually path formulae. To compute the reachability probability, we need to take the double nature of states into account: a state s can be either goal or non-goal, and its status can vary with time. To better describe this scenario, we will double the state space, creating for each state $s \in S$ a shadow copy $\bar{s}$, which represents the goal version of s. Shadow states $\bar{s}$ are always absorbing. Each transition in the CTMC entering an s state, instead, is directed towards s if and only if s is non-goal. Otherwise, it is rerouted towards $\bar{s}$.[4] This routing has to be changed whenever we hit a discontinuity in the time-dependent truth function. The situation is depicted below.

[4] Alternatively, we can add a new goal state s^*, as done in [32], and redirect all transitions entering any goal state to s^*.



Let $T_1, \ldots, T_k, T_{k+1}, \ldots$ be the times in which any state of S changes status (from goal to non-goal, or vice versa). Note that these time instants are fixed, and we can assume them to be known. To compute the reachability probability within time $[0, T]$, we start in $T_0 = 0$ by constructing the modified CTMC according to the goal set $G(0)$ at time 0. In between T_0 and T_1, the structure of the modified CTMC does not change, hence we can integrate the forward Kolmogorov equations, until the time t hits the first discontinuity time T_1. At this time, we need to perform some operations, according to whether the state s changes status to become a goal state or a non-goal state.

Goal to non-goal: in this case, we only need to reroute the transition matrix of the CTMC: all transitions entering $\bar{s}$, the shadow version of s, must now point to s. This is obtained by modifying the Q-matrix accordingly, deleting entries in the column $\bar{s}$ and adding entries in the column s.

Non-goal to goal: In this case, we need to reroute transitions which enter s to now point at $\bar{s}$entering s, and also add the probability $\pi_{s',s}(0,t)$ to $\pi_{s',\bar{s}}(0,t)$, afterwards set $\pi_{s',s}(0,t)$ to zero.

Once these bookkeeping operations have been performed, and the new probability matrix $\Pi(0, T_1^+)$ has been computed if needed, we can restart the integration of the forward Kolmogorov equations, with initial conditions given by precisely by $\Pi(0, T_1^+)$. We can then iterate this procedure, until the final time T is reached.

Running Example. We consider the running example, and compute the probability of the nested path formula $\psi = F^{[0,T]}(a_{patched} \wedge P_{\geq p}(G^{[0,T_a]} \neg a_{infected}))$, for $T_a = 10$. The time dependent truth of the inner temporal formula $\varphi = P_{\geq p}(G^{[0,T_a]} \neg a_{infected})$ has already been computed in the previous section (see Figure 7). Furthermore, the formula $a_{patched} \wedge P_{\geq p}(G^{[0,T_1]} \neg a_{infected})$ is false for any state different from p, and equal to $\mathbf{T}(\varphi, p, t)$ for the patched state p. Hence, we state p will be non-goal until time $T_d = 81.8$, and then it will become a goal state.

Thus, when computing the probability of the path formula ψ, we have that no state is goal until time T_d, and after T_d p will be the only goal state. If we look at the path probability of ψ as a function of T (Figure 8), we observe that this probability is zero for $T < T_d$, and then suddenly jumps at time T_d to $\pi_{x,p}(0, T_d)$, for $x \in S$, and keeps on increasing afterwards.

The algorithm to compute the path probability for a general reachability problem, with both unsafe and goal states, is similar to the one sketched above. In general, for an unsafe state s we disable all outgoing transitions, making it absorbing. The only difference resides in the bookkeeping operations that need to be performed when a state s changes its unsafe status.

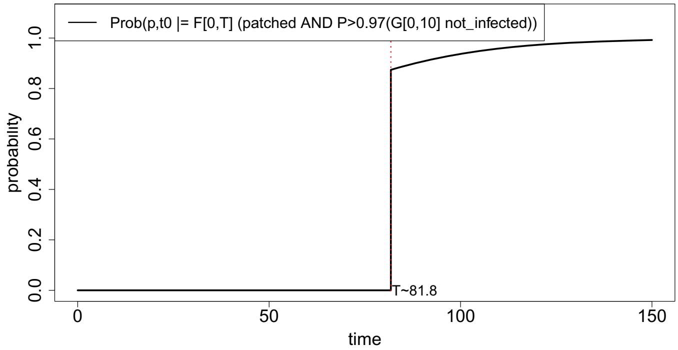


Fig. 8. Fluid estimate of the path probability of the formula $F^{[0,T]}(a_{patched} \wedge P_{\geq 0.97}(G^{[0,10]}\neg a_{infected}))$, as a function of T, starting from state p at time 0 (continuous black line). We can see that the probability is discontinuous, and there is a jump at time $T = 81.8$, corresponding to the time at which the truth value of $P_{\geq 0.97}(G^{[0,10]}\neg a_{infected})$ changes for state p, see Figure 7. Parameters of the model are as in Figure 2.

Unsafe to safe: in this case, we just need to re-enable all the outgoing transitions from s.

Safe to unsafe: In this case, we disable all outgoing transitions from s, but we also discard the probability $\pi_{s',s}(0,t)$, setting it to zero. This is because all trajectories started from s' at time 0 and being in s when it becomes unsafe become trajectories that can no longer reach a goal state avoiding unsafe ones, because they suddenly find themselves in an unsafe state.

A minor caveat when we have both unsafe and goal sets is to decide how a state that is both goal and unsafe behaves. In this case, by the definition of the until semantics, its goal nature will prevail.

Reachability Probabilities as a Function of Time. Up to now, we sketched an algorithm to compute the reachability probability starting from time zero up to time T, call it $\Upsilon(0,T)$. In order to extend the method to compute $\Upsilon(t,t+T)$, as a function of the initial time t, we will follow a similar approach to that of Section 6, finding an expression for $\Upsilon(t,t+T)$ and applying a generalised version of the forward and backward Kolmogorov equations to it, in order to obtain an ODE for Υ.

In this tutorial, we will just present this expression for Υ, which is obtained by combining Chapman Kolmogorov equations [42] with suitable matrices encoding the bookkeeping operations. Further details on the algorithm can be found in [12,13].

The first step is the definition of a matrix ζ encoding the bookkeeping operations. Let $G(t)$ and $U(t)$ be the time-varying goal and unsafe sets, and let $T_1, \ldots, \mathbf{true}_k, \ldots$ be the time instants in which one state changes status. Define the set of safe states $W(t) = S \setminus (G(t) \cup U(t))$. We define the following matrices for each discontinuity time T_i:

- $\zeta_W(T_i)$ is the $n \times n$ matrix, $|S| = n$, equal to 1 only on the diagonal elements corresponding to states s_j belonging to both $W(T_i^-)$ and $W(T_i^+)$ (i.e. states that are safe and not goals both before and after T_i), and equal to 0 elsewhere;
- $\zeta_G(T_i)$ is the $n \times n$ matrix equal to 1 in the diagonal elements corresponding to states s_j belonging to $W(T_i^-) \cap G(T_i^+)$ (safe and non-goal states becoming goal), and zero elsewhere;
- $\zeta(T_i)$ is the $2n \times 2n$ matrix defined by:

$$\zeta(T_i) = \begin{pmatrix} \zeta_W(T_i) & \zeta_G(T_i) \\ 0 & I \end{pmatrix}.$$

Now, assume in $[t_1, t_2]$ no discontinuity occurs, so that the Q-matrix of the modified CTMC does not change structure in $[t_1, t_2]$, and let $\tilde{\Pi}(t_1, t_2)$ be the probability matrix computed by solving the forward Kolmogorov equations, with $\tilde{\Pi}(t_1, t_1) = I$. Then, recalling that the Chapman Kolmogorov equations state that $\tilde{\Pi}(t_1, t_3) = \tilde{\Pi}(t_1, t_2)\tilde{\Pi}(t_2, t_3)$, we can compute Υ as follows [12,13]:

$$\Upsilon(t, t+T) = \tilde{\Pi}(t, T_1)\zeta(T_1)\tilde{\Pi}(T_1, T_2)\zeta(T_2)\cdots\zeta(T_{k_I})\tilde{\Pi}(T_{k_I}, t+T), \tag{9}$$

where $T_1, \ldots, T_{k_I}$ are all the discontinuity points between t and $t+T$. From this equation, observing it depends on t only in the first and last factor and using the backward and forward Kolmogorov equations, we can derive an ODE similar to equation (8), with Π replaced by Υ:

$$\frac{d\Upsilon(t, t+T)}{dt} = -\tilde{Q}(t)\Upsilon(t, t+T) + \Upsilon(t, t+T)\tilde{Q}(t+T), \tag{10}$$

where $\tilde{Q}(t)$ is the modified Q-matrix according to the goal and unsafe sets at time t. See [12,13] for further details, and for a sketch of the algorithms that can be used to integrate the so-obtained equation.

Steady State Properties. In this tutorial, like in [12,13], we have considered only time bounded operators. This limitation is a consequence of the very nature of the Theorem 2, which holds only for a finite time horizon. However, there are situations in which we can extend the validity of the theorem to the whole time domain, but this extension depends on properties of the phase space of the fluid ODE [10,14]. In those cases, we can prove convergence of the steady state behaviour of $Z^{(N)}$ to that of z, hence we can incorporate also operators dealing with steady state properties.

Checking these properties is relatively simple: we need to compute the unique fixed point $\mathbf{x}^*$ of the fluid ODE, which will be also the steady state measure of the

Table 1. Comparison of running times of the Fluid Model Checking algorithm of some properties discussed in the paper, with the running time for their statistical estimate (statistical model checking), for different population levels, computed from 10000 runs

Checked property	Fluid MC	Stat MC ($N = 1000$)	Stat MC ($N = 100$)
Kolmogorov Equations	~ 0.1 sec	~ 64 sec	~ 101 sec
$\mathbf{X}^{[0,T]} a_{infected}$	~ 0.06 sec	~ 6 sec	~ 24 sec
$\neg a_{infected} \mathbf{U}^{[0,T]} a_{patched}$	~ 0.05 sec	~ 5 sec	~ 20 sec

limit fluid agent $z(t)$, assuming it is irreducible. When at steady state, the rates of $z(t)$ do not depend on time anymore, hence it becomes a time-homogeneous CTMC. Therefore, to model check a formula like $S_{\bowtie}(\varphi)$, we just need to model check φ against this time-homogeneous CTMC, with standard algorithms, and then compute the satisfaction of the steady state operator as in the standard model checking for CTMC [5] (see also [32]).

Running Example. As an example, consider the steady state property $S_{\geq 0.75}(P_{\leq 0.1}(F^{[0,10]} a_{infected}))$. The fluid ODE for our example has a unique, globally attracting, fixed point $\mathbf{x}^* = (0.0209, 0.0767, 0.0383, 0.8641)$. Substituting this into the time-dependent Q matrix of the fluid agent z we obtain a time-homogeneous CTMC, for which the probability of $F^{[0,10]} a_{infected})$ is $(0.8526, 1, 1, 0.0276)$ and so the formula $P_{\leq 0.1}(F^{[0,10]} a_{infected})$ is true in state p and false in states s, d, i. By using ergodicity of z and the fact that $\mathbf{x}^*$ is also the limit steady state measure of an individual agent, hence the steady state measure of z, we can compute the steady state probability of satisfying $P_{\leq 0.1}(F^{[0,10]} a_{infected})$ as $0 * 0.0209 + 0 * 0.0767 + 0 * 0.0383 + 1 * 0.8641 = 0.8641$, which makes the steady state property true in all states.

8 Decidability and Convergence

In this section we will briefly discuss some theoretical features of the approximate model checking algorithm presented.

We will consider two main issues related to decidability and accuracy. Firstly, we will discuss the decidability of the algorithm to model check CSL specifications against ICTMC models. The fluid approximation of single agent properties is based on this as we have shown, and it is important to assess that this algorithm will yield an answer. Secondly, we must also consider the relationship between the truth of a CSL formula with a single agent derived through consideration of the fluid limit (i.e. representing the rest of the population only through the mean field) and the truth of the CSL formula for the single agent in a finite population model (i.e. with all agents represented explicitly). The approach is only useful if this relationship is close to identity.

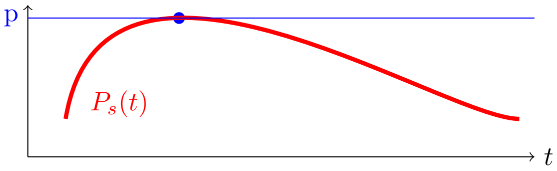
Efficiency and Decidability. Two desirable features of any model checking algorithm are decidability and computational efficiency. Efficiency, in particular, was the practical motivation of this work. Indeed, as can be seen in Table 1, already for the simple running example, the gain in computational time is remarkable: the fluid model checking approach is between 500 and 1000 times faster than statistical model checking for a population of 1000 nodes, with a negligible loss in accuracy. Furthermore, its complexity is independent of the population size, so that it can scale to very large systems.

The issue about decidability, instead, is more delicate, and depends heavily on the rate functions of the collective model and on the solution of the fluid ODE. In particular, the problem is with the nesting of CSL formulae. In order to model check a next or an until formula, containing nested temporal operators, we need to be able of performing a certain set of operations, specifically:

1. compute the set of zeros of $P(\psi, s, t) - p$ as a function of the time at which the formula $\varphi = P_{\bowtie p}(\psi)$ is evaluated;
2. check whether $P(\psi, s, t) < p$, $P(\psi, s, t) = p$, or $P(\psi, s, t) > p$ (to compute the truth function $\mathbf{T}(\varphi, s, t)$ for $\varphi = P_{\bowtie p}(\psi)$);
3. store in memory the truth function $\mathbf{T}(\varphi, s, t)$;

In order to deal with point 3 above, we need to guarantee that the number of zeros of the function $P(\psi, s, t) - p$ is finite in any finite time interval $[0, T]$. This is not true in general, but can be enforced by imposing additional regularity assumptions on the rate functions of the population model. Specifically, in [12,13] we restricted the rate functions of the collective model to be (piecewise) *real analytic functions* [33]. This class of functions has nice closure properties (they are closed for arithmetic operations, integration, differentiation, and so on), which guarantees that all probability functions $P_s(t)$ will remain (piecewise) analytic. Furthermore, they are reasonably general, including most of the functions used in practice (for instance, polynomials, exponentials, and so on). Finally, they enjoy the property that either they are identically zero, or have only a finite number of zeros in any finite time interval. This settles point 3.

Points 1 and 2 above, instead, are much more delicate. First of all, finding all zeros of an analytic function is not an easy task, and in general may not be decidable. In particular, finding simple zeros (those around which a function change sign) is decidable (using interval methods [2,41]), but we may not be able to find non-simple zeros, i.e. those in which the derivative is null, like local maxima or minima, see figure below.



Also the decidability of point 2 above, called the zero test problem, is unknown for analytic functions (in fact, its decidability is not known even for functions constructed using polynomials and the exponential, see [44]).

The way out this problem is to characterise precisely all the situations in which something bad can happen, and show that this are sufficiently *rare*. More precisely, we fixed a formula structure and the model parameters, and looked at what happens in terms of the thresholds p of the probability operators $P_{\bowtie p}$. If we have a formula with k temporal operators, than we have k such thresholds, which can take values in the hypercube $[0,1]^k$. We then looked at the subset R of points of $[0,1]^k$ for which we can guarantee that the algorithm terminates, and characterised it from a topological viewpoint. It turns out [12,13] that R is an *open* subset of *Lebesgue measure one* of $[0,1]^k$. This means that almost any formula will be decidable, and furthermore that decidability is robust with respect to small perturbations of the probability thresholds. In [12,13] this is termed *quasi-decidability*, and the CSL formulae which have thresholds probabilities that belong to the set R are called *robust*.

Convergence. We also investigated the limit behaviour of path probabilities and truth values of CSL formulae, evaluated for an individual agent $Z^{(N)}(t)$ in a finite population model, in the limit of $N \to \infty$. We proved that, in almost all cases, they converge to path probabilities and truth values computed for the limit individual agent $z(t)$. Convergence, however, does not hold always; it can fail exactly in those situations in which the limit model checking problem is not decidable. Given a CSL formula $\varphi = \varphi(\mathbf{p})$, with probability threshold arranged in a vector $\mathbf{p}$, we characterised the subset of $[0,1]^k$ of threshold for which convergence surely holds, obtaining that it coincides with the set R of thresholds making φ robust. More precisely, we have proved the following theorem [12,13]:

Theorem 3. *Let $\mathcal{X}^{(N)}$ be a sequence of CTMC models and let $Z^{(N)}(t)$ and $z(t)$ be defined from $\mathcal{X}^{(N)}$ as in Section 4.2. Assume that $Z^{(N)}(t)$, $z(t)$ have piecewise analytic infinitesimal generator matrices.*
Let $\varphi(p_1,\ldots,p_k)$ be a robust CSL formula. Then, there exists an N_0 such that, for $N \geq N_0$ and each $s \in \mathcal{S}$

$$s, 0 \models_{Z^{(N)}} \varphi \Leftrightarrow s, 0 \models_z \varphi.$$

This theorem states that, for a given robust CSL formula φ, we can find an index N_0 such that, for populations larger than N_0, φ will hold in the limit model if and only if it holds in a model with population N. This shows that the method presented here is consistent with respect to asymptotic approximation. Unfortunately, characterising such N_0 is extremely difficult, see also the discussion in [12,13] about erorr bounds.

9 Related Work

As this is a very new direction of research there is, as yet, only a small amount of related work. Model checking (time homogeneous) Continuous Time Markov Chains (CTMC) against Continuous Stochastic Logics (CSL) specifications has a long tradition in computer science [5,4,45]. At the core of our approach to study time-bounded properties there are similarities to that developed in [5], because we consider a transient analysis of a Markov chain whose structure has been modified to reflect the formula under consideration. But the technical details of the transient analysis, and even the structural modification, differ to reflect the time-inhomogeneous nature of the process we are studying.

To the best of the authors' knowledge, there has been no previous proposal of an algorithm to model check CSL formulae on a ICTMC. Nevertheless model checking of ICTMCs has been considered with respect to other logics. Specifically, previous work includes model checking of HML and LTL logics on ICTMC.

In [30], Katoen and Mereacre propose a model checking algorithm for Hennessy-Milner Logic on ICTMC. Their work is based on the assumption of piecewise constant rates (with a finite number of pieces) within the ICTMC. The model checking algorithm is based on the computation of integrals and the solution of algebraic equations with exponentials (for which a bound on the number of zeros can be found).

LTL model checking for ICTMC, instead, has been proposed by Chen *et al.* in [18]. The approach works for time-unbounded formulae by constructing the product of the CTMC with a generalized Büchi automaton constructed from the LTL formula, and then reducing the model checking problem to computation of reachability of bottom strongly connected components in this larger (pseudo)-CTMC. The authors also propose an algorithm for solving time bounded reachability similar to the one considered in this paper (for time-constant sets).

Our work is underpinned by the notion of fast simulation, which has previously been applied in a number of different contexts [22]. One recent case is a study of policies to balance the load between servers in large-scale clusters of heterogeneous processors [24]. These ideas also underlie the work of Hayden *et al.* in [25]. Here the authors extend the consideration of transient characteristics as captured by the fluid approximation, to approximation of first passage times, in the context of models generated from the stochastic process algebra PEPA. Their approach for passage times of individual components is closely related to the fast simulation result and the work presented in this paper. The main difference is that they consider just path properties, described by deterministic automata (formally treated in [26]), which they solve by integrating ODEs.

10 Conclusions

In this tutorial we presented a new method to approximatively model check properties of individual agents in a large population, exploiting mean field theory. This theory predicts that in the limit of an infinite population individual agents will decouple, evolving as independent CTMC connected only through the mean state of the system, described by the fluid ODE. This independence frees us from the necessity of representing the whole state space of the population, and instead we need only represent the state space of the individual agent. However, since the behaviour of this agent depends on the mean state of the system, its transition rates are not constants, but instead vary with time. Thus, in order to check properties for this limit model, we need to deal with a time-inhomogeneous CTMC. In this chapter we have presented a method to model check CSL formulae against ICTMC, whose complexity stems from the time dependency of truth values of temporal sub-formulae.

Our objective here has been to introduce the main ideas in an informal manner, explaining them by means of a simple example of a peer-to-peer network epidemic, in order to give the reader an intuition of how the approach works. The reader interested in the formal details is invited to study the fuller account given in the recent CONCUR paper [12] or its extended version [13].

The development of a fluid approximation for model checking, albeit only currently for time-bounded properties of individual agents opens the possibility of carrying out model checking on a wide range of population models that were previously extremely computationally costly or even beyond the scope of existing tools. Moreover there is a lot of potential of expanding the reach of model checking still further. Currently, we are extending the approach in several directions, including:

- moving beyond CSL to consider more complex path properties, for instance those expressed by Deterministic Timed Automata [19] (DTA), obtaining a logic for individual properties similar to asCSL [6] and CSL-TA [23];
- the lifting of individual specifications to the collective level, similarly to [32]. In this paper, the authors consider atomic collective properties stating that the expected fraction of agents satisfying a local CSL property meets a given bound $\bowtie p$. Instead of the expectation, we are considering approximations of the probability that the fraction of agents satisfying a local CSL property meets a given bound $\bowtie p$, using higher order fluid approximations, like the functional central limit [35] or linear noise approximation [50].

References

1. GNU Octave
2. Alefeld, G., Mayer, G.: Interval analysis: theory and applications. Journal of Computational and Applied Mathematics 121, 421–464 (2000)
3. Andreychenko, A., Crouzen, P., Wolf, V.: On-the-fly uniformization of time-inhomogeneous infinite Markov population models. In: Proceedings Ninth Workshop on Quantitative Aspects of Programming Languages, QAPL 2011. EPTCS, vol. 57, p. 1 (2011)
4. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds.) CAV 1996. LNCS, vol. 1102, pp. 269–276. Springer, Heidelberg (1996)
5. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model checking continuous-time Markov chains by transient analysis. In: Emerson, E.A., Sistla, A.P. (eds.) CAV 2000. LNCS, vol. 1855, pp. 358–372. Springer, Heidelberg (2000)
6. Baier, C., Cloth, L., Haverkort, B.R., Kuntz, M., Siegle, M.: Model checking Markov chains with actions and state labels. IEEE Trans. Software Eng. 33(4), 209–224 (2007)

7. Bakhshi, R., Cloth, L., Fokkink, W., Haverkort, B.R.: Mean-field analysis for the evaluation of gossip protocols. In: Proceedings of the Sixth International Conference on the Quantitative Evaluation of Systems, QEST 2009, pp. 247–256. IEEE Computer Society (2009)
8. Bakhshi, R., Cloth, L., Fokkink, W., Haverkort, B.R.: Mean-field framework for performance evaluation of push-pull gossip protocols. Perform. Eval. 68(2), 157–179 (2011)
9. Benaïm, M., Le Boudec, J.: A class of mean field interaction models for computer and communication systems. Performance Evaluation (2008)
10. Benaïm, M., Le Boudec, J.Y.: On mean field convergence and stationary regime. CoRR, abs/1111.5710 (2011)
11. Berec, L.: Techniques of spatially explicit individual-based models: construction, simulation, and mean-field analysis. Ecological Modelling 150(1-2), 55–81 (2002)
12. Bortolussi, L., Hillston, J.: Fluid model checking. In: Koutny, M., Ulidowski, I. (eds.) CONCUR 2012. LNCS, vol. 7454, pp. 333–347. Springer, Heidelberg (2012)
13. Bortolussi, L., Hillston, J.: Fluid model checking. CoRR, 1203.0920 (2012)
14. Bortolussi, L., Hillston, J., Latella, D., Massink, M.: Continuous approximation of collective systems behaviour: a tutorial. Performance Evaluation (2013)
15. Bortolussi, L.: On the approximation of stochastic concurrent constraint programming by master equation, vol. 220, pp. 163–180 (2008)
16. Bortolussi, L., Policriti, A.: Dynamical systems and stochastic programming: To ordinary differential equations and back. In: Priami, C., Back, R.-J., Petre, I. (eds.) Transactions on Computational Systems Biology XI. LNCS, vol. 5750, pp. 216–267. Springer, Heidelberg (2009)
17. Burden, R.L., Faires, J.D.: Numerical analysis. Thomson Brooks/Cole (2005)
18. Chen, T., Han, T., Katoen, J.-P., Mereacre, A.: LTL model checking of time-inhomogeneous Markov chains. In: Liu, Z., Ravn, A.P. (eds.) ATVA 2009. LNCS, vol. 5799, pp. 104–119. Springer, Heidelberg (2009)
19. Chen, T., Han, T., Katoen, J.P., Mereacre, A.: Model checking of continuous-time Markov chains against timed automata specifications. Logical Methods in Computer Science 7(1) (2011)
20. Clarke, E., Peled, A., Grunberg, A.: Model Checking. MIT Press (1999)
21. Darling, R.W.R.: Fluid limits of pure jump Markov processes: A practical guide (2002), `http://arXiv.org`
22. Darling, R.W.R., Norris, J.R.: Differential equation approximations for Markov chains. Probability Surveys 5 (2008)
23. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Trans. Software Eng. 35(2), 224–240 (2009)
24. Gast, N., Gaujal, B.: A mean field model of work stealing in large-scale systems. In: Proceedings of ACM SIGMETRICS 2010, pp. 13–24 (2010)
25. Hayden, R.A., Stefanek, A., Bradley, J.T.: Fluid computation of passage-time distributions in large Markov models. Theor. Comput. Sci. 413(1), 106–141 (2012)
26. Hayden, R.A., Bradley, J.T., Clark, A.: Performance specification and evaluation with unified stochastic probes and fluid analysis. IEEE Trans. Software Eng. 39(1), 97–118 (2013)
27. Hillston, J.: Fluid flow approximation of PEPA models. In: Proceedings of the Second International Conference on the Quantitative Evaluation of SysTems, QEST 2005, pp. 33–42 (September 2005)
28. Jensen, A.: Markov chains as an aid in the study of Markov processes. Skandinavisk Aktuarietidskriff 36 (1953)

29. Jha, S.K., Clarke, E.M., Langmead, C.J., Legay, A., Platzer, A., Zuliani, P.: A Bayesian approach to model checking biological systems. In: Degano, P., Gorrieri, R. (eds.) CMSB 2009. LNCS, vol. 5688, pp. 218–234. Springer, Heidelberg (2009)
30. Katoen, J.-P., Mereacre, A.: Model checking HML on piecewise-constant inhomogeneous Markov chains. In: Cassez, F., Jard, C. (eds.) FORMATS 2008. LNCS, vol. 5215, pp. 203–217. Springer, Heidelberg (2008)
31. Kolesnichenko, A., Remke, A., de Boer, P.-T., Haverkort, B.R.: Comparison of the mean-field approach and simulation in a peer-to-peer botnet case study. In: Thomas, N. (ed.) EPEW 2011. LNCS, vol. 6977, pp. 133–147. Springer, Heidelberg (2011)
32. Kolesnichenko, A., Remke, A., de Boer, P.-T., Haverkort, B.R.: A logic for model-checking of mean-field models. In: Proceedings of the 43rd International Conference on Dependable Systems and Networks, DSN 2013 (2013)
33. Krantz, S., Harold, P.R.: A Primer of Real Analytic Functions, 2nd edn. Birkhäuser (2002)
34. Kurtz, T.G.: Solutions of ordinary differential equations as limits of pure jump Markov processes. Journal of Applied Probability 7, 49–58 (1970)
35. Kurtz, T.G.: Approximation of population processes. SIAM (1981)
36. Kwiatkowska, M., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: A hybrid approach. International Journal on Software Tools for Technology Transfer 6(2), 128–142 (2004)
37. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
38. Le Boudec, J.-Y.: Performance Evaluation of Computer and Communication Systems. EPFL Press, Lausanne (2010)
39. Massink, M., Latella, D., Bracciali, A., Harrison, M., Hillston, J.: Scalable context-dependent analysis of emergency egress models. Formal Aspects of Computing 24(2), 267–302 (2012)
40. MATLAB: v. 7.10.0 (R2010a). The MathWorks Inc., Natick, Massachusetts (2010)
41. Neumaier, A.: Interval Methods for Systems of Equations. University Press, Cambridge (1990)
42. Norris, J.R.: Markov Chains. Cambridge University Press (1997)
43. Qian, H., Elson, E.L.: Single-molecule enzymology: stochastic michaelis-menten kinetics. Biophysical Chemistry 101, 565–576 (2002)
44. Richardson, D.: Zero tests for constants in simple scientific computation. Mathematics in Computer Science 1(1), 21–37 (2007)
45. Rutten, J., Kwiatkowska, M., Norman, G., Parker, D.: Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems. CRM Monograph Series, vol. 23. American Mathematical Society (2004)
46. Sumpter, D.T.J.: From Bee to Society: An Agent-based Investigation of Honey Bee Colonies. PhD thesis, University of Manchester (2000)
47. Szallasi, Z., Stelling, J., Periwal, V. (eds.): System Modeling in Cellular Biology, From Concepts to Nuts and Bolts. MIT Press (2012)
48. Tribastone, M., Ding, J., Gilmore, S., Hillston, J.: Fluid rewards for a stochastic process algebra. IEEE Trans. Software Eng. 38(4), 861–874 (2012)
49. Tribastone, M., Gilmore, S., Hillston, J.: Scalable differential analysis of process algebra models. IEEE Trans. Software Eng. 38(1), 205–219 (2012)
50. Van Kampen, N.G.: Stochastic Processes in Physics and Chemistry. Elsevier (1992)

51. van Moorsel, A.P.A., Wolter, K.: Numerical solution of non-homogeneous Markov processes through uniformization. In: Proceedings of the 12th European Simulation Multiconference - Simulation- Past, Present and Future, ESM 1998, pp. 710–717. SCS Europe (1998)

A Integrating the Combined Backward-forward Kolmogorov Equation

In this appendix we will look more closely at the problem of numerically integrating the combined backward-forward Kolmogorov equation (8), needed to compute the time-dependent reachability probability for until formulae, see Section 6. Integrating this equation is necessary to check nested formulae. A result of this integration, for the running example, has been shown in Figure 7. We recall that the equation is

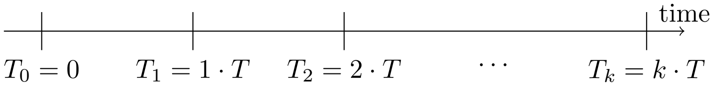
$$\frac{d\Pi(t,t+T)}{dt} = -Q(t)\Pi(t,t+T) + \Pi(t,t+T)Q(t+T),$$

which has to be solved from time t_0 to time t_1, with initial conditions $\Pi(t + 0, t_0 + T)$ computed using the forward equation.

In principle, this could be done by using one of the many ODE solvers available, e.g. those of Matlab$^{\mathrm{TM}}$ [40] or Octave [1]. Practically, using one of those solvers, we have observed that in most of the cases, we obtain a plot like the one shown in Figure 9, in which the numerical error explodes. This is an indicator that equation (8) is, in general, very stiff [17], hence a stiff integration method has to be used. Unfortunately, this blow up phenomenon persisted even using the most accurate stiff integrators of Matlab$^{\mathrm{TM}}$ or Octave, even with very high accuracy. We only obtained a reduction in the blow up speed.

In order to compute the trajectory in Figure 7, therefore, we need a different strategy. We present the idea in the following, showing how to compute the time-dependent reachability probability with time horizon T, without resorting to equation (8). The idea is to exploit the Chapman-Kolmogorov (CK) semigroup equations [42], $\Pi(t,t') = \Pi(t,t'')\Pi(t'',t')$, $t'' \in [t,t']$, in order to integrate the backward and the forward equations separately. The advantage of this is that the forward and the backward equations alone are, in general, quite stable.

For simplicity, we assume in the following that $t_0 = 0$ and $t_1 = k \cdot T$. The first operation is to split the time interval $[0, kT]$ into smaller time intervals, each of length T, as shown in the figure below:



Call T_j the time instant $j \cdot T$, fix $j \geq 1$, and pick $t \in [T_{j-1}, T_j]$. Applying the CK to times $t, T_j, t + T$, we get

$$\Pi(t,t+T) = \Pi(t,T_j)\Pi(T_j,t+T).$$

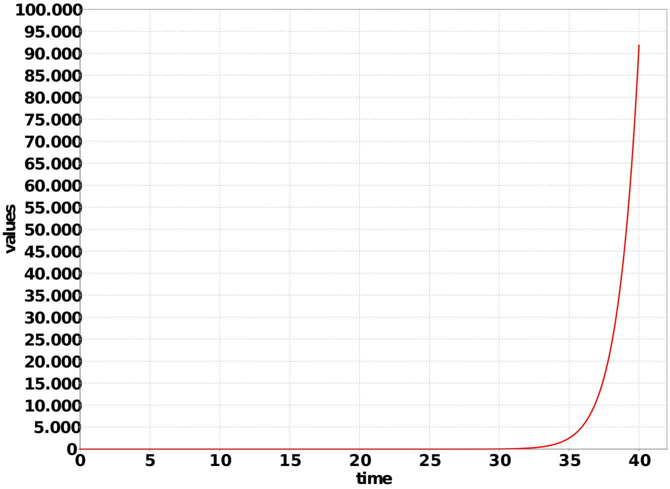


Fig. 9. Integration with a ODE numerical solver of the equation (8) for the formula $G^{[0,T_2]}\neg a_{infected}$, for $T_2 = 10$. Due to the high degree of stiffness of the equation and numerical instabilities, the error blows up.

As T_j is a constant, in order to compute $\Pi(t, t+T)$ for $t \in [T_{j-1}, T_j]$, we can integrate separately the backward equation for $\Pi(t, T_j)$, $t \in [T_{j-1}, T_j]$, with initial value $\Pi(T_{j-1}, T_j)$, and the forward equation for $\Pi(T_j, t')$, $t' \in [T_j, T_{j+1}]$, with initial value the identity matrix. These two equations can be solved simultaneously. Then, we can take the product of the so obtained matrices to compute $\Pi(t, t+T)$.

The full algorithm is a simple loop over j, observing that the initial conditions needed to integrate the backward equation are the last point computed by the forward equation in the previous iteration.

Practically, if we want just to visualize the result, we need to compute the product of $\Pi(t, T_j)$ and $\Pi(T_j, t+T)$ only at the sampled points of the function, generally a fixed grid of stepsize h. If, instead, we want to solve the equation $P(t) - p = 0$, in order to obtain the truth value, we need to take the product of the two matrices every time the root finding function (usually embedded in the ODE solver) needs to know the value of the function $P(t) - p$.

Topological Quantum Computation

Jiannis K. Pachos

School of Physics and Astronomy, Univesrity of Leeds, Leeds LS2 9JT, UK
j.k.pachos@leeds.ac.uk

Abstract. This pedagogical introduction to topological quantum computation includes the following parts. First we provide an introduction to anyons and topological models. In particular we consider the properties of anyons and their relation to topological quantum computation. Then we present the quantum double models. These are stabiliser codes, that can be described very much like quantum error correcting codes. They include the toric code and various Abelian and non-Abelian extensions. Next the Jones polynomials are presented, which are topological invariants of links and knots that are related to anyons. Their evaluations by classical algorithms is computationally complex, but their approximation by quantum algorithms is efficient. Finally, we presenter an overview of the current state of topological quantum computation and present some open questions.

1 Introduction

1.1 Statistics for Quantum Computation

Physics should remain unchanged if we exchange two identical particles. This is a fundamental symmetry with far reaching consequences. In three dimensions it dictates the existence of bosons and fermions. Their wave function acquires a $+1$ or a -1 phase, respectively, whenever two particles are exchanged. In one dimension the exchange of particles causes them inevitably to collide. When one considers two dimensions, a variety of statistical behaviours is possible. Apart from bosonic and fermionic behaviours, arbitrary phase factors, or even non-trivial unitary evolutions, can be obtained when two particles are exchanged [1]. Particles with such an exotic statistics are called anyons.

The study of anyons started as a theoretical construction of two dimensional models [2]. It was soon realised that they can be encountered in physical systems with effective two dimensional behaviour. For example, confined gases of electrons in two dimensions in the presence of sufficiently strong magnetic field and low temperatures give rise to the Fractional Quantum Hall Effect [3–5]. The low energy excitations of these systems are localised quasiparticle excitations that can actually exhibit anyonic statistics. Alternatively, one can engineer two dimensional spin lattice models with quasiparticles that exhibit anyonic statistics.

Systems that support anyons are called topological as their properties depend on global characteristics and not on local details. They have highly entangled

M. Bernardo et al. (Eds.): SFM 2013, LNCS 7938, pp. 150–179, 2013.

degenerate ground states that give rise to their exotic behaviour. The order parameters that detect the topological phases of systems are non-local, in contrast e.g. to magnetisation. Various smoking guns exist for topological order, such as ground state degeneracy, topological entropy or the explicit detection of anyons. As topological order comes in various forms [6], the study and characterisation of topological systems in their generality is complex and still an open problem.

Quantum computation requires the encoding of quantum information and its manipulation with quantum gates [7]. Qubits, the quantum version of classical bits, provide a two dimensional Hilbert space. Quantum gates are necessary to manipulate information and to perform a computation. A universal quantum computer employs a sufficiently large set of gates in order to perform arbitrary quantum algorithms. In recent years, there have been two main quests for quantum computation: to find new algorithms, i.e. beyond searching [8] and factorizing [9], and to perform fault-tolerant evolutions.

There have been several proposals of quantum computation that are conceptually different, but equivalent to the circuit model. One way quantum computation [10] starts from a large entangled state. Information is processed by single qubit measurements, which contrasts the popular belief that quantum computation must be reversible. Adiabatic quantum computation is another way of processing information [11]. There, the answer to the problem is encoded into the unique ground state of a Hamiltonian. Then an adiabatic evolution of a simple starting Hamiltonian with a known ground state is considered. The ground state of the final Hamiltonian is a bit string encoding the answer to a problem.

In the nineties a surprising connection was made. It was argued that anyons could be employed to perform quantum computation [12]. Kitaev [13] demonstrated that anyons could actually be used to perform fault-tolerant quantum computation. This was a very welcomed advance as errors are present in any physical realisation of quantum computation, coming from the environment or from control imperfections. Shor [14] and Steane [15] independently demonstrated that for sufficiently isolated quantum systems and for sufficiently precise quantum gates, quantum error correction can allow fault-tolerant computation. However, the required limits are too stringent and demand a large overhead in qubits and quantum gates. In contrast to this, anyonic quantum computation promises to resolve the problem of errors from the hardware level.

Topological systems can serve as quantum memories and as quantum computers. One can encode quantum information in simple topological systems in such a way that it is shielded from environmental perturbations. This is an important property for constructing quantum hard disks. Complex enough topological systems can realise quantum computation. They can manipulate information with very accurate quantum gates, while keeping the information protected at all times. In these systems, information is encoded in the possible outcomes when bringing two anyons together. The exchange of anyons gives rise to statistical logical gates. Fundamental properties of the quasiparticles can thus become the means to perform quantum computation. Fault-tolerance stems from the ability to keep the quasiparticles intact. The result is a surprisingly effective and aesthetically appealing method for performing fault-tolerant quantum computation.

1.2 Anyons Anyone?

Braiding in Three and Two Dimensions. It is commonly accepted that point-like particles, elementary or not, come in two species: bosons or fermions. These statistical behaviours can be obtained by circulating a particle around an identical one and observing the topological characteristics of their evolution. In three dimensions this evolution spans a path γ_1 that can be continuously deformed to γ_2, as seen in Figure 1.1, and then to a trivial path. As a consequence the wave function, $|\Psi(\gamma_1)\rangle$, of the system after the circulation has to be exactly the same as the original one $|\Psi(0)\rangle$, i.e.

$$|\Psi(\gamma_1)\rangle = |\Psi(\gamma_2)\rangle = |\Psi(0)\rangle \,. \tag{1}$$

It is easily seen that a full circulation is effectively equivalent to two successive particle exchanges. Thus, a single exchange can result in a phase factor $e^{i\varphi}$ that has to square to unity in order to be consistent with (1), giving, finally, $\varphi = 0, \pi$. These two cases correspond to the bosonic and fermionic statistics, respectively.

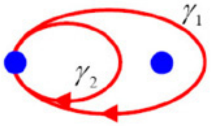


Fig. 1. A particle spans a loop around another one. In three dimensions it is possible to continuously deform the path γ_1 to the path γ_2 which is equivalent to a trivial path.

When we restrict ourselves to two spatial dimensions, then there are more possibilities in statistical behaviours. If the particle circulation γ_1 of Figure 1 is performed on a plane, then it is not possible to continuously deform it to the path γ_2. Still the evolution that corresponds to γ_2 is equivalent to the trivial evolution as seen in Figure 2. As we are not able to deform the evolution of path γ_1 to the trivial one the above argument does not apply. Actually, it is possible to assign an arbitrary phase factor, or even a whole unitary, to this evolution. Thus, particles in two dimensions can have richer statistical behaviours, possibly different from bosons or fermions.

Aharonov-Bohm Effect and Berry Phases. The statistical phases of anyons can be viewed as Aharonov-Bohm phases or as Berry phases. While these descriptions might inspire some readers they are not fundamental in understanding the properties of anyons. To visualise the behaviour of anyons one should think

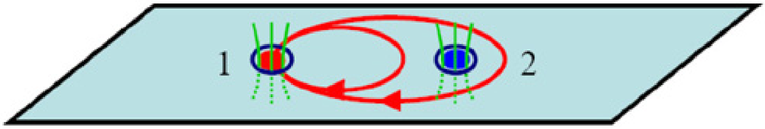


Fig. 2. In two dimensions the two paths γ_1 and γ_2 are topologically distinct. This gives the possibility of having non-trivial phase factors appearing when one particle circulates the other. This can be visualized by having the particles carrying electric charge as well as magnetic flux giving rise to the Aharonov-Bohm effect.

of them as being composite particles consisting of a magnetic flux $\varPhi$ and a ring of electric charge q, as depicted in Figure 2. If particle 1 circulates particle 2, then its charge q goes around the flux $\varPhi$, thereby acquiring a phase factor $U = e^{iq\varPhi}$. This is known as the Aharonov-Bohm Effect [16], which gives rise to the magnetic field $\mathbf{B} = \nabla \times \mathbf{A}$ and the corresponding flux, $\varPhi$, through

$$\varPhi = \oint_\gamma \mathbf{A} \cdot d\mathbf{l} = \iint_S \nabla \times \mathbf{A} \cdot d\mathbf{s} = \iint_S \mathbf{B} \cdot d\mathbf{s}. \tag{2}$$

Here γ is the spanned looping path, $d\mathbf{l}$ is an elementary segment of the path, S is a surface enclosed by γ and $d\mathbf{s}$ is its element. Even though the vector field, $\mathbf{A}$, can be non-zero on the whole plane, the magnetic flux is confined at the neighbourhood of the anyon 2. Hence the resulting phase factor $U = e^{iq\varPhi}$ does not depend on the details of the path of particle 1. It depends only on the number of times it circulates around it. Hence, it is topological in nature and it can faithfully describe the mutual statistics of the particles. The statistical angle of these anyons is $\varphi = q\varPhi/2$.

Non-Abelian charges and fluxes generate unitary matrices U instead of phase factors. In this case a circulation of one anyon around another can lead to its final state being in superposition. The anyons that have such statistics are called non-Abelian, while anyons that obtain a simple phase factor are called Abelian. In reality the presence of charge and flux come from an effective, emerging gauge theory that describes the low energy behaviour of the model [17].

The connection between the physical system and the effective gauge theory is best formulated in terms of the geometrical Berry phase [18, 19]. To define the Berry phase consider a Hamiltonian $H(t)$ that changes in time through a set of time dependent parameters λ^a with $a = 1, ..., n$. For simplicity, we initiate the system in its ground state with energy $E_0 = 0$ and assume there is a finite energy gap $\varDelta E$ separating it from the excited states. If one changes the parameters slowly in time compared to the energy gap $\varDelta E$ then the evolution is adiabatic, causing no population transfer to the excited states. If there are many degenerate ground states,$\{|\,\psi^\alpha\rangle\,, \alpha = 1, ..., n\}$ then the spanning of a loop γ in the parameters λ results in a non-Abelian Berry phase given by

$$\varGamma_A(\gamma) = \mathbf{P} \exp \oint_\gamma \mathbf{A} \cdot d\lambda, \tag{3}$$

where $\mathbf{P}$ denotes path ordering. The resulting evolution is an element of $U(n)$ evolving the ground state of the system in the following way $\Psi(\gamma) = \Gamma_A(\gamma)\Psi(0)$. The connection is a matrix with components given by

$$(A_\mu)^{\alpha\beta} = \langle\psi^\alpha(\lambda)|\frac{\partial}{\partial\lambda^\mu}|\psi^\beta(\lambda)\rangle \tag{4}$$

This is the non-Abelian generalisation of the usual Berry phase which was first presented by Wilczek and Zee [20] and reduces to the usual Berry phase for non-degenerate ground states.

Topological systems are many-body systems with localised quasiparticle excitations. The control parameters λ^μ of topological systems are identified with the coordinates of the quasiparticle. When these quasiparticles are braided, their evolution can be described by a geometrical phase that is independent of the shape of the path, thus simulating the Aharonov-Bohm effect. This topological characteristic makes the corresponding evolutions to be representations of the braid group. It has been explicitly demonstrated by Arovas, Schrieffer and Wilczek [21], how the statistics of Abelian anyons, appearing in the Fractional Quantum Hall Effect, can be expressed as such a Berry phase. The non-Abelian statistics from Berry phases was considered by Read [22] and Lahtinen and Pachos [23].

1.3 Fusion and Braiding Properties of Anyons

As the statistical properties dominate the behaviour of the anyonic states, it is convenient to employ the world lines of the particles to keep track of their positions (see Figure 3). We assume that we can trap and move the anyons around the plane leading to world lines in $2+1$ dimensions. Exchanges of the anyons can be easily described by just braiding their world lines. We can also depict the pair creation of anyons from the vacuum as well as their fusion when they are brought together. The fusion gives new anyons that correspond to the possible outcomes when the original anyons are combined together.

To illustrate these properties let us consider the Ising anyonic model with particle types 1 (vacuum), σ (non-Abelian anyon) and ψ (fermion). These can be thought of as conserved quantum numbers of the quasiparticles. The conservation of these quantum numbers is given by the following fusion rules

$$\sigma \times \sigma = 1 + \psi, \quad \sigma \times \psi = \sigma, \quad \psi \times \psi = 1,$$

with 1 fusing trivially with the rest of the particles ($\sigma \times 1 = \sigma$ and $\psi \times 1 = \psi$). The first fusion rule signifies that if we bring two σ anyons together they might annihilate (i.e. σ is its own antiparticle) or they can give rise to the fermion ψ depending on their total state. Hence, the fusion of two σs has two possible channels. The second fusion rule indicates that fusing a ψ with a σ gives back a σ. The third states that when two fermions are brought together they are fused to the vacuum.

Assume that one creates from the vacuum two pairs of anyons (σ, σ), as seen in Figure 3(a). Then the fusion rules imply that the generation of anyons results

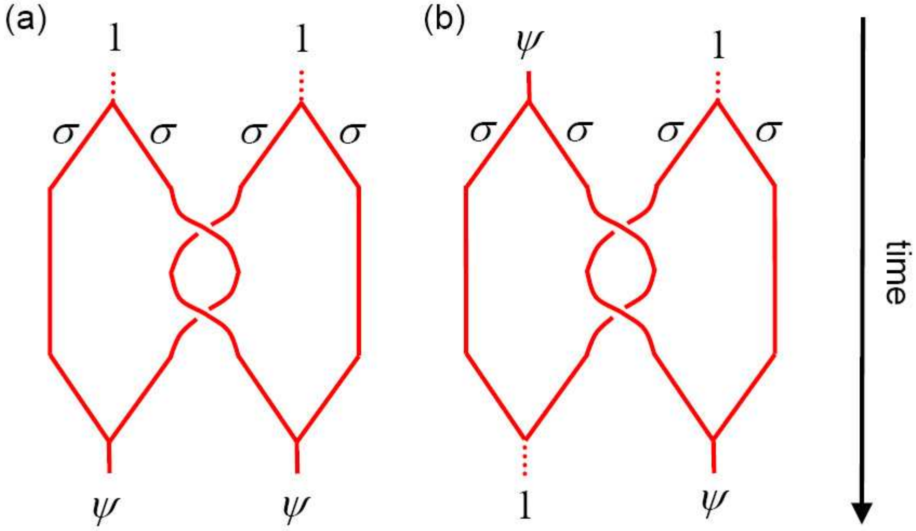


Fig. 3. Figure 1.3: The world lines of the Ising anyons where the third dimension depicts time running downwards. (a) From the vacuum two pairs of Ising anyons are generated depicted by (σ, σ) and (σ, σ). Then an anyon from each pair are braided by circulating one around the other. Finally, the anyons are pairwise fused, but they do not necessarily return to the vacuum. In the case of Ising anyons the fusion outcomes are fermions ψ. (b) A similar evolution, where two pairs of σ Ising anyons are created from a fermion ψ and the vacuum 1. The braiding causes the teleportation of the fermion between the two pairs.

in a well defined pair of anyon anti-anyon. The state of these anyons can be denoted as $|\sigma, \sigma \to 1\rangle$ indicating that if we fuse back these anyons their fusion outcome is known. In general, when two arbitrary anyons are fused, it is possible to have various outcomes depending on their total state. In Figure 3(a), one can see that the fusions result in ψ anyons. For this fusion outcome we define the state $|\sigma, \sigma \to \psi\rangle$. It corresponds to the case where two anyons of type fuse to the fermion ψ when brought together. The states $|\sigma, \sigma \to 1\rangle$ and $|\sigma, \sigma \to \psi\rangle$ give rise to a two dimensional Hilbert space, the fusion space. As seen in Figure 3(a) it is possible to evolve the initially prepared state of anyons from the state $|\sigma, \sigma \to 1\rangle \otimes |\sigma, \sigma \to 1\rangle$ to the state $|\sigma, \sigma \to \psi\rangle \otimes |\sigma, \sigma \to \psi\rangle$. Due to the conservation of the total type of particles we need to keep track of the particles of one pair only. Hence, the braiding evolution can be described by a two dimensional matrix that rotates the fusion states from $|\sigma, \sigma \to 1\rangle$ to $|\sigma, \sigma \to \psi\rangle$ up to an overall phase.

In general, the fusion rules are given by

$$a \times b = N_{ab}^{c} c + N_{ab}^{d} d + ... \tag{5}$$

where anyons a and b are fused to produce anyon c or d or any other possible outcome. The integers N_{jlk} denote the multiplicity with which the particles l are generated when fusing particles j and k. This procedure is similar to the tensor product notation of spins that results in a new spin basis, e.g. $\frac{1}{2} \otimes \frac{1}{2} = 0 \oplus 1$. Abelian anyons, in particular, have only a single fusion channel $a \times b = c$ so their fusion space is one dimensional. Non-Abelian anyons have necessarily multiple fusion channels that gives rise to higher dimensional fusion spaces.

The Hilbert space $\mathcal{M}_{(n)}$ corresponding to n anyons $(a_1, ..., a_n)$ has dimension

$$\dim(\mathcal{M}_{(n)}) = \sum_{b_1 \ldots b_{n-2}} N^{b_1}_{a_1 a_2} ... N^{c}_{b_{n-2} a_n} \tag{6}$$

The representation of the fusion states is given by $|\, a, b \to c; \mu\rangle$ where $\mu = 1, ..., N^c_{ab}$ parameterises possible multiplicity in a certain fusion channel. In the case of the Ising anyons where the only non-zero coefficients are $N^1_{\sigma\sigma} = 1$, $N^\psi_{\sigma\sigma} = 1$, $N^\sigma_{1\sigma} = 1$ and $N^\sigma_{\psi\sigma} = 1$. For example, substituting $a_1 = a_2 = a_3 = c = \sigma$ in equation (6) and having the summation running over $b_1 = 1, \psi$ gives $\dim(\mathcal{M}_{(4)}) = 2$ as expected.

There is an alternative way to compute the dimension of $\mathcal{M}_{(n)}$ corresponding to n identical anyons, a, using the concept of *quantum dimension.* The quantum dimension da quantifies the rate of growth of Hilbert space dimension $\dim(\mathcal{M}_{(n)}) \to d_a^n$ when one additional a particle is inserted for large n. Starting from the fusion rules one can show that the quantum dimension satisfies the following relation

$$d_a d_b = \sum_c N^c_{ab} d_c.$$

For the case of the Ising model the quantum dimension of the ψs is given by $d_\psi^2 = 1 \Rightarrow d_\psi = 1$. For the σs we have $d_\sigma^2 = 1 + d_\psi \Rightarrow d_\sigma = \sqrt{2}$.

It is possible to access the fusion space by certain operations on the anyons. These are physically allowed operations given by changes in the fusion order, that correspond to basis change, and exchanges, or braids, leading to non-trivial evolutions. For example, one can fuse the anyons a, b and c in two distinctive ways. One can first fuse a with b and then their outcome with c or first fuse b with c and then their outcome with a. As shown in Figure 4(a), these two processes are related by a unitary matrix with elements $(F^d_{abc})^i_j$. The F matrix facilitates between the change of basis in the fusion space. For the case of the Ising model, the F matrix is given by

$$F^\sigma_{\sigma\sigma\sigma} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \tag{7}$$

in the $\{1, \psi\}$ basis.

To manipulate the states of the fusion space, one can braid the anyons before fusing them. This operation is described by the diagonal R matrix, depicted in Figure 4(b). In the case of two σ Ising anyons the components of the R matrix

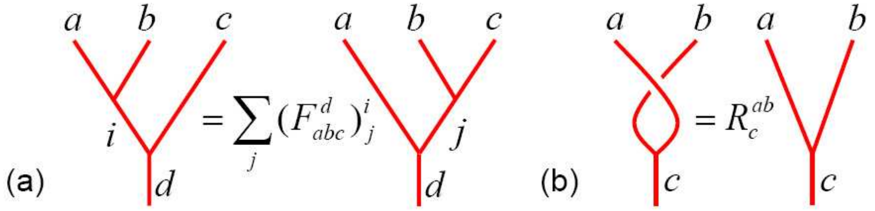


Fig. 4. Fusion and braiding properties. (a) When the order of fusion between three anyons, a, b and c with outcome d is changed then a rotation in the fusion space is performed given by the matrix F^d_{abc}. This corresponds to a change of basis in the fusion space. (b) A braiding operation between anyons a and b with fusion outcome c gives the phase R^{ab}_c.

are given by $R^{\sigma\sigma}_1 = e^{-i\pi/8}$ and $R^{\sigma\sigma}_\psi = e^{-i3\pi/8}$ [24]. That is, the fermionic fusion channel acquires an additional phase $\pi/2$ during the π rotation due to the spin 1/2 nature of the fermion. The superposition of multiple fusion outcomes in the braiding process results in unitary operations.

Return now to Figure 3(a). Initially, we pair create two anyonic pairs. Then two anyons, one from each pair, are braided by circulating one around the other. Finally, the corresponding pairs are fused. The fusion may not result in the vacuum as the braiding process could change the internal state of the anyons. Indeed, for the case of the Ising anyons the outcome of the fusion is ψ in both cases. These ψs can be further fused giving the vacuum that we had started with in agreement with the conservation of the total quantum numbers. Nevertheless, the braiding has dramatically changed the internal fusion space of each pair, even though they have not been in contact. Figure 3(b) shows the generation of one pair of Ising anyons from a fermion and another one from the vacuum. The braiding process causes the fermion to be teleported from one pair to the other. Such non-trivial unitary rotations of the fusion space are possible in the case of non-Abelian anyons by braiding operations.

Finally, one can show that consistency equations can be considered that give a relation between statistical processes and fusion relations. These consistency equations are called pentagon and hexagon equations [25] due to their geometrical interpretation (see Figure 5 and Figure 6, respectively). They are the subject of study of Topological Quantum Field Theory [26]. From the pentagon rule, the following relation between the elements of the F matrices is obtained,

$$(F^5_{12c})^d_a (F^5_{a34})^c_b = \sum_e (F^d_{234})^x_a (F^5_{1e4})^d_b (F^b_{123})^e_a . \tag{8}$$

Similarly, from the hexagon rule

$$\sum_b (F^4_{231})^c_b R^{1b}_4 (F^4_{123})^b_a = R^{13}_c (F^4_{213})^c_a R^{12}_a . \tag{9}$$

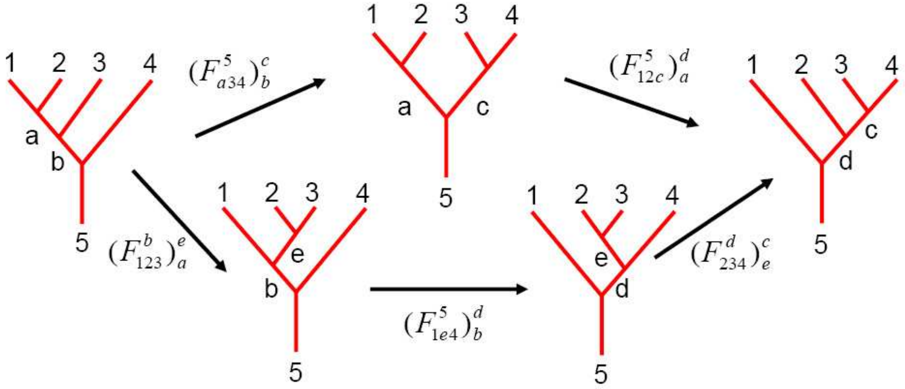


Fig. 5. The pentagon identity. Starting from a four anyon combination, a sequence of five fusion rearrangements returns to the original configuration. It is taken as an axiom that this sequence is the identity mapping.

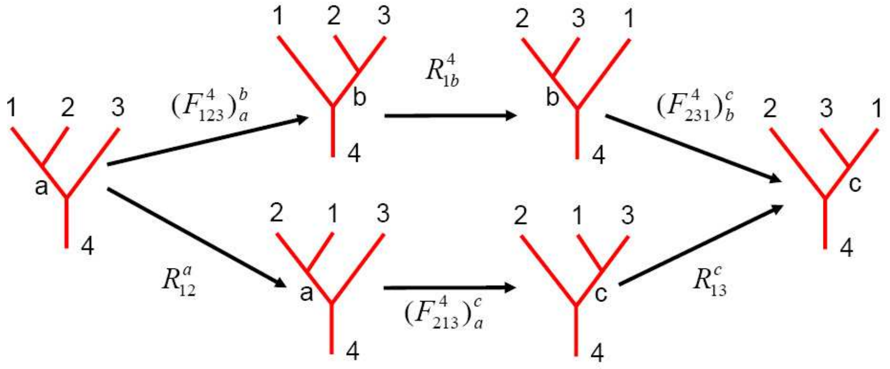


Fig. 6. The hexagon identity. It is relating the fusion of three anyons by a sequence of fusion rearrangements and braiding operations.

Consistent anyon models are characterised by some R- and F-matrices. which have to satisfy these polynomial equations. Conversely, the solutions of these two polynomial equations give a discrete set of F and R matrices. This property, known as the Ocneanu rigidity, makes the R and F unitaries immune against small perturbations [24].

The interpretation of the anyons with world lines, and in particular with world ribbons makes apparent the connection between statistics and spin [27]. In Figure 7 we see the schematical equivalence between the process of exchanging two Abelian anyons and the rotation of an anyon by 2π. The first is evolved by

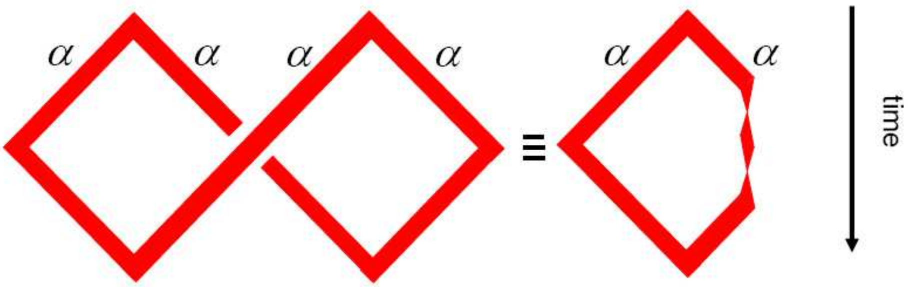


Fig. 7. The equivalence between spin and statistics. The anyons are depicted here as ribbons to keep track of their spin rotation. Two anyons, each one from two anyon anti-anyon pairs, are exchanged and then fused to the vacuum. This process can be continuously deformed to rotating one anyon from a pair by 2π and recombining them causing an evolution due to the spin of the anyon.

the statistical unitary R (a phase factor for Abelian anyons) and the second is expected to obtain a phase factor $e^{i2\pi J}$, where J is the spin of the anyon. A direct application leads to the connection between the integer spins for bosons and half integer spins for fermions. This is in agreement with the interpretation of anyons as a composite object of charge and flux. In this case the rotation of an anyon by 2π will rotate the charge around the flux, thereby leading to the Aharonov-Bohm effect.

1.4 Anyonic Quantum Computation

To implement universal quantum computation consider n non-Abelian anyons of the same type a. Apart from degrees of freedom that can be measured locally, the system possesses non-local anyonic fusion degrees. The corresponding fusion Hilbert space $\mathcal{M}(n)$ encodes the outcomes when the anyons are pairwise fused. The dimension of $\mathcal{M}(n)$ grows exponentially with the number n of anyons, i.e. $\dim(\mathcal{M}(n)) = d_a^n$, though, this does not necessarily admit a tensor product structure. Nevertheless, a qubit tensor product subspace can always be identified, where quantum information can be encoded in the usual way.

Logical gates can be performed by braiding the anyons that gives rise to applications of the R-matrix. This operation does not affect the type of anyons neither their local degrees of freedom, but it can have a non-trivial effect on the fusion space. In combination with the F-matrices one can evolve the encoded information in a non-trivial way. When these two matrices efficiently span a dense set of unitaries acting on the qubits, the corresponding non-Abelian model can support universal quantum computation. If two anyons are brought next to each other, then part of the information of the anyonic fusion space is accessible, as their fusion channel can be determined. This procedure can be employed to finally measure the computational outcome.

The information encoded in the fusion space is not at all accessible by local operations. Hence, the environment, assumed to act in a local way, cannot alter it. This is the fault-tolerant characteristic that makes anyons a favourable medium for performing quantum computation. The environmental errors that can be avoided efficiently by the topological systems are local perturbations to the Hamiltonian. Nevertheless, probabilistic errors on the system due to a finite temperature do affect the encoded space. It is still an open problem of great interest to device a method that can efficiently overcome temperature errors.

1.5 Example: Fibonacci Anyons

In this section we will present probably the most celebrated non-abelian anyonic model not only due to its simplicity and richness in structure, but also due to its connection to the Fibonacci series. In this model there are two different types of anyons, 1 (vacuum) and τ (non-Abelian anyon), that have the following fusion rules

$$1 \times 1 = 1, \;\; 1 \times \tau = \tau, \;\; \tau \times \tau = 1 + \tau \tag{10}$$

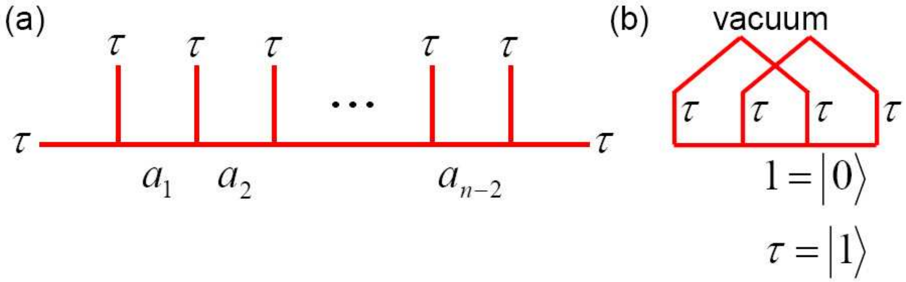


Fig. 8. Depiction of the fusion process for anyons. (a) A series of τ anyons are fused together ordered from left to right. The first two τ anyons are fused and then their outcome is fused with the next τ anyon and so on. (b) Four Fibonacci anyons in state τ created from the vacuum can be used to encode a single logical qubit.

It is interesting to study all the possible outcomes when we fuse $n+1$ anyons of type τ arranged as in Figure 8. For this we initially fuse the first two anyons, then their outcome is fused with the third τ anyon and the single outcome is fused with the next one and so on. At each step i we assign an index a_i that indicates the outcome of the fusion at that step. The states $|a_1, a_2, ..., a_{n-2}\rangle$ belong to the fusion Hilbert space of the anyons, $\mathcal{M}(n)$. These states are not all independent. As the τ anyons have two possible fusion outcome states, it is natural to ask, in how many distinct ways, $d_\tau(n)$, can one fuse $n+1$ anyons of type τ to yield finally a τ. At the first fusing step the possible outcomes are 1 or τ, giving $d_\tau(2) = 1$. When we fuse the outcome with the next anyon then $1 \times \tau = 1$ and $\tau \times \tau = 1 + \tau$, resulting to two possible τs coming from two different processes and a 1, i.e. $d_\tau(3) = 2$. Taking the possible outcome and fusing it with the next anyon gives a space of τs which is three dimensional, $d_\tau(4) = 3$. One soon notices that the dimension of the fusion space $\dim(\mathcal{M}(n))$ when n anyons of type τ are fused, is actually the Fibonacci series. This dimension is given approximately by the following formula

$$\dim(\mathcal{M}(n)) \propto \phi^n$$

where $\phi \equiv (1 + \sqrt{5})/2$ is the golden mean. The quantum dimensions for the two particles types can easily be solved for from the fusion rules in Eq. (10): $d_1^2 = d_1$ and $d_\tau^2 = d_1 + d_\tau$ giving $d_1 = 1$ and $d_\tau = \phi$. The golden mean has been used extensively by artists, such as Leonardo Da Vinci, in geometrical representations of nature (plants, animals or humans) to describe the ratios that are aesthetically appealing.

The Fibonacci anyonic model is a good example for realising quantum computation. We are interested in encoding information in the fusion space of anyons and then processing it appropriately. The encoding of a qubit can be visualised by employing four τ anyons as in Figure 8(b). There are two distinguishable ways the particles can be fused together that can encode the qubit $|\, 0\rangle = |\, \tau, \tau \to 1\rangle$ and $|\, 1\rangle = |\, \tau, \tau \to \tau\rangle$. To determine the quantum gates one needs to evaluate the F and R matrices. From the fusion rules of Fibonacci anyons and the pentagon identity one finds the non-zero values

$$\begin{aligned} &(F^{\tau}_{\tau\tau 1})^1_1 = (F^{\tau}_{1\tau\tau})^1_1 = (F^{\tau}_{\tau 1 \tau})^1_1 = F(^{1}_{\tau\tau\tau})^1_1 = 1, \\ &(F^{1}_{111})^0_0 = 1, F^{\tau}_{\tau\tau\tau} = \begin{pmatrix} \frac{1}{\phi} & \frac{1}{\sqrt{\phi}} \\ \frac{1}{\sqrt{\phi}} & -\frac{1}{\phi} \end{pmatrix}. \end{aligned}$$

These solutions are unique up to a choice of gauge. Inserting these values into the relations demanded by the hexagon identity, one obtains the following R matrix describing exchange of two particles

$$R = \begin{pmatrix} e^{4\pi i/5} & 0 \\ 0 & -e^{2\pi i/5} \end{pmatrix}. \tag{11}$$

It can be shown that the unitaries $b_1 = R$ and $b_2 = F^{\tau}_{\tau\tau\tau} R (F^{\tau}_{\tau\tau\tau})^{-1}$ acting in the logical space $|\, 0\rangle$ and $|\, 1\rangle$ are dense in $SU(2)$ in the sense that they can reproduce any element of $SU(2)$ with accuracy ϵ in a number of operations that scales like $O(poly(log(1/\epsilon))$ [28]. Thus an arbitrary one qubit gate can be performed as follows. Begin from the vacuum and prepare four anyons labelled a_1, a_2, a_3, a_4. Braiding the first and second anyons implements b_1 and braiding the second and third anyons implements b_2. A measurement of the outcome upon fusing a_1 and a_2 projects onto logical $|\, 0\rangle$ or $|\, 1\rangle$. Similarly, by performing braiding over 8 anyons in state 1, one obtains a dense subset of $SU(d(7))$. Since $SU(4) \subset SU(13)$, we can also implement any two logical qubit gate, e.g. the CNOT gate, with arbitrary accuracy. Hence the Fibonacci anyon model allows for universal computation on n logical qubits using $4n$ physical anyons [29].

1.6 Exercises

- Exercise 1: Show that the Ising model F and R matrices satisfy the pentagon and hexagon identities. Conversely, solve the pentagon and hexagon equations given the Ising type of anyons and their fusion rules.
- Exercise 2: Construct the qubit space with Ising anyons and demonstrate that the F and R matrices do not provide a universal set of gates. What gates are missing? (see [30]).
- Exercise 3: Solve the pentagon and hexagon equations for the the F and R matrices of the Fibonacci model.
- Exercise 4: From the spin statistics theorem demonstrate what is the spin of the Ising and Fibonacci anyons.
- Exercise 5: Bratteli diagrams give a pictorial representation of the fusion outcomes of n anyons of the same type. Their horizontal axis gives the increasing number of anyons involved in the fusion and the vertical axis gives the fusion outcome. There are many distinct paths that start from the first anyon and evolve according to the choice of the fusion channels at each step. Draw all the distinct Bratteli diagrams for the Ising and Fibonacci models involving six anyons.
- Exercise 6: Starting from the initial vacuum state with Ising anyons can you generate entangled states?
- Exercise 7: Generalise the spin statistics theorem for the non-Abelian anyonic case.

2 Quantum Double Models

2.1 From Error Correction to Topological Models

Quantum Error Correction. Quantum error correction is the means we have to combat environmental and control errors when performing quantum computation. As errors infest any physical realisation of a quantum computer the importance of quantum error correction cannot be overstated. Much in parallel to classical error correction quantum error correction works on the principle of employing a large Hilbert space to encode information in a redundant way. The goal is to perform complex encoding and decoding of information so that for low enough error rates the effect of the environment is eventually neutralised. In the following we shall review some basic properties of quantum error correction. It is pedagogical and conceptually appealing to approach topological models from the quantum error correction point of view. Consider a Hilbert space $\mathcal{H}$ of a quantum system spanned by n finite dimensional complex subsystems $\mathcal{V}$, i.e. $\mathcal{H} = \mathcal{V} \otimes ... \otimes \mathcal{V}$. For simplicity, and unless it is otherwise stated, we shall initially consider a set of qubits, i.e. $\dim(\mathcal{V}) = 2$. The code space $\mathcal{C}$ is a linear subspace of $\mathcal{H}$ where the logical information is encoded. A general k-local operator $\mathcal{O}$ is an operator that acts non-trivially to at most k subsystems of $\mathcal{H}$ (also known as operator of length k). Then $\mathcal{C}$ is called a k-code if

$$\Pi_{\mathcal{C}} \mathcal{O} \Pi_{\mathcal{C}} \propto \Pi_{\mathcal{C}} \qquad (12)$$

where $\Pi_{\mathcal{C}}$ is the projector on $\mathcal{C}$. Hence, the operator $\Pi_{\mathcal{C}}\mathcal{O}$ is a mapping $\mathcal{C} \mapsto \mathcal{C}$ up to a multiplicative scalar for any k-local operator $\mathcal{O}$. It has been shown [31] that such a code can effectively protect against errors that act on less than $k/2$ qubits. The code is also called $[[n, d, k]]$ where n is the total number of qubits and 2^d is the dimension of the k-code $\mathcal{C}$. This code requires n physical qubits to encode d logical ones protected against errors that are at most $\lfloor k/2 \rfloor$-local. Quantum error correcting codes are commonly expressed in stabiliser formalism introduced in the following.

Stabiliser Formalism. A stabiliser $\mathbf{T}_n$ is a set of hermitian operators, T_i, with $i = 1, ..., n$ that commute with each other, $[T_i, T_j] = 0$ for all i, j. The stabilised space consists of all eigenstates $|\Psi\rangle$ with eigenvalue $+1$ for all operators T_i. A particularly example of stabilisers can be constructed from the Pauli group, $\mathbf{P}_n$, generated by the Pauli matrices $\sigma^x, \sigma^y, \sigma^z$ and the identity $\mathbb{1}$ acting on n qubits. As different Pauli operators acting on the same qubit anticommute only a subset of the Pauli group commute with each other and hence form a stabiliser set. These operators are Hermitian and they square to the identity, so they have eigenvalues ± 1. Such a maximal stabiliser set could admit a common set of 2^n eigenstates uniquely identified by the pattern of the stabiliser eigenvalues. Generalisations of stabilisers to qudits are also possible. These will play a special role in the definition of the quantum double topological models presented in the following section.

One can define an error correcting code with the stabiliser formalism e.g. based on the Pauli group $\mathbf{P}_n$. Consider a certain commuting subgroup S of the group $\mathbf{P}_n$. Then the set of eigenstates of all elements of S with eigenvalue $+1$ is the stabiliser code $\mathcal{C}$, where information can be stored. If S has s elements, it can encode $d = n - s$ qubits. Any operation acting on $\mathcal{C}$ that does not commute with S can be detected by measurements of the S observables and it can be corrected. The set of operators that commute with all the elements of S is called the centraliser $Z(S)$. The centraliser naturally includes S, but in general will have extra elements. The distance k of the code $\mathcal{C}$ is the minimal length among the elements of $Z(S) \setminus S$ up to a sign. Such elements serve as the encoded logical operations. For an efficient encoding we thus assume that the errors are less than $\lfloor k/2 \rfloor$-local.

Topological Models. One can obtain topological models that support anyons by employing the error correction formalism. This facilitates the presentation of the fault-tolerant encoding of information in a physical system. Consider a general two dimensional lattice defined on a surface M and a Hamiltonian $H = -\sum_i h_i$ with local interaction terms h_i acting on the links of the lattice. Assume that the hi are the elements of the subgroup S. Then the ground state space of H is the stabiliser code $\mathcal{C}$ which is 2^d-dimensional and it is separated from the rest of the states by a finite energy gap. Turning an error correcting code into a Hamiltonian is a dramatic conceptual step which was first taken by Kitaev [13]. Apart from the fault- tolerant characteristics it provides a geometrical interpretation of quantum error correction, thus bringing it closer to its physical realisation. Its drawback is an overhead in the number of employed qubits. We should bear in mind that, as the error correcting procedure has been substituted by a gapped. Hamiltonian that aims to penalise the generation of errors, the errors we consider here are coherent, i.e. they are perturbations to the Hamiltonian. The latter can cause virtual excitations which are automatically suppressed by keeping the characteristic size of the system large compared to the length of the perturbation. The size of the system corresponds to having a large distance k for the $\mathcal{C}$ code, as it will become apparent in the following section. If the errors are generated by thermal noise, then the mechanism described above cannot automatically correct them. Alternative methods have to be considered that are the subject of ongoing research.

We can demonstrate now with general arguments that particular Hamiltonians described by the error correcting formalism can actually support quasiparticles with anyonic statistics [34]. These quasiparticles are localised excitations of a Hamiltonian H. Their location i is determined from the violation of the stabiliser condition $h_i \,|\,\Psi\rangle = |\,\Psi\rangle$ which corresponds to the ground state. We shall focus on the space $\mathcal{C}$ of states that corresponds to the presence of a number of excitations. Assume that one can change the position of the quasiparticles in time by making the Hamiltonian time dependent, H_t. In particular, we are interested in the evolution of the state space $\mathcal{C}$ when one quasiparticle is braided around another. For simplicity we discretise the time evolution of the punctures in small steps. Consider such a small step with corresponding Hamiltonians H_{t_i} and $H_{t_{i+1}}$. Assume that these Hamiltonians are related by a $\lfloor k/2 \rfloor$-local Hermitian operator $\mathcal{O}_i$ in an isospectral way

$$H_{t_{i+1}} = e^{-i\mathcal{O}_i\varepsilon} H_{t_i} e^{i\mathcal{O}_i\varepsilon} \tag{13}$$

for some small value of ε. We are interested to see, what is the action of $\mathcal{O}_i$ on the corresponding code spaces given by $\mathcal{C}_{t_i}$ and $\mathcal{C}_{t_{i+1}}$. If both of the Hamiltonians H_{t_i} and $H_{t_{i+1}}$ correspond to k-codes, then the rotation of the Hamiltonian acts on the space $\mathcal{C}_{t_{i+1}}$ with the projector $\Pi_{G_{t_i}} \mathcal{O}_i$. This gives rise to an adiabatic transport. The total evolution of the system from time 0 to time T is given by

$$U(0,T) = \mathbf{T} \lim_{N\to\infty} \prod_{i=1}^{N} \Pi_{\mathcal{C}} \mathcal{U}_i e^{-iH_{t=0}\Delta t} \mathcal{U}_i^\dagger \Pi_{\mathcal{C}} \tag{14}$$

where $\mathcal{U}_i = \prod_{j\leq i} e^{-i\mathcal{O}_j\varepsilon}$. Assuming that the adiabaticity condition takes place at each time step, then an initially prepared system in the code space $\mathcal{C}$ will remain there. Nevertheless, states in the code space evolve in general. One can explicitly show [19] that the operator $U(0,T)$ evolves the code space by the Holonomy $\Gamma_A(\gamma)$ given in (3) for the case of cyclic adiabatic evolutions of the Hamiltonian. Hence, by carefully engineering Hamiltonians one can obtain evolutions that give rise to an Abelian or a non-Abelian geometrical phase $\Gamma_A(\gamma)$. In the following we shall present a systematic way how to develop Hamiltonians that support quasiparticles with a variety of anyonic statistical behaviours.

2.2 Quantum Double Models

The Simple Case of the Toric Code. The toric code [13] is the simplest topological lattice model that supports Abelian anyons. It is one of the most studied topological models serving as a platform to develop new computational schemes and as a test bed to probe the properties of topological systems. It comprises of a square lattice with qubits positioned at its links and interaction terms that act at the vertices, $A(v)$, and plaquettes, $B(p)$, of the lattice. They are given by

$$A(v) = \sigma^x_{v,1}\sigma^x_{v,2}\sigma^x_{v,3}\sigma^x_{v,4}, \quad B(p) = \sigma^z_{p,1}\sigma^z_{p,2}\sigma^z_{p,3}\sigma^z_{p,4} \tag{15}$$

where the indices $1, ..., 4$ of the Pauli operators, σ^z and σ^x, enumerate the vertices of each plaquette or vertex in a clockwise fashion. The defining Hamiltonian is

$$\mathcal{H} = -\sum_v A(v) - \sum_p B(p) \ , \tag{16}$$

Each of the interaction terms commute with the Hamiltonian as well as with each other. Thus, the model is exactly solvable and its ground state is explicitly given by

$$|\text{gs}\rangle = \prod_v \frac{1}{\sqrt{2}}\big(\mathbb{1} + A(v)\big)|00...0\rangle \ , \tag{17}$$

with $\sigma^z|0\rangle = |0\rangle$. The state $|\text{gs}\rangle$ represents the anyonic vacuum state and it is unique for systems with open boundary conditions.

Starting from this ground state one can excite pairs of anyons connected by a string on the lattice using single qubit operations. More specifically, by applying σ^z on some qubit of the lattice a pair of so called e-type anyons is created on the two neighbouring vertices (see Figure 9(a)). The system is described by the state $|e\rangle = \sigma^z|\text{gs}\rangle$. An m pair of anyons lives on the plaquettes and is obtained by a σ^x operation. The combination of both creates the composite quasiparticle ϵ with $|\epsilon\rangle = \sigma^z\sigma^x|\text{gs}\rangle = i\sigma^y|\text{gs}\rangle$. These excitations are detected by measuring the eigenvalues of the corresponding $A(v)$ or $B(p)$ operators. Two equal Pauli rotations applied on qubits of the same plaquette or vertex create two anyons on this plaquette or vertex, respectively. The fusion rules, $1 \times 1 = e \times e = m \times m = \epsilon \times \epsilon = 1$, $e \times m = \epsilon$, $1 \times e = e$, etc., where 1 is the vacuum state, describe the outcome from combining two anyons. In the above example, if two anyons are created on the same plaquette or vertex then they annihilate. This operation also glues two single strings of the same type together to form a new string, again with a pair of anyons at its ends (Figure 9(b)). If the string forms a loop, the anyons at its end annihilate each other, thus removing the anyonic excitation. In case that only a part of the string forms a loop, the string gets truncated (Figure 9(c)). For non-compact systems with boundaries a string may end up at the boundary describing a single anyon at its free endpoint.

Anyonic statistics is revealed as a non-trivial phase factor acquired by the wave function of the lattice system after braiding anyons, e.g., after moving an m anyon around an e anyon (Figure 9(d)) or vice versa. Consider the initial

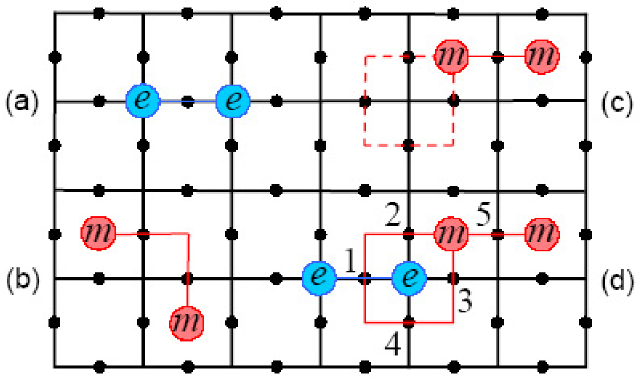


Fig. 9. The toric code lattice with qubits at the links of a square lattice. Qubit rotations enable manipulations of anyons on neighbouring plaquettes or vertices. (a) Application of σ^z on a single qubit yields two e-type anyons placed at neighbouring vertices, where the string passes trough the rotated qubits. Similarly, m anyons are created on plaquettes by σ^x rotations. (b) Two σ^x rotations create two pairs of m-type anyons. If one anyon from each pair is positioned on the same plaquette then they annihilate, thereby connecting their strings. (c) When a part of a string forms a loop around unpopulated plaquettes, the loop cancels (dashed). (d) Anyon e is produced by a σ^z on qubit 1, $|e\rangle = \sigma_1^z|\text{gs}\rangle$. Subsequently an m anyon is circulated around the e anyon giving rise to a non-trivial phase.

state $|\Psi_{\text{ini}}\rangle = \sigma_5^x\sigma_1^z|\text{gs}\rangle$ with neighbouring$w$ and m anyons. One m anyons is then moved around an e along the path generated by successive applications of σ^x rotations on the four qubits of v where e resides. The final state is

$$|\Psi_{\text{fin}}\rangle = \sigma_1^x\sigma_2^x\sigma_3^x\sigma_4^x|\Psi_{\text{ini}}\rangle = -\sigma_1^z(\sigma_1^x\sigma_2^x\sigma_3^x\sigma_4^x|\text{gs}\rangle) = -|\Psi_{\text{ini}}\rangle \ . \qquad (18)$$

Such a minimal loop, which vanishes the moment it is closed, is analogous to the application of the respective interaction term $A(v) = \sigma_{v,1}^x\sigma_{v,2}^x\sigma_{v,3}^x\sigma_{v,4}^x$ (or $B(p) = \sigma_{p,1}^z\sigma_{p,2}^z\sigma_{p,3}^z\sigma_{p,4}^z$) of the Hamiltonian. This operator has eigenvalue $+1$ for all plaquettes of the ground state $|\text{gs}\rangle$. It signals an excitation, e.g., $|\Psi_{\text{ini}}\rangle$, with eigenvalue -1, when applied to the plaquette where an anyon resides, which is our case. However, (18) is much more general, as the actual path of the loop is irrelevant. It is worth noticing that the e and m anyons are distinguishable as they reside exclusively on vertices or plaquettes of the lattice, respectively. Hence, their exchange is not possible only their braiding (double exchange). This still gives a behaviour that is different from the braiding of bosons or fermions. More complicated anyonic models can give rise to anyonic statistics between indistinguishable particles.

Alternatively, we can interpret (18) as a description of twisting ϵ, the combination of an e and an m-type anyon, by 2π. The phase factor of -1 thereby reveals its 4π-symmetry, which is characteristic for half spin, fermionic particles [32]. Note that the e (m) anyons exhibit bosonic statistics with respect to themselves [13].

The properties mentioned above do not need a torus configuration for the lattice Hamiltonian. This becomes necessary only when one wants to employ the toric code to encode quantum information. Indeed, non-trivial genus can give rise to degeneracy that can serve as a quantum memory. This is a general property that holds for other Abelian models as well. For example, moving from one ground state to another on a torus with genus one involves creating a pair of anyons and then moving them along non-contractible loops before re-annihilating them, as seen in Figure 2.2. Denoting two non-equivalent trajectories on the torus as 1 and 2 then one can define the states

$$|\,\Psi_1\rangle\,,\;|\,\Psi_2\rangle = C_e^1\,|\,\Psi_1\rangle\,,\;|\,\Psi_3\rangle = C_e^2\,|\,\Psi_1\rangle\,,\;|\,\Psi_4\rangle = C_e^1 C_e^2\,|\,\Psi_1\rangle\,. \tag{19}$$

The operators C_e^1 and C_e^2 correspond to generating a pair of e anyons, moving then along the directions 1 or 2 respectively and then annihilating them. Continuously deformed anyonic loops correspond to the same states. So only four states can be created in this way. A linearly dependent set of states can be obtained by employing the loop operators that correspond to m or combinations of e and m anyons. Hence, a four dimensional Hilbert space arises that can encode two qubits. If the toric code is defined on a surface with genus g, then it can encode $2g$ qubits.

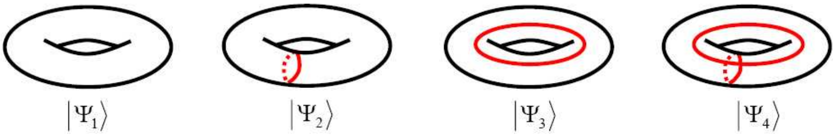


Fig. 10. The torus with genus $g = 1$ with the toric code model defined on it. The ground state has fourfold degeneracy $|\,\Psi_i\rangle$ for $i = 1, ..., 4$. Starting from the vacuum state $|\,\Psi_1\rangle$ one can create a pair of m anyons and wrap them around two non-trivial inequivalent loops on the torus giving the states $|\,\Psi_2\rangle$ and $|\,\Psi_3\rangle$. State $|\,\Psi_4\rangle$ corresponds to generating two inequivalent loops.

One should note the similarity with error correcting codes: the operators $A(v)$ and $B(p)$ are the commuting operators that detect errors, while operators that create a pair of anyons, move one of them around a non-contractible loop and re-annihilate them correspond to encoded logical gates. When anyonic errors are detected, then a string of operations is performed along the shortest distance between the anyons that annihilates them. This elimination of errors can affect the logical space only if the two errors have propagated at distance larger than $L/2$, where L is the linear size of the torus. In that case the error correction step might result in a non-contractible loop that corresponds to a logical gate. Hence, the toric codes corresponds to a $[[L^2, 2, L]]$ error correcting code. With the Hamiltonian present, the generation of errors is penalised by an energy gap. Errors in the form of virtual anyonic excitations are exponentially suppressed from going around the torus. Hence, large tori sizes are favourable. Such Abelian models can serve only as memories as the encoded logical gates are not sufficient to perform universal quantum computation.

General $D(G)$ Quantum Double Models. Quantum double models are particular two dimensional lattice models with Hamiltonians that support a rich variety of anyonic excitations [13, 33]. The toric code is a special case of a quantum double model, $D(Z_2)$, based on the group $Z_2 = \{1, e; e^2 = 1\}$. Consider the orthonormal basis $\{|\, g\rangle : g \in G\}$ that produces a Hilbert space, $\mathcal{H}$, with dimensionality $|G|$. Assign a space $\mathcal{H}$ to each link of the lattice which can be thought of as a spin or qudit. In the case of the toric code, this space is two dimensional corresponding to a qubit.

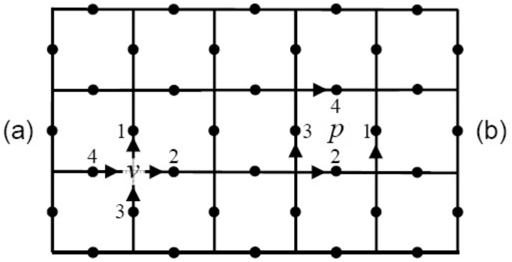


Fig. 11. Quantum double models can be defined on a square lattice where qudits are places at the links. The lattice is oriented with vertical links pointing upwards and horizontal ones rightwards. The enumeration for the links of a vertex (a) and of a plaquette (b) is also given.

To define the Hamiltonian we need the linear operators L_+^g, L_-^g, T_+^h and T_-^h acting on $\mathcal{H}$ with $g, h \in G$ acting as

$$L_+^g \,|\, z\rangle = |\, gz\rangle\,, \quad L_-^g \,|\, z\rangle = \left|\, zg^{-1}\right\rangle, \quad T_+^h \,|\, z\rangle = \delta_{h,z} \,|\, z\rangle\,, \quad T_-^h \,|\, z\rangle = \delta_{h^{-1},z} \,|\, z\rangle \tag{20}$$

These operators satisfy the following commutation relations

$$L_+^g T_+^h = T_+^{gh} L_+^g, \quad L_-^g T_+^h = T_+^{hg^{-1}} L_-^g, \quad L_+^g T_-^h = T_-^{hg^{-1}} L_+^g, \quad L_-^g T_-^h = T_-^{gh} L_-^g \tag{21}$$

Let us consider an orientation for each edge of the lattice. For concreteness we take a square lattice were the vertical links are oriented upwards and the horizontal ones rightwards, as seen in Figure 11. For each vertex v of the lattice we assign a vertex operator defined by

$$A(v) = \frac{1}{|G|} \sum_{g \in G} L_+^g(e_1) L_+^g(e_2) L_-^g(e_3) L_-^g(e_4), \tag{22}$$

where the e_i's correspond to the four edges connected to vertex v, as seen in Figure 11. Similarly for a plaquette p one can define

$$B(p) = \sum_{h_1 \dots h_4 = 1} T_-^{h_1}(e_1) T_-^{h_2}(e_2) T_+^{h_3}(e_3) T_+^{h_4}(e_4). \tag{23}$$

$A(v)$ projects out the states that are gauge invariant at vertex v and $B(p)$ projects out the states with vanishing magnetic charge at plaquette p. All of the operators $A(s)$'s and $B(p)$'s commute with each other. Hence, the Hamiltonian

$$H = -\sum_v A(v) - \sum_p B(p) \tag{24}$$

is in the stabiliser formalism and can be easily diagonalised. The ground state $|\,\mathrm{gs}\rangle$ satisfies

$$A(v)\,|\,\mathrm{gs}\rangle = |\,\mathrm{gs}\rangle\,, \quad B(p)\,|\,\mathrm{gs}\rangle = |\,\mathrm{gs}\rangle \tag{25}$$

for all v and p. The excitation states of this Hamiltonian are quasi-particles that live on the vertices or the plaquettes of the lattice or simultaneously on a vertex and a neighbouring plaquette, where the conditions (25) are violated. It is possible to find the projectors that identify the type of quasi-particles and their properties, but this problem is complex in its generality. The quasiparticles can be Abelian, arising for example from the toric code model with Z_2 group, or non-Abelian arising e.g. from the S_3 group.

The main property of the non-Abelian anyonic Hamiltonians that is of interest for quantum computation is their large fusion space degeneracy created by the presence of non-Abelian anyons. There quantum information can be encoded which is protected from errors by the finite energy gap above it. Moreover, the encoding can be performed in a non- local way making it inaccessible to environmental decoherence. The advantage over the Abelian anyon encoding, as we have seen for example with the toric code, is that now one can manipulate the information by braiding the anyons together, rather than by creating anyons and circulating them around the torus. In addition, the dimension of the encoding space can be increased by creating more anyons rather than changing the topology (genus) of the surface. This dramatically simplifies the control procedure and can give rise, for certain types of non-Abelian models, to universal quantum computation.

2.3 Exercises

- Exercise 1: Develop explicitly the Quantum Double theory for the Z_2 group. $D(Z_2)$ is the toric code.
- Exercise 2: Demonstrate the anyonic properties of the Abelian $D(Z_n)$ model.
- Exercise 3: Write down all the anyonic particles for $D(S_3)$ and demonstrate their non-Abelian fusion and statistics.

3 Jones Polynomials

3.1 A New Quantum Algorithm!

The study of anyonic systems for performing quantum computation has led to the exciting discovery of a new quantum algorithm that evaluates the Jones polynomials [38]. These polynomials are topological invariants of knots and links. They were first connected to topological quantum field theories by Witten [26]. Since then they have found far reaching applications in various areas such as in biology for DNA reconstruction and in statistical physics [39]. The best known classical algorithm for the exact evaluation of Jones polynomials demands exponential resources [40]. Employing anyons only a polynomial number of resources is required to produce an approximate answer of this problem [41]. The techniques used by manipulating anyons resemble an analogue computer. Indeed, the idea is equivalent to the classical setup, where a wire is wrapped several times around a solenoid that confines magnetic flux: by measuring the current that runs through the wire one can obtain the number of times the wire was wrapped around the solenoid. The translation of the corresponding anyonic evolution to a quantum algorithm was explicitly demonstrated in [42].

3.2 Braid Group and Traces

To better understand the structure of the computation, let us first introduce a few necessary elements. The main mathematical structure behind the evolution of anyons is the braid group B_n on n strands. Its elements b_i for $i = 1, ..., n-1$ can be viewed as braiding the world lines of anyons. Specifically, if n anyons are placed in a certain order, then the element b_i describes the effect of exchanging the position of anyons i and $i+1$, e.g. in a counterclockwise fashion. Thus all possible manipulations between the anyons can be written as a combination of the b_is. The elements of the group B_n satisfy the following relations

$$b_i b_j = b_j b_i, \text{ for } |i-j| \geq 2, \tag{26}$$

$$b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1}, \text{ for } 1 \leq i < n, \tag{27}$$

$$b_i b_i^{-1} = b_i^{-1} b_i = e, \tag{28}$$

where e is the identity element of the group. These relations have a simple diagrammatic interpretation, which can be found in Figure 12. Even though we can represent the braids with diagrams we should not forget that we are actually interested in their matrix representation.

The next element we need for the quantum algorithm is the introduction of a trace that establishes the equivalence between braidings and knots or links. A version of this tracing procedure called the Markov trace consists of connecting the opposite endpoints of the braids together, as shown in Figure 13(a). Alternatively, one might connect neighbouring strands in a pairwise fashion. This gives rise to the Plat trace shown in Figure 13(b). Hence, a braid with a trace gives a knot or a link. Surprisingly, every knot or link is equivalent to a braid with a trace due to Alexanders theorem [43]. Hence, one can simulate a knot or a link by simply braiding anyons of the same type.

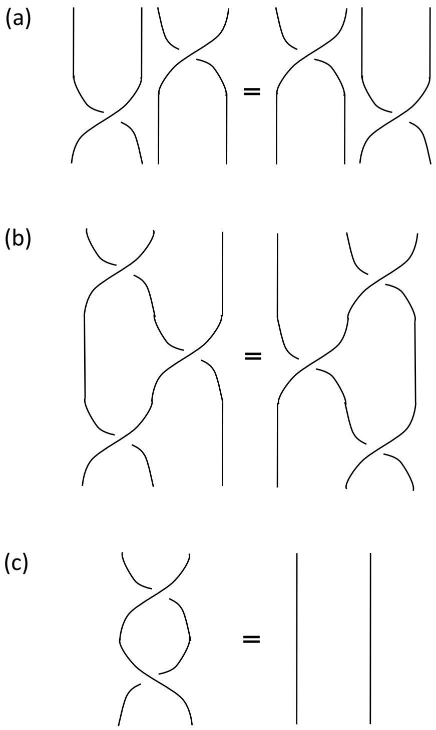


Fig. 12. Schematic representation of the Yang-Baxter equations. (a) Exchanging the order of two braids does not have an effect if they are sufficiently far apart, i.e. $b_i b_j = b_j b_i$ when $|i - j| \geq 2$. (b) Two braidings are equivalent under simple continuous deformations of the strands, $b_i b_{i+1} b_i = b_{i+1} b_i b_{i+1}$ for $1 \leq i < n$. (c) Undoing a braid gives the identity $b_i b_i^{-1} = e$.

3.3 Reidemeister Moves and Skein Relations

To demonstrate the relation between anyons and the Jones polynomials we need to introduce a way of assigning topological invariant polynomials to links. First we need to define equivalency classes between links that can be continuously deformed into each other. Surprisingly three elementary moves are sufficient to establish these equivalencies. They are called the Reidemeister moves depicted in Figure 14. Topologically equivalent links can be related by these moves.

Next we want to assign polynomials to links that are insensitive to Reidemeister moves. As seen in Figure 15(a,b), the Skein relations reduce the crossings of the links to a combination of avoided crossings and coefficients parameterised by A. If this is applied to all crossings, then the only left components are unlinked loops. By substituting each loop with d as seen in Figure 15(c) we obtain a Laurent polynomial in A. This polynomial is called the Kauffman bracket $\langle L \rangle (A)$ of the link L. Kauffman brackets have the following properties $\langle LO \rangle = d \langle L \rangle$ for L being a general non-empty link and O being the trivial link, while $\langle O \rangle = 1$.

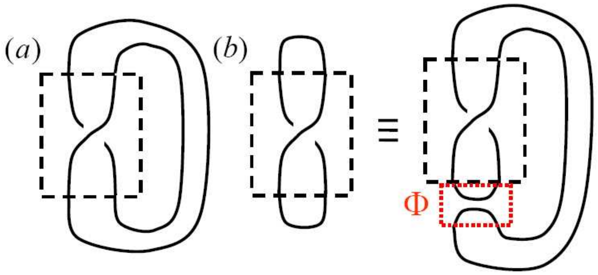


Fig. 13. (a) The Markov trace performed by linking the opposite ends of the strands. (b) The Plat trace connects pairwise neighbouring strands. The Plat trace can be expressed as a Markov trace with the addition of the *Φ* graph element.

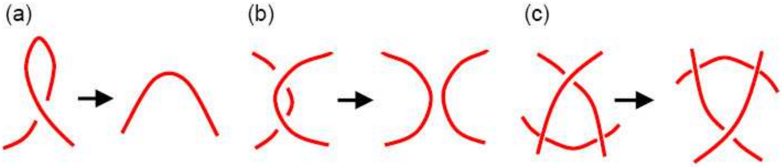


Fig. 14. (a) The first Reidemeister move undoes a twist. (b) The second Reidemeister move separates two unbraided strands. (c) The third Reidemeister move slides a strand under a crossing.

One can show that the Kauffman bracket satisfies the Reidemeister moves (a,b). Indeed, by employing the Skein relations we see that the last two Reidemeister moves are identically satisfied. To satisfy the first Reidemeister move one needs to rescale the Kauffman bracket in the following way

$$V_L(A) = (-A)^{3w(L)}\langle L\rangle(A) \tag{29}$$

The parameter $w(L)$ is the writhe or twist of the link. For an oriented link assign a $+1$ to a clockwise crossing and a -1 to an anticlockwise crossing. The writhe is then the sum of these signs for all crossings. One can show that the Jones polynomial, $V_L(A)$, is an invariant with respect to the first Reidemeister move. It is hence a topological invariant of links.

Finally, one can show that the trace of unitary representations of the braiding group corresponds to the Kauffman bracket of the Markov trace of braided strands [42]. Thus one has

$$\mathrm{tr}(B(A)) = \langle (B)^{\mathrm{Markov}}\rangle(A). \tag{30}$$

where the braid B have been Markov traced and then the corresponding Kauffman bracket has been evaluated.

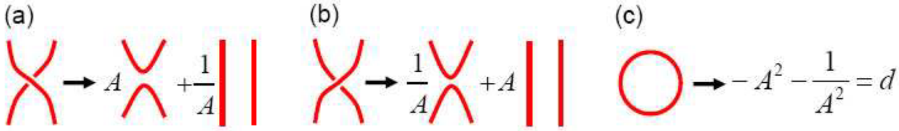


Fig. 15. (a) and (b) depict the Skein relations. (c) For every closed loop we assign the number d.

3.4 Analog Evaluation of Jones Polynomials

Consider the following anyonic evolution. Create n anyons of quantum dimension d from the vacuum state in pairs. In the following, $|\,\alpha\rangle$, denotes the fusion state corresponding to the case where the fusion of the anyons in the pairwise fashion in which they were created results in the vacuum for each pair. Assume we perform an arbitrary braiding B among these anyons before we pairwise fuse them with the same ordering as the pair creation. The probability of obtaining the vacuum state at the final fusion is a measurable quantity given by

$$\langle\alpha\,|\,B(A)\,|\,\alpha\rangle=\frac{1}{d^{n/2-1}}\langle(B)^{\mathrm{Plat}}\rangle(A) \tag{31}$$

Here B has been Plat traced and then the corresponding Kauffman bracket has been evaluated. Hence one can obtain the Kauffman bracket from the probability of finally obtaining the vacuum state. The evaluation of the writhe is a polynomially easy task. So the above prescription efficiently gives the Jones polynomial. The choice of braid representation $B(A)$, associated to a particular type of anyons, depends on the parameter A that also appears as a variable of the Jones polynomial.

The Jones polynomial has been shown to be a topological invariant [38], i.e. its value for a given link L is unchanged under continuous deformations. Consider two links L_1 and L_2. If $V_{L_1}\neq V_{L_2}$ then L_1 and L_2 are inequivalent, i.e. they cannot be mapped to each other by continuous deformations. Note, however, that inequivalent links may have the same Jones polynomial. Computation of the Jones polynomial by a classical computer appears to be exponentially hard due to the fact that there are exponentially many terms to sum and no closed form for the number of loops as a function of the resolution of the link exists. On the other hand it is rather easy to approximate its value by employing anyons. The translation of the anyonic evolution to algorithms was performed by Aharonov, Jones and Landau in [42].

3.5 Exercises

- Exercise 1: Demonstrate that the last two Reidemeister moves are compatible with the Skein relations.
- Exercise 2: Demonstrate that the Jones polynomials are invariant under the first Reidemeister move. For that you might need to first check how the Skein relations reduce a single twist.

- Exercise 3: Demonstrate that the trace of a braiding corresponds uniquely to the Markov trace.
- Exercise 4: Demonstrate that the Kauffman bracket of the trefoil is given by $A^5 + A^3 - A^{-7}$.

4 Outlook

4.1 Topological Entropy

In order to perform topological quantum computation one first needs to identify if a certain medium has topological properties. As topological properties are non-local in nature we cannot expect a local order parameter to be adequate. Hamma, Ionicioiu and Zanardi [45] revealed entropic properties of the toric code ground states that are unique, due to their topological character. Subsequently, Kitaev and Preskill [46] and simultaneously Levin and Wen [47] introduced the concept of topological entropy that distinguishes if a system is topologically ordered or not. We shall briefly review these constructions.

Consider a pure system prepared in its ground state and a bipartition in R and its complement $\bar{R}$ that are separated by the boundary ∂R. Denote by ρ_R the reduced density matrix of R. Assume that its von Neumann entropy $S_R = -\text{tr}(\rho_R \ln \rho_R)$ satisfies

$$S_R = \alpha|\partial R| - \gamma + \epsilon(|\partial R|), \tag{32}$$

where $\epsilon(|\partial R|)$ tends to zero as the size of the boundary, $|\partial R|$, tends to infinity. The von Neumann entropy does not have a 'volume' term as it corresponds to a pure state, while the area law is generically expected for a gapped system. As discussed in [45–47], systems with non-zero γ are topologically ordered. Indeed, γ is related to the total topological dimension $\mathcal{D}$ of the model, i.e.

$$\gamma = \ln \mathcal{D} = \ln \sqrt{\sum_q d_q^2} \tag{33}$$

where d_q is the quantum dimension associated with anyon q. When the system does not posses anyons, then $\mathcal{D} = 1$ as it gets contributions only from the vacuum, giving $\gamma = 0$.

One can isolate the constant term from the von Neumann entropy in two different ways. Consider a system defined on a closed surface Σ, e.g. a sphere, admitting the partition in four areas, A, B, C and D, as seen in Figure 16. Then γ can be obtained from the following linear combination of entropies [46]

$$\gamma = S_A + S_B + S_C - S_{AB} - S_{AC} - S_{BC} + S_{ABC}. \tag{34}$$

This can be demonstrated from a direct substitution of (32). The entropy S_{AB} is evaluated for the composite area of A and B.

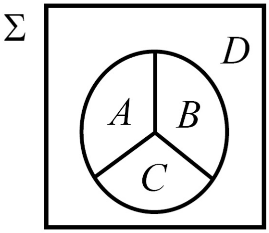


Fig. 16. The surface Σ partitioned in the areas A, B, C and D

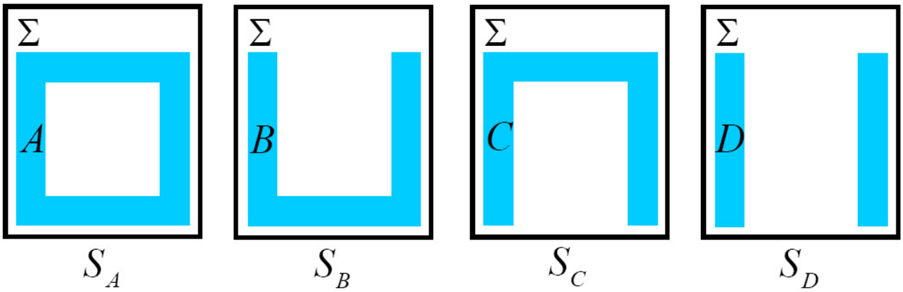


Fig. 17. The surface Σ partitioned in four different ways, A, B, C and D with corresponding von Neumann entropies S_A, S_B, S_C and S_D respectively

Alternatively, one can evaluate the entropy for the areas depicted in Figure 17. The constant part can be isolated by the following combination

$$\gamma = -\frac{1}{2}\left[(S_A - S_B) - (S_C - S_D)\right] \qquad (35)$$

Note that these are different partitions of the same system rather than parts of the same partition as employed in (34). Both relations are exact in the limit where all the involved areas R are large enough so that $\epsilon(|\partial R|) \to 0$. The latter method gives an intuitive picture for the topological character of γ. As the system is gapped the von Neumann entropies obtain in general contributions from short range correlations. The difference $S_A - S_B$ has contributions that come from the upper horizontal part of the area A. Similarly, the contributions from $S_C - S_D$ come from the a similar upper part. Hence, their difference should go to zero when the respective areas become sufficiently large. The only possible contribution could arise from a non-local operator, like a loop that wraps non-trivially around the area A. This contribution can not be cancelled from the entropies of S_B, S_C or S_D; hence it results in a non-zero value of γ. Such a contribution is indeed possible if the system is topologically ordered.

4.2 Topological Memories

As topological systems can store information in an efficient way, they have been considered as quantum memories. Indeed, it has been discussed how virtual local excitations are suppressed due to the finite energy gap that protects non-locally encoded information. If errors happen in a probabilistic way, e.g. in the presence of a non-zero temperature, then topological models seem to be weak in fighting decoherence. By studying the behaviour of the topological entropy against probabilistic errors it has been shown that finite temperature destroys the topological order of a system in general [48–50]. In other words, a finite probability of anyon generation in a system will generally alter the encoded quantum information.

Lately, several models have been presented to combat temperature errors. Their goal is to slightly alter topological models in order to recover a decoherence time that increases with the size of the system (the system is hence stable in the thermodynamic limit), or at least remains finite, but large. General limits on the passive protection of quantum information by Hamiltonians is given in [51]. Recently, Hamma, Castelnovo and Chamon [52] demonstrated that if one couples the toric code appropriately with phonon fields then at- tractive interactions between anyons are created that increase logarithmically with their distance. This will eventually suppress the generation of unwanted anyons and their propagation around the system that destroys the encoded information. An alternative method was presented by Chesi, Roethlisberger and Loss [53]. They demonstrated that polynomially repulsive interactions between the toric code anyons suppress thermal errors efficiently. Both of these models need either detailed engineering or long range interactions to become effective so they are rather hard to implement physically. It is a fascinating current topic of research to establish whether or not it is possible to engineer a quantum memory that can protect against errors in an efficient way.

In parallel a number of algorithms have been developed that aim to combat thermal errors that are inspired from topological models. For example, one could employ error correction in order to combat the loss of physical qubits from the topological system [54]. Alternative methods, that do not rely on a Hamiltonian are given by Raussendorf and Harrington [55]. They proposed a one-way quantum computation scheme integrated with topological quantum computation and magic state distillation [56] that manages to achieve an error threshold of 0.75%. This is a very successful scheme that significantly narrows the gap between theoretical constraints and experimental abilities for the performance of quantum computation.

4.3 Anyons: Where Are They?

One can easily recognise the significance of topological systems for performing fault-tolerant quantum computation. There are two main categories of physical proposals for the realisation of two dimensional topological systems. Systems that are defined on the continuum and discrete systems. It is natural to ask, which are the most promising architectures for realising them in the laboratory. Undoubtable the Fractional Quantum Hall Effect is so far the most developed area. It is concerned with a two dimensional cloud of electrons in the presence of a strong perpendicular magnetic field. There is a big variety of topological phases that arise as a function of the magnetic field and the density of electrons [57]. The most striking characteristic is that the system behaves as if it is composed of fractions of the electron charge. This was first demonstrated experimentally by Tsui, Stormer and Gossard [58] and it was theoretically explained by Laughlin [4]. Alternative systems are the $p_x + ip_y$ superconductors that can support fractional vortices with anyonic statistics [59, 60]. Recently, topological insulators came into play where they present similar properties as the fractional quantum Hall effect without the presence of a magnetic field [61]. Alternative to the continuum cloud of electrons one can consider two dimensional lattice systems. The quantum double models presented here have been proposed to be realised with Josephson junctions [62]. First experimental results for the toric code have already appeared [63] where the four spin interaction terms are realised. The demonstration of the anyonic statistics in the toric code has performed with four photons [64] and six photons [65]. A drawback of these models is the need for many body interactions. A non-Abelian anyonic model that requires only two body interactions has been presented by Kitaev [24]. A Proposal for its realisation with cold atoms was given by Duan, Demler and Lukin [66] and with polar molecules by Micheli, Brennen and Zoller [67].

4.4 Exercises

- Exercise 1: Demonstrate relation (4.2).
- Exercise 2: Demonstrate relation (4.3).
- Exercise 3: Demonstrate relation (4.4).

References

1. Leinaas, J.M., Myrheim, J.: Nuovo Cimento B 37, 1 (1977)
2. Wilczek, F.: Phys. Rev. Lett. 49, 957 (1982)
3. Tsui, D.C., Stormer, H.L., Gossard, A.C.: Phys. Rev. Lett. 48, 1559 (1982)
4. Laughlin, R.B.: Phys. Rev. Lett. 50, 1395 (1983)
5. Camino, F.E., Zhou, W., Goldman, V.J.: Phys. Rev. Lett. 98, 076805 (2007)
6. Rowell, E., Stong, R., Wang, Z.: arXiv:0712.1377 (2007)
7. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, Cambridge (2000)
8. Grover, L.K.: 28th Annual ACM Symposium on the Theory of Computing, p. 212 (1996)
9. Shor, P.: SIAM J. Comput. 26, 1484 (1997)
10. Raussendorf, R., Briegel, H.-J.: Phys. Rev. Lett. 86, 5188 (2001)
11. Farhi, E., Goldstone, J., Gutmann, S., Lapan, J., Lundgren, A., Preda, D.: Science 292, 472 (2001)
12. Castagnoli, G., Rasetti, M.: Int. J. of Mod. Phys. 32, 2335 (1993)

13. Kitaev, A.: Annals Phys. 303, 2 (2003)
14. Shor, P.: Phys. Rev. A 52, 2493 (1995)
15. Steane, A.M.: Phys. Rev. Lett. 77, 793 (1996)
16. Aharonov, Y., Bohm, D.: Phys. Rev. 115, 485 (1959)
17. Nussinov, Z., Ortiz, G.: Proc. Nat. Ac. Sc. USA 106, 16944 (2009)
18. Berry, M.V.: Roc. R. Soc. A 392, 45 (1984)
19. Pachos, J.K., Zanardi, P.: Int. J. Mod. Phys. B 15, 1257 (2001)
20. Wilczek, F., Zee, A.: Phys. Rev. Lett. 52, 2111 (1984)
21. Arovas, D., Schrieffer, J.R., Wilczek, F.: Phys. Rev. Lett. 53, 722 (1984)
22. Read, N.: Phys. Rev. B 79, 045308 (2009)
23. Lahtinen, V., Pachos, J.K.: New J. Phys. 11, 093027 (2009)
24. Kitaev, A.: Annals of Physics 321, 2 (2006)
25. Turaev, V.G.: Quantum invariants of knots and 3manifolds, de Gruyter Studies in Mathematics 18. Walter de Gruyter & Co (1994)
26. Witten, E.: Commun. Math. Phys. 121, 351 (1889)
27. Finkelstein, D., Rubinstein, J.: J. Math. Phys. 9, 1762 (1968)
28. Preskill, J.: Lecture Notes for Physics 219: Quantum Computation (2004), `http://www.theory.caltech.edu/apreskill/`
29. Freedman, M., Larsen, M., Wang, Z.: Comm. Math. Phys. 228, 177 (2002)
30. Bravyi, S.: Phys. Rev. A 73, 042313 (2006)
31. Gottesman, D.: Phys. Rev. A 57, 127–137 (1998)
32. Rauch, H., et al.: Phys. Lett. A 54, 425 (1975)
33. Bais, F.A., van Driel, P., de Wild Propitius, M.: Phys. Lett. B 280, 63 (1992)
34. Freedman, M.H., Kitaev, A., Larsen, M.J., Wang, Z.: Bull. Amer. Math. Soc. 40, 31 (2003)
35. Aguado, M., Brennen, G.K., Verstraete, F., Cirac, J.I.: Phys. Rev. Lett. 101, 260501 (2008)
36. Brennen, G.K., Aguado, M., Cirac, J.I.: New Jour. Phys. 11, 053009 (2009)
37. Wootton, J.R., Lahtinen, V., Pachos, J.K.: Proceedings of Theory of Quantum Computation, Communication and Cryptography, arXiv:0906.2748 (2009)
38. Jones, V.F.R.: Bull. Amer. Math. Soc. 12, 103 (1985)
39. Kauffman, L.H.: Knots and Physics. World Scientific, Singapore (1991)
40. Jaeger, F., Vertigan, D.L., Welsh, D.J.A.: Math. Proc. Cambridge Phil. Soc. 108, 35 (1990)
41. Freedman, M.H., Kitaev, A., Larsen, M.J., Wang, Z.: Bull. Amer. Math. Soc. 40, 31 (2003)
42. Aharonov, D., Jones, V., Landau, Z.: Algorithmica 55, 395 (2009)
43. Alexander, J.W.: Proc. Nat. Acad. Sci. 9, 93 (1923)
44. Kauffman, L.H., Lomonaco, S.: quant-ph/0606114 (2006)
45. Hamma, A., Ionicioiu, R., Zanardi, P.: Phys. Lett. A 337, 22 (2005)
46. Kitaev, A., Preskill, J.: Phys. Rev. Lett. 96, 110404 (2006)
47. Levin, M., Wen, X.-G.: Phys. Rev. Lett. 96, 110405 (2006)
48. Castelnovo, C., Chamon, C.: Phys. Rev. B 76, 184442 (2007)
49. Iblisdir, S., Perez-Garca, D., Aguado, M., Pachos, J.K.: Phys. Rev. B 79, 4303 (2009); to appear in Nuc. Phys. B, arXiv:0812.4975 (2008)
50. Chung, S.B., Yao, H., Hughes, T.L., Kim, E.-A.: arXiv:0909.2655 (2009)
51. Pastawski, F., Kay, A., Schuech, N., Cirac, I.: arXiv:0911.3843 (2009)
52. Hamma, A., Castelnovo, C., Chamon, C.: Phys. Rev. B 79, 245122 (2009)
53. Chesi, S., Roethlisberger, B., Loss, D.: arXiv:0908.4264 (2009)
54. Stace, T.M., Barrett, S.D., Doherty, A.C.: Phys. Rev. Lett. 102, 200501 (2009)

55. Raussendorf, R., Bravyi, S., Harrington, J.: Phys. Rev. A 71, 062313 (2005)
56. Bravyi, S., Kitaev, A.: Phys. Rev. A 71, 022316 (2005)
57. Nayak, C., Simon, S.H., Stern, A., Freedman, M., Das Sarma, S.: Rev. Mod. Phys. 80, 1083 (2008)
58. Tsui, D.C., Stormer, H.L., Gossard, A.C.: Phys. Rev. Lett. 48, 1559 (1982)
59. Read, N., Green, D.: Phys. Rev. B 61, 10267 (2000)
60. Ivanov, D.A.: Phys. Rev. Lett. 86, 268 (2001)
61. Kane, C.L., Mele, E.J.: Science 314, 1692 (2006)
62. Doucot, B., Ioffe, L.B., Vidal, J.: Phys. Rev. B 69, 214501 (2004)
63. Gladchenko, S., Olaya, D., Dupont-Ferrier, E., Doucot, B., Ioffe, L.B., Gershenson, M.E.: arXiv:0802.2295 (2008)
64. Pachos, J.K., Wieczorek, W., Schmid, C., Kiesel, N., Pohlner, R., Weinfurter, H.: New J. Phys. 11, 083010 (2009)
65. Lu, C.-Y., et al.: Phys. Rev. Lett. 102, 030502 (2009)
66. Duan, L.-M., Demler, E., Lukin, M.D.: Phys. Rev. Lett. 91, 090402 (2003)
67. Micheli, A., Brennen, G.K., Zoller, P.: Nature Physics 2, 341 (2006)

Author Index