Formal Analysis of Artificial Collectives using Parametric Markov Models

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor of Philosophy by

Paul Gainer

June 2020

Abstract

There are many potential applications for the deployment of distributed systems composed of identical autonomous agents such as swarm robotic systems or wireless sensor networks, including remote monitoring, space exploration, or environmental clean up. Such systems need to be robust, and the loss of a small number of agents should not compromise the effectiveness of the system as they will often operate in hostile environments where individual members of that system may suffer failures, or communication may be hindered. To address this, these artificial systems are often designed to imitate the behaviour of self-organising systems found in nature, where simple reactive behaviours for individual members of a system can lead to complex global behaviours, and the collective remains robust to the loss of individuals.

Despite much research being conducted into the development and evaluation of these systems, the industrial application of these technologies is still low. This issue could be addressed by further demonstrating that they can reliably, and predictably, achieve given objectives. Designing such systems is challenging, and often detailed simulations are developed for their analysis. Simulations give invaluable insight into the behaviour of such a system, however, there are often corner cases that might be overlooked. By developing a formal model of the system using some appropriate formalism, mathematical techniques can be applied during development to ensure that the system behaves correctly

with respect to some given specification. These dynamic and inherently stochastic systems can be modelled as Markov processes; memoryless stochastic processes whose behaviour at any moment in time is determined solely by their current state. Model checking is an algorithmic technique to exhaustively check that a representation of a system as a Markov process exhibits some desirable property; furthermore, such an analysis can be extended to analyse systems whose parameters may not be known in an advance. However, the analysis of formal models of large systems is limited due to the resources that are required for their analysis: the size of the model may grow exponentially with the size of the system, and the subsequent analysis may prove to be impossible due to hardware or time constraints.

This thesis investigates the suitability of parametric Markov models for the analysis of swarm robotic systems and wireless sensor networks. The analysis of such models is costly in terms of the size of the formal model representing a system, and the computation time required for its subsequent analysis. Modelling techniques and abstractions are developed for the construction of macroscopic models that abstract away from the identities of individual swarm robots or sensor nodes, and instead focus on the desirable global behaviours of such a system, resulting in smaller formal models. New techniques are then introduced to facilitate the analysis of large families of such models, where similarities between models who share some parameter values are exploited to speed up their analysis. In addition, new representations for such models are developed that allow for larger models to be analysed, and also significantly reduce the time required for that analysis.

Acknowledgements

Firstly, I would like to thank my supervisors Clare Dixon, Ullrich Hustadt, and Michael Fisher for their guidance and support throughout my studies. I would also like to thank my advisors Sven Schewe, Davide Grossi, and Dominik Wojtczak for their encouragement and advice, and many talks over (much) coffee. I am very grateful for all of the constructive feedback I received from my examiners Alice Miller and Boris Konev, who kindly took the time to read through this work.

I am grateful to all other members of the Computer Science department at the University of Liverpool for both their support and companionship, and for making it such a pleasant place to work. I have met many great colleagues over the past few years who have become firm friends, and I was lucky enough to share this journey with you all. Richard, Elisa, Fabio, Sven, Francesco, Antony – thank you for all the fun times, and for keeping me sane!

I would like to express my deepest gratitude to my mother, Blodwen, and my father, Rodney, without whose love and support this would not have been possible. Also thank you to my parents-in-law Nahid and Ali, and sister-in-law Mojgan, for making me feel welcome in your family!

Finally, my special thanks to my wife Maryam. I met you on this journey, and now I am lucky enough to finish it with you. You have been my constant source of inspiration

and encouragement when times were hard, and the best of friends to share life with when times were good.

Contents

List of Figures

List of Tables

List of Algorithms

Notation

The following notations and abbreviations are found throughout this thesis:

$\mathbb{B}$	The Boolean domain.
$\mathbb{N}$	The set of natural numbers.
$\mathbb{Z}$	The set of integers.
$\mathbb{Q}$	The set of rational numbers.
$\mathbb{R}$	The set of real numbers.
$\mathbb{R}_{>0}$	The set of positive real numbers.
$\mathbb{C}$	The set of complex numbers.
$\lfloor \cdot \rceil$	The nearest integer function.
$[a, b]$	The closed interval $\{x \in \mathbb{R} \mid a \leqslant x \leqslant b\}$.
(a, b)	The open interval $\{x \in \mathbb{R} \mid a < x < b\}$.
$\mathsf{Dom}(f)$	The domain of the function f.
$f \restriction_{\mathcal{A}}$	The restriction of the function f to the set $\mathcal{A}$.
$f_1 \oplus f_2$	The overriding union of f_1 and f_2, $f_1 \restriction_{\mathsf{Dom}(f_1) \setminus \mathsf{Dom}(f_2)} \cup f_2$.

$x^{(n)}$	The rising factorial $x(x+1)\dots(x+n-1)$.
$\mathbf{P}$	A stochastic matrix.
$\mathbf{R}$	A reward function.
$\mathrm{E}[X]$	Expectation of the random variable X.
DTMC	A discrete-time Markov chain.
PMC	A parametric Markov chain.
$\mathcal{D}$	A DTMC.
$\mathcal{D}_{\mathcal{X}}$	A PMC.

Chapter 1

Introduction

There are many potential real-world applications for autonomous multiagent systems with distributed control, and they have been the focus of intense study [109]. The interest is well-justified: in the future such systems could be deployed to monitor the condition of hazardous environments, to complete tasks that require miniaturisation such as distributed sensing tasks in the human body, to provide a cheap solution for agricultural or mining tasks, to contribute to space exploration, or to assist with cleaning up environmental waste. These systems will need to be robust, as they will have to function in highly dynamic and potentially hostile environments, where individual agents may malfunction or suffer damage, and where communication channels may be unreliable. To achieve this, much work has been conducted to design control algorithms and protocols for these systems [17] that imitate the behaviour of distributed systems found in nature, for example colonies of ants. Individual members of these natural systems often behave according to a set of simple, reactive rules. However, these simple interactions can lead to complex global behaviours.

The desirable emergent properties of artificial systems that imitate natural systems are

often investigated by designing and analysing detailed simulations [48]. Simulations give invaluable insight into the behaviour of such a system, however there are often corner cases that might be overlooked. Formal verification is the process of mathematically proving or disproving properties of a system. These methods are applied during the development of highly critical systems where correctness of the system is essential. The application of formal verification techniques during the development of robot swarms would complement their simulation and testing, increasing their safety and reliability. Despite much research being conducted into the development and evaluation of robot swarms [17, 18, 29, 52, 109], the industrial application of these technologies is still low. This issue could be addressed by further demonstrating that robot swarm systems can reliably, and predictably, achieve given objectives.

1.1 Artificial collectives

Artifical collectives are systems composed of multiple autonomous agents, mobile robots, or network nodes, that often take inspiration from swarm systems in nature [109]. Swarm intelligence is the study of natural and artificial collectives consisting of multiple individuals, where the coordination of the system is decentralised and local interactions between individuals lead to emergent collective behaviours and self-organisation [17]. Such swarms are often *homogeneous*, without any defined hierarchy, and usually exhibit *reactive*, memoryless behaviour. For our purposes, members of the swarm are *anonymous*, and only the collective emergent behaviour of the swarm is of interest.

Decentralised control and communication algorithms based on swarming behaviour have been developed and widely deployed for both swarm robotic systems and wireless sensor networks. Swarm robotics combines the principles of swarm intelligence and robotics to develop multi-robot systems with decentralised control [11, 132]. Individual swarm

robots are generally low-cost hardware platforms with simple control algorithms where coordination between members of the swarm is achieved through self-organization via local interactions. These local interactions between individual robots should lead to complex global behaviours [17, 132]. The decentralised nature of robot swarms aids scalability and fault-tolerance, making them an ideal solution for deployment in areas where direct human control is infeasible. There are many potential real-world applications for robot swarm technologies, including post-disaster relief [77] and space exploration [34].

There are many examples found in nature of decentralised systems that solve complex problems. Many species of insects, birds and fish exhibit behaviours that lead to a swarm of such creatures being more robust. For example, colonies of ants deposit pheromones along paths when foraging for food. Paths leading to better food sources are proportionately reinforced with more pheromone which results in collective decisions that determine the best food sources [8]. A common approach in swarm robotics has been to develop control algorithms based on abstractions of these natural systems. In particular, much work has been conducted to develop control algorithms based on the behaviours of social insects, such as foraging for food [29, 110], cooperative nest building [141], and efficient distribution of labour[18]. Designing such algorithms for swarms is challenging. Simple control and communication mechanisms are designed at the microscopic level, and both the interactions between members of the swarm, and interactions between swarm members and their environment, should result in desired emergent behaviours at the macroscopic level.

Many distributed swarm algorithms have analogues in wireless sensor network protocols. A wireless sensor network (WSN) is composed of some number of physically disconnected sensors, often referred to as nodes, that use wireless communication to collect and disseminate data, and to organise their activity [130]. The autonomous, spatially dis-

tributed sensors monitor conditions in their environment, and have applications in many domains including environmental sensing, industrial monitoring, health monitoring, home applications, and threat detection [1].

The capabilities of individual nodes in a network vary according to the application. Many applications impose constraints on the cost and size of nodes, resulting in corresponding constraints on their hardware capabilities and resources [146, 135]. Deployed networks are often isolated, and individual nodes with limited energy capacity might have to operate for long periods of time before their batteries can be replaced. The optimisation of node power consumption is a critical design consideration.

When data has been collected by a node it is transmitted through the network. The transmissions are either transported along the most cost-effective multi-hop route in the network until they arrive at some target destination nodes, known as sink nodes (routing), or are broadcast, resulting in propagation throughout the whole network (flooding) [1]. Similar to swarm robotic systems, nodes in a WSN are often homogeneous, with no defined hierarchy. However, a sink node may sometimes have augmented energy capacity or sensing and communication capabilities, if these are required for its distinguished role in the network [109].

Swarm intelligence has also been widely applied for the optimisation of the performance of WSNs. Examples include algorithms for determining optimal routes between source and destination nodes in a network based on the pheromone-based foraging behaviour of ants [52], and protocols that minimise the power consumption of nodes in a network by coordinating their activity in a similar manner to that observed in synchronising firefly colonies [95, 147].

Distributed control paradigms for artificial collectives offer several advantages over more traditional design patterns, including scalability, reliability, and adaptability. However,

one of the main obstacles for their implementation and deployment is the difficulty of identifying the optimal configuration of system parameters that will result in the desirable global behaviour being exhibited [109].

1.2 Formal analysis of artificial collectives

Formal methods are a collection of tools and techniques that are used to specify and analyse systems mathematically. System designers can construct a formal specification of a system — an unambiguous, and mathematically rigorous prescription of its expected behaviour — that can be used to thoroughly check that the system exhibits desirable properties. Given a property for a system specified in a formal language, formal verification is the process of determining whether that property holds in the formal model of the system. Manual proof constructions using axioms and formal inference rules may be difficult, tedious, and require creativity [36]. Automated theorem proving and model checking [35] are verification techniques which attempt to automate this process.

The input for automated theorem proving is a set of axioms and some theorem statement expressed in a suitable formal logic. From this input the theorem prover derives new logical consequences according to a set of deductive rules, and produces either a proof showing that the theorem statement is a logical consequence of the axioms, or a refutation of the theorem statement. The input for model checking is a formal model, usually a finite-state automaton whose states and transitions model the behaviour of the system of interest, and some desirable property expressed in a temporal logic. Temporal logics are classical propositional or first order logics extended with additional operators to reason about propositions in terms of time. The model checker algorithmically and exhaustively checks this property against all possible execution paths of the system from some distinguished initial state, and either confirms that the property holds in the system or produces

a trace showing a possible execution of the model that violates the property.

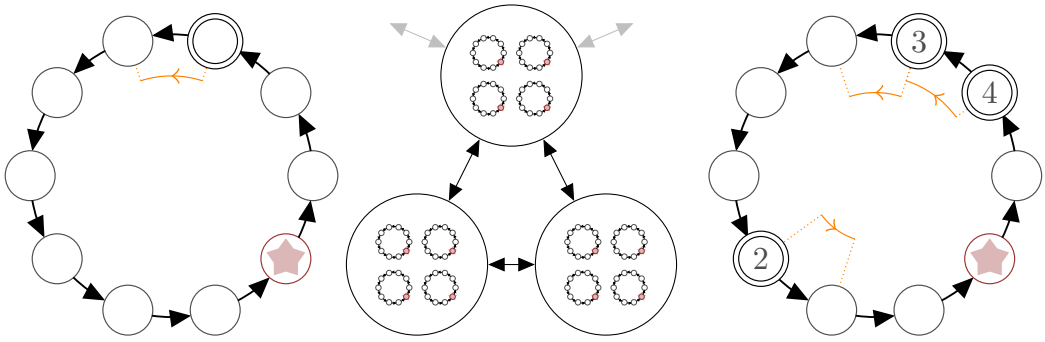
Formal techniques, in particular model checking, have already been applied to formally investigate the emergence of desirable behaviour in artificial collectives such as robot swarms and and wireless sensor networks [23, 32, 48, 80, 97, 96]. Emergent properties of artificial collective systems are often investigated by designing and analysing simulations, or by conducting real world experiments. Experiments and simulation can give detailed insight into how the global behaviour of such a systems changes over time. The formal analysis of the behaviours of artifical collectives can complement the design of algorithms by revealing potential problems that may go unnoticed by empirical analysis [48]. Recently, work has been conducted to develop property-driven top-down design techniques that can be used to facilitate the engineering of swarm systems, reducing the need for simulation or extensive testing [23].

In [9] deductive verification was applied to prove properties of the foraging behaviour of a swarm of mobile robots. Statistical runtime verification combined with agent-based simulation was used to determine the likelihood of emergent swarm behaviours in [80]. Dixon et al. [48] used algorithmic verification techniques to analyse and refine swarm aggregation. An agent-based approach using epistemic temporal logic was employed by Kouvaros and Lomuscio in [97] to specify and verify the emergence of desirable properties in swarms, and in [96] Konur et al. conducted a probabilistic analysis of population-based swarm models. While the techniques used in [48, 9] showed that temporal verification could indeed be applied to prove properties of swarms, verification of properties was only possible for models of small swarms due the exponential blow up in the size of the model as the size of the swarm was increased. To avoid this combinatorial explosion a different approach was adopted in [80], where statistical analysis was applied to runs of a model generated by an agent-based simulator. It was demonstrated that larger systems could be

analysed using this technique, however the results lacked verification strength since only a subset of all possible runs of the system were considered.

The number of possible configurations for large sensor networks has also limited the feasibility of formal techniques for their analysis. Chen et al. reviewed how different methods may be used to investigate ad-hoc routing protocols [32], suggesting that model checking is suitable for small networks, while analytical methods are necessary for larger networks. Höfner and Kamali [82] used statistical model checking to analyse a routing protocol for a network of sixteen nodes. Yue and Katoen [151] used probabilistic model checking to optimise the energy consumption of a leader election protocol in networks of up to nine nodes. Probabilistic model checking was also used by Fehnker and Gao [57] to analyse flooding and gossip protocols. While they were able to use model checking to analyse networks with at most eight nodes, it was necessary to use Monte Carlo simulations for the analysis of larger networks.

To formally model the behaviour of an artificial collective an automaton is often constructed that models the behaviour of some individual robot or node. Given a population of such nodes a naive approach would be to construct a model of the system as the product of all individual automata. For example, consider the automaton shown in Figure 1.1a that represents the behaviour of an entity with cyclic behaviour, where each circle represents a different state of the entity, and each arrow represents a transition between two states. Figure 1.1b shows three possible states of the automaton that would result from taking the product of four individual automata, each representing a single entity with the aforementioned cyclic behaviour. This approach results in very large models, even for low population sizes, for which it is infeasible to check any properties. In general, if each of n entities has k distinct states, then the product automata obtained will have k^n distinct states [36]. Developing macroscopic *population models*, where the homogeneity of entities



(a) (b) (c)

Figure 1.1: (a) A microscopic finite state automaton modelling the cyclic behaviour of an individual agent. (b) Three states of the composite automaton obtained by taking the product of individual microscopic models for four interacting agents. (c) A single state of the macroscopic model of four interacting agents, where counters are used to record agents sharing the same state.

in the system is exploited, leads to smaller models that require less resources to analyse, at the expense of losing information about the state of any distinguished node. This is achieved by using counter abstractions, where variables are introduced that record the number of entities that share the same state. Each counter records the number of entities sharing some common state, and the sum of all counters is equal to the population size. For example, Figure 1.1c illustrates the use of counters to represent the state of nine behaviourally identical entities, where the cyclic behaviour of each entity can be described by the automaton in Figure 1.1a. The population is partitioned into three groups of sizes 4, 3, and 2, where all entities in a group share the same state. The global state of such models can be represented as a vector where each element of the vector records the number of entities sharing some state. In general, given n entities with k states, the resulting population model will have a number of global states given by the multinomial coefficient $\binom{n+k-1}{k-1}$ [24].

More recent work by Kouvaros and Lomuscio [97, 111] has made some initial steps

towards addressing the open problem of state explosion encountered when verifying properties of swarm systems of any substantial size. A counter abstraction technique was proposed to verify properties of swarms independently of their size by finding a cut-off point for the size of the swarm after which the property holds, however the problem of finding this point becomes intractable if the cut-off point is high. Brambilla et al. constructed macroscopic models of swarm robot systems in [23], and stochastic properties of the systems expressed in a probabilstic temporal logic [75] were checked against prescriptive models of how the system should behave. The results obtained from model checking were then used to refine the models of the system.

Stochastic approaches are suitable for the modelling and formal analysis of artificial collectives, as uncertainty is inherent in the real world environments in which they are deployed [109]. Additional noise is often present in the system from many sources; actuators for mobile robots may be hindered by friction, and messages transmitted by wireless sensor networks may be lost in the communication medium, since such systems are often expected to operate under adverse environmental conditions. Furthermore, it is often the case that stochasticity is intentionally introduced into such systems in order to bring about some desirable global behaviour (see for example, Chapter 3). These inherently stochastic systems are often modelled as stochastic processes, in particular discrete-time Markov chains [23, 25, 30, 96, 109, 111, 112]. This formalism is appropriate because the reactive behaviour of such systems directly satisfies the Markov property; the actions of individuals in the system are based solely on their current state, and the current state of their environment.

A discrete-time Markov chain can be represented as a finite set of states and directed transitions. The system has some distinguised starting state, and evolves over time by transitioning to a state that is selected probabilistically from the set of all possible successor

states. Traditionally, each transition is labelled with some constant value corresponding to the probability that the system will take that transition. Parametric Markov chains are a generalisation of Markov chains introduced in [41, 104] and refined in [70], where expressions over some set of model parameters can be used to label model transitions.

Parametric Markov chains are suitable for modelling artificial collectives where values for the parameters of the system may not be known. One might, for example, want to model the likelihood of a message being transmitted by a wireless sensor network node being lost due to some external environmental factors. If the exact conditions under which that network will operate are not known, then an exact value for the probability of message loss cannot be determined a priori. Such parameters can be left open in the model, and desirable properties of the model can be checked using parametric model checking techniques [70]. Parametric analysis can play an important role in the analysis of such systems, as it facilitates investigation into the effects of adjustments to a parameter on the global behaviour of the model. The results of parametric analysis can be used to determine optimal settings for a system, or to adjust a formal model to more closely match observations of a real system. While parametric Markov chains are appropriate and useful for the analysis of artificial collectives, their analysis is costly. The construction of representations for these models is often impeded by state-space explosion, and exploring the parameter space of parameter-wise different families of these models is time consuming.

1.3 Thesis contributions

This thesis aims to address the following research questions:

1. Can parametric Markov models be used to efficiently reason about artificial collectives?

2. Can new techniques be developed to facilitate the parametric analysis of large families of parametric Markov models?

3. How can the efficiency of existing techniques for the parametric analysis of families of parametric Markov models be improved?

The analysis of parametric models is costly in terms of model size and computation time. To address the first research question, case studies are conducted in Chapters 3, 4 and 5 that demonstrate how state-space explosion can be alleviated by constructing appropriate macroscopic models of such systems, hence allowing larger systems to be analysed. The second research question is addressed in Chapter 6, where structural similarities between parameterwise-different instances of a model are exploited to speed up their analysis. Finally, the third research question is addressed in Chapter 7, where new representations for the rational functions labelling the transitions of parametric Markov models are introduced, and are shown to integrate with, and improve the efficiency of, existing techniques for their analysis.

An initial case study is presented in Chapter 3 in which we investigate the behaviour of a swarm of flying micro-air vehicles, deployed to find some target in an unknown environment. A population model is used to record the number of swarm members located at different locations in an abstraction of the environment. Formal models of a system of micro-air vehicles are constructed given parameters determining the size of the swarm, the maximum explorable distance, and a constant determining the probability of a member of the swarm choosing a previously explored path over an unexplored path. The models are procedurally generated for different sets of parameters using a script. Quantitative analysis of the models determines the probability of the swarm locating the target user within some given time bound, and also the expected deployment time for a swarm to guarantee locating the user. The models are then validated by comparing the results to those found in simulations.

The synchronous flashing of fireflies has inspired many coordination protocols for both swarms and WSNs. The ideas developed in the initial case study are further developed in Chapter 4, and applied to develop a generalised population model for these nature-inspired synchronisation protocols. Again, models are generated using a script for different sets of parameters pertaining to the number of nodes, the granularity of temporal abstraction, the degree of uncertainty for communication between nodes, and other parameters defining the model of synchronisation to be used. The models are analysed with respect to the probability of a system synchronising, and the time taken for that system to synchronise. Results for two different models of synchronisation are compared, and results directly comparable to those found in simulations are obtained. The population model is then refined to allow for the analysis of larger networks of synchronising nodes. The work presented in Chapter 5 corresponds to a further extension of this work, and focuses on annotating the models with rewards corresponding to the energy that would be consumed by the network over time. In addition, a metric for synchronisation is derived from other work on pulse-coupled oscillators.

Each aforementioned case study consists of exploring the parameter space for families of parametric probabilistic models. Parameters investigated are typical of these considered when evaluating distributed systems, for instance the number of interacting entities, or the likelihood of interactions between entities resulting in some desirable global behaviour. The effects of parameters that do not change the underlying structure of a model are well-studied, however parameters that induce structural change have received less interest. When analysing hundreds of parameterwise-different, yet similar, models with such parameters, the time and resources taken can be reduced by re-using as much of the analysis as possible at each step. Some parameters result in dramatic changes to the structure of the underling model; others result in only minor changes where much of the information

obtained from the analysis of the previous instance can be re-used for its analysis. New algorithms for the incremental analysis of such models are developed and applied to the previous synchronisation protocol case study, along with a prototypical implementation illustrating the approach. The results of this work are presented in Chapter 6.

When changes to model parameters do not result in changes in the structure of a model, for instance when investigating the effect of environmental effects on communication in the second case study, a wide variety of existing techniques and tools can be used. However, when analysing thousands of parameter instantiations the analysis is often slow. The application of parametric model checking yields a closed-form expression over the parameters of the model that corresponds to some simple temporal property or expected reward, and can be evaluated for different parameter instances. Traditionally, these closed-form expressions (rational functions) have been represented as co-prime rational polynomials. During the computation of the closed-form expression expensive methods often have to be used to simplify the representations (cancel common factors of the rational function), that can grow in size and slow down the analysis. When using directed acyclic graphs to represent these expressions, expensive methods are not necessary to cancel out common factors, and common sub-expressions can be shared by different formulae. This approach, and the speed up obtained when analysing models from previous case studies, is described in detail in Chapter 7.

1.4 Related formal approaches for macroscopic analysis

A model of computation for the interactions of a population of identical finite-state agents is presented by Anluin et al. in [5]. The model consists of *population protocols* and *population configurations*. Given a finite set of all possible configurations for an agent (states), a population configuration is a multiset of elements of that set, specifying the

state of each member of the population. A population protocol is the set of all possible configurations for an agent along with functions to map input and output alphabets to those states, and a relation describing how the model transitions from one population configuration to another when agents communicate. In contrast to the models presented in Chapters 3, 4 and 5, communication in population protocols is always between two agents, where one agent initiates the communication and the other responds. Furthermore, even though the agents cannot identify the other agents in the network, within the global model each agent is uniquely associated with a state. In the models presented here, two different agents sharing the same state cannot be distinguished, even at the global level. Finally, the agents modelled in the case studies may change their state without interacting with other agents, while the agents in a population protocol must communicate with another agent to change their internal state.

Other techniques have been used to model populations of processes. For example population-based models using PEPA, a stochastic process algebra, are discussed in [81]. The work introduces the modelling of individuals using PEPA, and if the identification of individuals is not necessary a population-based approach is advocated to allow larger populations to be modelled. Unlike the approaches discussed later in this thesis, the population-based models make use of an asymptotic approximation of the discrete behaviour. Usually, these approximations are referred to as *mean field approximations* [21], and these can be separated into two main classes. The first class considers systems where the different components of that system evolve asynchronously. A model of the system is constructed as a discrete-time Markov chain (DTMC), by sampling the state of the system at some fixed, constant rate. Adjusting the sample rate regulates the granularity of local transitions in the model, and by setting it to a sufficiently high value some constant number of local transitions for each component between observations can be guaranteed. Increasing

the number of components in the system increases the density of local transitions, and in the limit the behaviour of the system becomes continuous and can be characterised as a set of ordinary differential equations. The second class of mean field approximations considers synchronous systems where different components of the system evolve synchronously, such as those presented in this thesis in Chapters 3, 4, and 5. As the number of components for such a system are increased the density of local transitions at each step remains constant, and the limit of the sequence of DTMCs will also reside in discrete time, resulting in a set of difference equations that asymptotically approximate the system. Recently, such techniques have been shown to be useful for checking temporal, probabilistic properties similar to those investigated in this thesis, relating to the expected behaviour of homogeneous distributed systems [105, 106]. However, they are better suited for the analysis of systems where the number of components is several orders of magnitude larger than those normally investigated for swarm robotic systems or wireless sensor networks, as the approximation of the collective behaviour is obtained in the limit of the number of components of the system, and relies on the observation that random, individual choices leading to noise in the system become increasingly irrelevant as more individuals interact [21].

Stream X-machines are finite state machines extended with internal memory, where transitions between states are labelled with functions over an input stream and the internal memory state [53, 107]. Communicating stream X-machines are stream X-machines that can communicate via messages sent over some shared medium, and different variants have been proposed for the formal modelling of agent-based systems [39]. In particular, communicating stream X-machines were used by Gheorge et al. [67] to model the collective foraging behaviour of a bee colony, and by Chen et al. [31] to identify emergent properties in multiagent simulations. In contrast to parametric Markov chains there is little tool support for the analysis of X-machines [131], despite the development of appropriate model

checking algorithms [54], and proposed stochastic extensions [116].

Vector addition systems, introduced by Karp and Miller in [92] and generalized to vector addition systems with states (VASS) by Hopcroft and Pansiot [83], can be used to represent and analyse parallel processes, and are equivalent to Petri nets. A VASS consists of a set of states labelled with integer vectors (known as configurations). Transitions between states are also labelled with integer vectors of the same size, and one state is reachable from another if the sum of the configuration vectors of the old state and the transition is equal to the configuration vector of the new state. Probabilistic vector addition systems with states (PVASS), equivalent to discrete-time stochastic Petri nets [118], are stochastic extensions of vector addition systems where transitions between states are weighted, and the probability of a given transition is determined by dividing its weight by the total weight of all enabled transitions [26]. A PVASS is *structurally bounded* if the set of states reachable from any initial configuration is finite [124]. Each of the models developed and analysed in the case studies presented in Chapters 3, 4 and 5 could be encoded as a structurally bounded PVASS, as in each case the configuration of the system of interest is encoded as an integer vector. However, our models are encoded as a parametric DTMC, an instantiation of the more general PVASS, which was preferable as more specialised tools are available for their analysis.

The state graph of a product composition of identical processes (see for example, Figure 1.1b) will often exhibit substantial symmetry. Symmetry reduction is a technique introduced by Clark et al. in [38] that can be used in model checking to ameliorate state-space explosion by exploiting structural replication in a system. Given a graph representing the product of multiple, identical processes that exhibits such symmetry, a new quotient structure is constructed by identifying sets of states that are equivalent with respect to an automorphism of the graph. Temporal properties of interest can then be checked against

the smaller quotient structure, however the number of equivalence classes is exponential in the number of processes. This was addressed in [55] by Emerson et al. through the use of *generic representatives*, where the behaviour of multiple processes was represented generically using a set of counters that recorded the number of processes that are in each state, but abstracted away from the identity of each process. This technique was applied to probabilistic model checking by Donaldson et al. in [49], and later generalised in [51]. The formal modelling language Symmetric Probabilistic Specification Language (SPSL) for symmetric systems was introduced, built on the guarded command language of the model checker PRISM [101], that is decribed in detail in Section 2.7.1. The GRIP tool [50] provides an implementation of an algorithm for the translation of a set of SPSL system specifications into to a generic form that can be analysed by a probabilistic model checker such as PRISM.

While is has been demonstrated that GRIP can be used to substantially reduce the size of symmetrical models [50], it was not possible to encode the global semantics of the models introduced in Chapters 4 and 5 in SPSL due to the interactions between processes resulting in a global semantics that differed from typical parallelisation operations (see Section 4.3.1).

1.5 Corresponding publications

In this section the chapters of the dissertation are linked to the related scientific publications.

1. The formal analysis of a bio-inspired algorithm discussed in Chapter 3 was originally presented in [61], where the author developed the formal model and conducted the analysis.

- **Paul Gainer**, Clare Dixon, and Ullrich Hustadt. Probabilistic model checking of ant-based positionless swarming. In *Conference Towards Autonomous Robotic Systems*, volume 9716 of *LNCS*, pages 127–138. Springer, 2016.

2. The development of the formal model of oscillator synchronisation discussed in Chapter 4 was originally presented in [65], where the author developed the formal model and contributed towards its analysis. The further reduction of that model was originally presented in [64], where the author developed the reduced model and contributed towards its analysis.

   - **Paul Gainer**, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. Investigating parametric influence on discrete synchronisation protocols using quantitative model checking. In *International Conference on Quantitative Evaluation of Systems*, volume 10503 of *LNCS*, pages 224–239. Springer, 2017.
   - **Paul Gainer**, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. Multi-scale verification of distributed synchronisation. *Formal Methods in System Design*. Submitted for publication.

3. The investigation into the power consumption of networks of synchronising oscillators conducted in Chapter 5 was originally presented in [66], and the extension of this to reduced models was originally presented in [64], where the author developed the synchronisation metric and reward structures and contributed towards the analysis of the models.

   - **Paul Gainer**, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. The power of synchronisation: formal analysis of power consumption in networks of pulse-coupled oscillators. In *International Conference on Formal Engineering Methods (Best Paper Award)*, pages 160–176. Springer, 2018.

- **Paul Gainer**, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. Multi-scale verification of distributed synchronisation. *Formal Methods in System Design*. Submitted for publication.

4. The development of novel algorithms for the incremental analysis of parametric Markov chains discussed in Chapter 6 was originally presented in [63], where the author devised the algorithms, developed the prototypical tool, and conducted the analysis.

   - **Paul Gainer**, Ernst Moritz Hahn, and Sven Schewe. Incremental verification of parametric and reconfigurable markov chains. In *International Conference on Quantitative Evaluation of Systems*, pages 140–156. Springer, 2018.

5. The proposal of new representations for the labels of transitions in parametric Markov chains, and resulting acceleration of their analysis, presented in Chapter 7 was originally presented in [62], where the author contributed to the devising of the approach and conducted the analyis.

   - **Paul Gainer**, Ernst Moritz Hahn, and Sven Schewe. Accelerated model checking of parametric Markov chains. In *International Symposium on Automated Technology for Verification and Analysis*, pages 300–316. Springer, 2018.

All models, scripts, and results presented in Chapters 3, 4, 5, 6, and 7 are available online[1] or upon request.

[1] https://github.com/PaulGainer/PhDThesis

1.6 Thesis outline

Chapter 2 introduces the formalisms and techniques that are used throughout this thesis. Discrete-time Markov chains and their generalisation, parametric Markov chains, are introduced, and an overview of the logic PCTL and model checking techniques used to analyse such models are discussed. In addition, a summary of the state-based input language developed for the probabilistic model checker PRISM [101], used to specify the models presented in this thesis, is provided.

In Chapter 3, a formal model of a swarming algorithm inspired by the behaviour of foraging ants is developed, and the results of its analysis are presented and discussed. This first case study illustrates how behavioural homogeneity can be exploited when constructing formal models of such systems.

A formal model for networks of nodes synchronising using bio-inspired protocols is developed and analysed in Chapter 4. This second case study again exploits behavioural homogeneity, and yields a highly parametrised family of models that are used as benchmarks to illustrate the effectiveness of new techniques presented in later chapters. The analysis is extended in Chapter 5, where the power consumption of such networks is investigated, and new metrics for synchronisation are introduced.

Chapter 6 introduces new algorithms for the incremental analysis of parametric Markov chains, that exploit similarities in the underlying graphs of parameterwise-different models to accelerate their analysis. In particular, the technique is shown to be effective when exploring the parameter space of the family of models introduced in Chapters 4 and 5.

The acceleration of model analysis achieved in Chapter 6 is complemented by the work presented in Chapter 7, where new representations for the transition labels of parametric Markov chains are proposed. The implementation of these representations into a probabilistic model checker, and the subsequent empirical analysis of both the models presented

in previous chapters and others taken from the literature, illustrate a further increase in model analysis times, and facilitate the parametric analysis of larger models.

In Chapter 8 the thesis is summarised, and suggestions are given for future research directions.

Chapter 2

Preliminaries

This chapter introduces the formalisms and techniques that will be used throughout the rest of this thesis. First discrete-time Markov chains (DTMCs) are formally defined – stochastic processes that can be used to model the evolution of discrete systems. The definition is then extended to parametric Markov chains (PMCs), a generalisation of DTMCs where parameters determining the stochastic evolution of the system are unknown. The annotation of these models with rewards is then discussed. An appropriate formalism for reasoning over discrete stochastic processes, Probabilistic computation tree logic (PCTL), is then defined. Algorithms for model checking PCTL over a DTMC are introduced first, followed by algorithms for model checking properties over a PMC. The final two sections give a brief overview of statistical methods that can be applied to obtain approximate model checking results, and introduce the available tools that were used to to obtain the results that are presented in this thesis.

2.1 Discrete-time Markov chains

Discrete-time Markov chains are stochastic processes with discrete space and discrete time. The evolution of the system at any moment in time can be represented by a discrete probabilistic choice over several outcomes [100].

Definition 1. *A discrete-time Markov chain $\mathcal{D}$ is a tuple $(\mathcal{S}, s_0, \mathbf{P}, L)$ where $\mathcal{S}$ is a finite set of states. s_0 is the initial state, and $L\colon \mathcal{S} \to \mathbb{P}(\mathcal{L})$ is a labelling function that assigns properties of interest from a set of labels $\mathcal{L}$ to states. $\mathbf{P}\colon \mathcal{S} \times \mathcal{S} \to [0,1]$ is the* transition probability matrix *subject to $\sum_{s' \in \mathcal{S}} \mathbf{P}(s,s') = 1$ for all $s \in \mathcal{S}$, where $\mathbf{P}(s,s')$ gives the probability of a transition from s to s'. There is a transition between two states $s, s' \in \mathcal{S}$ if $\mathbf{P}(s,s') > 0$.*

Intuitively, a DTMC $\mathcal{D}$ is a state transition system where transitions between states are labelled with the probability of that transition being taken. If there are no properties of interest with which to label states the labelling function will often be omitted from the tuple.

Definition 2. *Given a DTMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$, the* underlying graph *of $\mathcal{D}$ is given by $\mathcal{G}_{\mathcal{D}} = (\mathcal{S}, \mathcal{E})$ where $\mathcal{E} = \{(s,s') \mid \mathbf{P}(s,s') \neq 0\}$.*

Given a state $s \in \mathcal{S}$, the sets of all immediate predecessors and successors of s in the underlying graph of $\mathcal{D}$ are denoted by $\mathsf{pre}_{\mathcal{D}}(s)$ and $\mathsf{post}_{\mathcal{D}}(s)$, respectively. If s' is reachable from s in the underlying graph of $\mathcal{D}$ then this is denoted by $\mathsf{reach}_{\mathcal{D}}(s,s')$. Given a set of states $\Omega \subseteq \mathcal{S}$ the term $\mathsf{reach}_{\mathcal{D}}(s,\Omega)$ denotes that $\mathsf{reach}_{\mathcal{D}}(s,s')$ holds for some state $s' \in \Omega$. These are simplified to pre, post, and reach if $\mathcal{D}$ is clear from the context.

Definition 3. *Given a DTMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$ with underlying graph $\mathcal{G}_{\mathcal{D}} = (\mathcal{S}, \mathcal{E})$ a* bottom strongly connected component *(BSCC) is a set $\mathcal{B} \subseteq \mathcal{S}$ such that in the underlying*

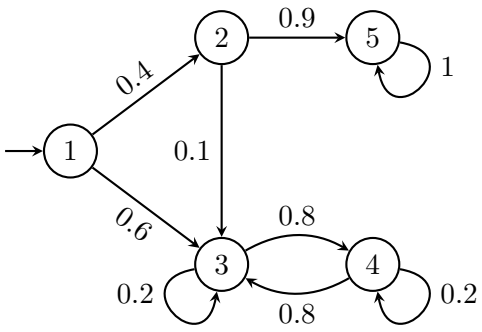


Figure 2.1: An example of a discrete-time Markov chain.

graph each state $s_1 \in \mathcal{B}$ *can reach each state* $s_2 \in \mathcal{B}$ *and there is no* $s_3 \in \mathcal{S} \setminus \mathcal{B}$ *reachable from* s_1*.*

Example 1. *Figure 2.1 shows a DTMC* $\mathcal{D}$ *where* $\mathcal{S} = \{1 \dots 5\}$*,* 1 *is the initial state, and the probability matrix is*

$$\mathbf{P} = \begin{bmatrix} 0 & 0.4 & 0.6 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.9 \\ 0 & 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\mathbf{P}(i, j)$ *is the probability to transition from state* i *to state* j*. The bottom strongly components of* $\mathcal{D}$ *are* $\{3, 4\}$ *and* $\{5\}$*.*

Paths and measures An execution *path* is a non-empty finite, or infinite, sequence of states $s_0 s_1 s_2 \dots$ where $s_i \in \mathcal{S}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for $i \geqslant 0$. For example, 123, 12555…, and 1344344344… are possible paths of the DTMC in Figure 2.1. The i^{th} state of path ω

is denoted by $\omega[i]$, where $\omega[0]$ is the first state along that path.

Definition 4. *The set of all (infinite) paths starting in state s is denoted by* $\mathsf{Paths}^D(s)$*, and the set of all finite paths starting in s by* $\mathsf{Paths}^{\mathcal{D}}_{\mathit{fin}}(s)$*.*

Paths where the first state along that path is the initial state s_0 are simplified to $\mathsf{Paths}^{\mathcal{D}}$ and $\mathsf{Paths}^{\mathcal{D}}_{\mathsf{fin}}$, and furthermore to Paths and $\mathsf{Paths}_{\mathsf{fin}}$ if $\mathcal{D}$ is clear from the context. For a finite path $\omega \in \mathsf{Paths}(s)$ the *cylinder set* of ω is the set of all infinite paths in $\mathsf{Paths}(s)$ that share prefix ω. The probability of a finite path $s_0, s_1, \ldots, s_n \in \mathsf{Paths}_{\mathsf{fin}}(s_0)$ is given by $\prod_{i=1}^{n} \mathbf{P}(s_{i-1}, s_i)$. This measure over finite paths can be extended to a probability measure Pr_s over the set of infinite paths Paths, where the smallest σ-algebra over Paths is the smallest set containing all cylinder sets for paths in $\mathsf{Paths}_{\mathsf{fin}}$. For paths where the first state along that path is the initial state s_0 this is simplified to Pr. For a detailed description of the construction of the probability measure the reader is referred to [93].

Transient probabilities It is often useful to determine the probability that a system is in some specific state after taking some number of steps from an initial state.

Definition 5. *The* transient probability *of being in a given state s' after taking k steps starting from state s is given by*

$$\tau^k(s, s') = \mathsf{Pr}_s\{\omega \in \mathsf{Paths}(s) \mid \omega[k] = s'\}.$$

The transient probabilities can be obtained for all s, s' by computing the matrix power $\mathbf{P}^k$. Each $\tau^k(s, s')$ is then given by $\mathbf{P}^k(s, s')$.

Sub-stochastic models Sometimes only a subgraph of the underlying graph of a DTMC is considered, by setting the probability of one or more transitions in the probability matrix

to zero. This can result in the sum of probabilities labelling outgoing transitions being less than one from some states. For such graphs a state s is

- *stochastic* if $\sum_{s' \in \mathsf{post}(s)} \mathbf{P}(s, s') = 1$,
- *sub-stochastic* if $\sum_{s' \in \mathsf{post}(s)} \mathbf{P}(s, s') \in (0, 1)$,
- *absorbing* if $\mathbf{P}(s, s) = 1$ or $\sum_{s' \in \mathsf{post}(s)} \mathbf{P}(s, s') = 0$.

2.2 Parametric Markov chains

Parametric Markov chains (PMCs) are a generalisation of DTMCs where all aspects of the model are not necessarily fixed, and can depend on parameters of the model [41, 104]. More specifically, transition probabilities make not take concrete values, and can instead be labelled with expressions over a finite set of model parameters. For example, a node in a wireless sensor network that broadcasts a message to other nodes might fail to broadcast its message with probability p, and succeed with probability $1 - p$.

Let $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a finite set of variables over $\mathbb{R}$. A *polynomial* ρ over the set of variables $\mathcal{X}$ is a sum of monomials

$$\rho(x_1, \ldots, x_n) = \sum_{i_1, \ldots, i_n} c_{i_1, \ldots, i_n} x_1^{i_1} \cdots x_n^{i_n},$$

where each $i_j \in \mathbb{N}$ and each $c_{i_1, \ldots, i_n} \in \mathbb{R}$. A *rational function* q over a set of variables $\mathcal{X}$ is a fraction

$$q(x_1, \ldots, x_n) = \frac{\rho_1(x_1, \ldots, x_n)}{\rho_2(x_1, \ldots, x_n)}$$

of two polynomials ρ_1, ρ_2 over $\mathcal{X}$. The set of all rational functions from $\mathcal{X}$ to $\mathbb{R}$ is denoted by $\mathcal{Q}_{\mathcal{X}}$.

Definition 6. *A* parametric Markov chain *is a tuple $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$, where $\mathcal{S}$ is a finite set of states, $s_0 \in \mathcal{S}$ is the initial state, $L\colon \mathcal{S} \to \mathbb{P}(\mathcal{L})$ is a labelling function that assigns properties of interest from a set of labels $\mathcal{L}$ to states, $\mathcal{X} = \{x_1, \ldots, x_n\}$ is a finite set of* parameters*, and $\mathbf{P}$ is the probability matrix $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to \mathcal{Q}_{\mathcal{X}}$.*

Again if states do not need to be labelled with properties of interest the labelling function will be omitted from the tuple.

Example 2. *Figure 2.2 shows a sample PMC where $\mathcal{S} = \{1 \ldots 5\}$, the initial state is 1, and the set of parameters is $\mathcal{X} = \{p, q, r\}$. The probability matrix is*

$$\mathbf{P} = \begin{bmatrix} 0 & p & 1-p & 0 & 0 \\ 0 & 0 & 1-q & 0 & q \\ 0 & 0 & r & 1-r & 0 \\ 0 & 0 & 1-r & r & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

An *evaluation* for $\mathcal{X}$ is a function $\upsilon\colon \mathcal{X} \to \mathbb{R}$. If $\mathsf{Dom}(\upsilon) = \mathcal{X}$ for some evaluation υ then that evaluation is *total*. Given some set of variables $\mathcal{V} \subseteq \mathcal{X}$ and an evaluation υ for $\mathcal{X}$,

$$\mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon]\colon \mathcal{S} \times \mathcal{S} \to \mathcal{Q}_{\mathcal{X} \setminus \mathsf{Dom}(\upsilon)}$$

denotes the probability matrix that is obtained by considering each variable $x \in \mathsf{Dom}(\upsilon)$, and substituting each occurrence of that variable in the range of $\mathbf{P}$ with $\upsilon(x)$.

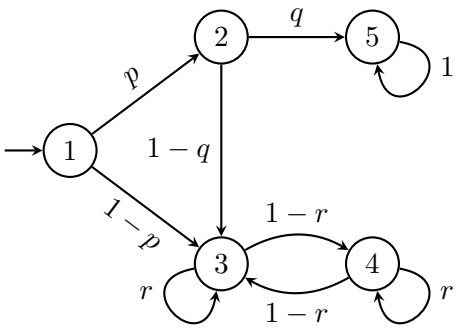


Figure 2.2: An example of a parametric Markov chain.

Definition 7. *Given a PMC* $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ *and* υ*, an evaluation for* $\mathcal{X}$*, let* $\mathcal{X}_\upsilon$ *denote the set* $\mathcal{X} \setminus \mathsf{Dom}(\upsilon)$*. The PMC induced by* υ *is defined as*

$$\mathcal{D}_{\mathcal{X}_\upsilon} = (\mathcal{S}, s_0, \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon], L, \mathcal{X}_\upsilon).$$

A total evaluation υ is *well-defined* if for all $s, s' \in \mathcal{S}$,

$$\begin{aligned} \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon](s,s') &\in [0,1] && \text{for all } s, s' \in \mathcal{S}, \\ \sum_{s' \in \mathcal{S}} \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon](s,s') &= 1 && \text{for all } s \in \mathcal{S}. \end{aligned}$$

Observe that for a total well-defined evaluation the resulting induced PMC is a regular DTMC, since all occurrences of the parameters in $\mathcal{X}$ have been replaced with concrete values. A *strictly* well-defined evaluation υ for some PMC $\mathcal{D}_{\mathcal{X}}$ is a well-defined evaluation such that $\mathcal{D}_{\mathcal{X}_\upsilon}$, the PMC induced by υ, does not differ structurally from $\mathcal{D}_{\mathcal{X}}$. That is, given $\mathcal{G}_{\mathcal{D}_{\mathcal{X}}} = (\mathcal{S}, \mathcal{E})$ and $\mathcal{G}_{\mathcal{D}_{\mathcal{X}_\upsilon}} = (\mathcal{S}, \mathcal{E}_\upsilon)$, the underlying graphs of the original PMC and the induced PMC, υ is strictly well-defined if, and only if, $\mathcal{E} = \mathcal{E}_\upsilon$.

Example 3. *Consider again the PMC shown in Figure 2.2, and let* $\upsilon = \{p \mapsto 0.4, q \mapsto$

$0.9, r \mapsto 0.2\}$ *be a total well-defined evaluation for that PMC. Since* υ *is total all occurrences of parameters are replaced with the concrete values defined by* υ*, yielding the induced DTMC shown in Figure 2.1. Observe that the underlying graph of the PMC and the induced DTMC is the same, and hence* υ *is strictly well-defined.*

2.3 Markov reward models

While probabilistic reachability properties allow models to be quantitatively analysed with respect to the likelihood of reaching some desirable state, they do not allow reasoning about other properties of interest, for instance the expected time taken for a network of WSN nodes to reach some desirable state, or the expected energy consumption of that network. This can be achieved by annotating the states and transitions of the PMC with rewards (respectively costs, should values be negative) that are awarded when states are visited, or transitions taken.

Definition 8. *Given a PMC* $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ *a* reward function $\mathbf{R}\colon \mathcal{S} \cup (\mathcal{S} \times \mathcal{S}) \to \mathcal{Q}_{\mathcal{X}}$ *associates rewards (rational polynomials over* $\mathcal{X}$*) with states and transitions of* $\mathcal{D}_{\mathcal{X}}$*.*

Example 4. *Figure 2.3a shows a sample PMC where* $\mathcal{S} = \{1 \ldots 5\}$*, the initial state is* 1*, and the set of parameters is* $\mathcal{X} = \{p, q, r, s\}$*. The PMC is augmented with a reward function* $\mathbf{R}$ *with* $\mathbf{R}(1) = \mathbf{R}(2) = s$*,* $\mathbf{R}(3) = \mathbf{R}(4) = \mathbf{R}(5) = 1$*,* $\mathbf{R}(1,2) = s$*,* $\mathbf{R}(2,3) = \mathbf{R}(2,5) = 2s$*, and* $\mathbf{R}(i,j) = 0$ *for all other* $i, j \in \mathcal{S}$*.*

For any evaluation υ for a PMC let

$$\mathbf{R}[\mathsf{Dom}(\upsilon)/\upsilon]\colon \mathcal{S} \cup (\mathcal{S} \times \mathcal{S}) \to \mathcal{Q}_{\mathcal{X} \setminus \mathsf{Dom}(\upsilon)}$$

denote the reward function that is obtained by considering each variable $x \in \mathsf{Dom}(\upsilon)$ and

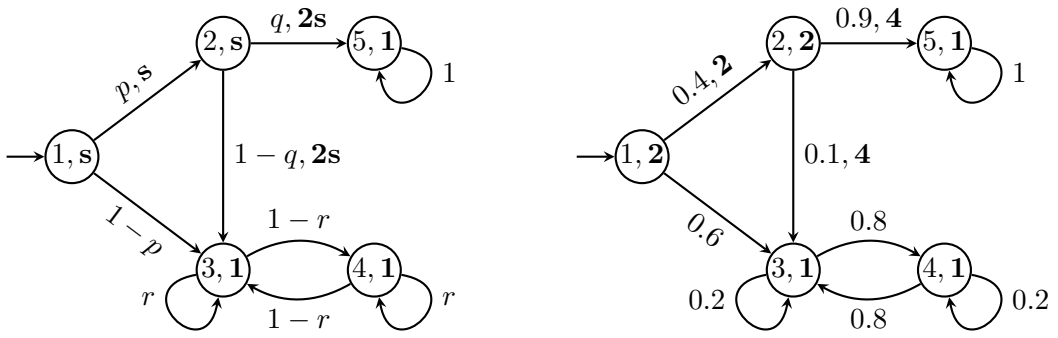


Figure 2.3: A parametric Markov chain with rewards (a), and the DTMC induced by the evaluation $\upsilon = \{p \mapsto 0.4, q \mapsto 0.9, r \mapsto 0.2, s \mapsto 2\}$ (b).

substituting each occurrence of that variable in the range of $\mathbf{R}$ with $\upsilon(x)$. If υ is total then the induced reward function simply maps states and transitions to values in $\mathbb{R}$.

Example 5. *Consider again the PMC in Figure 2.3a augmented with the rewards defined by the reward function* $\mathbf{R}$*, and the evaluation* $\upsilon = \{p \mapsto 0.4, q \mapsto 0.9, r \mapsto 0.2, s \mapsto 2\}$ *for* $\mathbf{R}$*. Figure 2.3b shows the induced PMC for that evaluation. The induced PMC is a DTMC since* υ *is total and strictly well-defined. The induced reward function* $\mathbf{R}[\mathcal{X}/\upsilon]$ *for the DTMC then assigns real values to states and transitions.*

2.4 Probabilistic model checking

Probabilistic Computation Tree Logic [75] (PCTL) is a probabilistic extension of the temporal logic CTL [35], where existential and universal quantification over paths is replaced with a probabilistic operator that allows the specification of properties such as "the network will eventually reach a synchronised state with probability greater than 0.5". The semantics of PCTL is defined over discrete probabilistic systems, and a PCTL formula is

satisfied by a set of possible paths of that system. Probabilistic model checking [145] of a PCTL formula is the computation of the probability measure of this set of paths satisfying the formula, and then comparing the computed value to some given threshold. Firstly the logic PCTL is introduced. Then a summary of techniques for model checking PCTL properties over DTMCS is given.

2.4.1 Probabilistic computation tree logic

Definition 9. *The syntax of PCTL is given by:*

$$\Phi = \mathit{true} \mid l \mid \neg\Phi \mid \Phi \wedge \Phi \mid \mathsf{P}_{\bowtie\lambda}[\Psi]$$
$$\Psi = \mathrm{X}\,\Phi \mid \Phi\,\mathrm{U}^{\leqslant k}\,\Phi$$

where l is an atomic proposition taken from the set of labels $\mathcal{L}$, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $\lambda \in [0,1]$, and $k \in \mathbb{N} \cup \{\infty\}$.

Formulae denoted by Φ are *state formulae* and formulae denoted by Ψ are *path formulae*. A PCTL formula is always a state formula, and a path formula can only occur inside the P operator. The semantics of PCTL over a DTMC are now defined.

Definition 10. *Given a DTMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$, the satisfaction relation $\models$ for any state $s \in \mathcal{S}$ is inductively defined as follows, recalling that Pr is the probability measure over the set of infinite paths:*

$$\begin{aligned}
s &\models l && \Leftrightarrow && l \in L(s)\\
s &\models \neg\Phi && \Leftrightarrow && s \not\models \Phi\\
s &\models \Phi \wedge \Phi' && \Leftrightarrow && s \models \Phi \wedge s \models \Phi'\\
s &\models \mathsf{P}_{\bowtie\lambda}[\Psi] && \Leftrightarrow && \mathsf{Pr}_s\{\omega \in \mathsf{Paths}(s) \mid \omega \models \Psi\} \bowtie \lambda
\end{aligned}$$

where $l \in \mathcal{L}$, and for any path ω of $\mathcal{D}$ as follows:

$$\begin{array}{lcl} \omega \models \mathrm{X}\,\Phi & \Leftrightarrow & \omega[1] \models \Phi \\ \omega \models \Phi\,\mathrm{U}^{\leqslant k}\,\Phi' & \Leftrightarrow & \exists i \in \mathbb{N}(i \leqslant k \wedge \omega[i] \models \Phi' \wedge \forall_{j \in \mathbb{N}}.j < i \implies \omega[j] \models \Phi). \end{array}$$

Disjunction, *false*, and implication are derived as usual, and eventuality is defined as $\mathrm{F}^{\leqslant k}\,\Phi \equiv \mathit{true}\,\mathrm{U}^{\leqslant k}\,\Phi$. The simpler forms $\mathrm{F}\,\Phi$ and $\Phi\,\mathrm{U}\,\Phi'$ are used when $k = \infty$. When model checking formulae of the form $\mathsf{P}_{\bowtie\lambda}$, the actual probability is first calculated, and then compared to the bound [102]. Here the notation $\mathsf{P}[\Psi] = \mathsf{Pr}_s\{\omega \in \mathsf{Paths}(s) \mid \omega \models \Psi\}$ is used to denote this calculated probability.

Example 6. *Returning again to the DTMC shown in Figure 2.1, let* $L = \{1 \mapsto \{a\}, 2 \mapsto \{b\}, 3 \mapsto \emptyset, 4 \mapsto \emptyset, 5 \mapsto \emptyset\}$. *Then trivially,* $2 \models b$, *and* $2 \not\models a$. *The relation* $1 \models \mathsf{P}_{\leqslant 0.4}[\mathrm{F}\,5]$ *(equivalently* $1 \models \mathsf{P}_{\leqslant 0.4}[\mathit{true}\,\mathrm{U}\,5]$*) holds since the set of all infinite paths starting in* 1 *that satisfy* $\mathit{true}\,\mathrm{U}\,5$ *is the singleton* $\{12555\ldots\}$ *and* $\mathsf{Pr}\{12555\ldots\} = 0.4 \cdot 0.9 \cdot 1 \cdot 1 = 0.36$ *which is less than or equal to* 0.4.

Given a DTMC augmented with a reward function $\mathbf{R}$ it is often useful to reason about the reward that is accumulated along a path $\omega = s_0 s_1 s_2 \ldots \in \mathsf{Paths}$ that eventually passes through some set of target states $\Omega \subseteq \mathcal{S}$. Firstly, for any finite path $s_0 \ldots s_k \in \mathsf{Paths}_{\mathsf{fin}}$ let the total reward accumulated along that path with respect to a reward function $\mathbf{R}$, and up to, but not including, s_k be

$$\mathsf{total}_{\mathbf{R}}(s_0 \ldots s_k) = \sum_{n=0}^{k-1} \mathbf{R}(s_n) + \sum_{n=0}^{k-1} \mathbf{R}(s_n, s_{n+1}). \tag{2.1}$$

Secondly, define a random variable

$$X_\Omega \colon \mathsf{Paths} \to \mathbb{R} \cup \{\infty\}$$

over the set of infinite paths to be

$$X_\Omega(\omega) = \begin{cases} \infty & \text{if } \omega_\Omega = \emptyset \\ \mathsf{total}_{\mathbf{R}}(s_0 \dots s_k) & \text{otherwise, where } k = \min\{i \mid s_i \in \Omega\}. \end{cases}$$

Intuitively, X_Ω maps infinite paths of a DTMC to the rewards accumulated along those paths until one of the target states in Ω is reached. If a target state is never reached then the accumulated reward is infinite. The expectation of X_Ω with respect to Pr_s is given by

$$\mathrm{E}[X_\Omega] = \int_{\omega \in \mathsf{Paths}(s)} X_\Omega(\omega) d\mathsf{Pr}_s = \sum_{\omega \in \mathsf{Paths}(s)} X_\Omega(\omega) \mathsf{Pr}_s\{\omega\}.$$

Example 7. *Consider the DTMC shown in Figure 2.3b annotated with the rewards defined by the reward function* $\mathbf{R}$*, and let the set of target states be* $\Omega = \{5\}$*. Then for the infinite path* $12555\dots$

$$X_\Omega(12555\dots) = \mathsf{total}_{\mathbf{R}}(125) = (2+2) + (2+4) = 10.$$

The logic of PCTL can be extended to include reward properties by introducing the state formula $\mathsf{R}_{\bowtie r}[\mathrm{F}\,\Phi]$ where $\bowtie \in \{<, \leqslant, \geqslant, >\}$ and $r \in \mathbb{R}$. Given a state s, a real value r and a PCTL state formula Φ, the semantics are given by

$$s \models \mathsf{R}_{\bowtie r}[\mathrm{F}\,\Phi] \Leftrightarrow \mathrm{E}[X_{\mathsf{Sat}(\mathrm{F}\,\Phi)}] \bowtie r,$$

where $\mathsf{Sat}(\Phi)$ denotes the set of all states that satisfy Φ, and the term $\mathrm{E}[X_{\mathsf{Sat}(\mathrm{F}\,\Phi)}]$ is the expected reward to reach any state in $\mathsf{Sat}(\Phi)$.

When model checking formulae of the form $\mathsf{R}_{\bowtie\lambda}$, the expected reward is first calculated, and then compared to the bound. We use the notation $\mathsf{R}[\Psi] = \mathrm{E}[X_{\mathsf{Sat}(\mathrm{F}\,\Phi)}]$ to denote this expected reward.

2.4.2 Model checking PCTL

The model checking algorithm for PCTL over a DTMC was first presented in [75]. Given a DTMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$ and a PCTL formula Φ the output is the the set of all states of the model that satisfy Φ. The notation $\mathsf{Sat}(\Phi) = \{s \in \mathcal{S} \mid s \models \Phi\}$ is used to denote the set of all states satisfying Φ, and the notation $\underline{\Phi}$ denotes the vector in $\{0,1\}^{|S|}$ where

$$\underline{\Phi}(s) = \begin{cases} 1 & \text{if } s \in \mathsf{Sat}(\Phi) \\ 0 & \text{otherwise.} \end{cases}$$

The algorithm proceeds as per the model checking algorithm for CTL [37], and can be summarised as follows:

$$\begin{aligned} \mathsf{Sat}(\mathit{true}) &= \mathcal{S} \\ \mathsf{Sat}(l) &= \{s \in \mathcal{S} \mid l \in L(s)\} \\ \mathsf{Sat}(\neg\Phi) &= \mathcal{S} \setminus \mathsf{Sat}(\Phi) \\ \mathsf{Sat}(\Phi \wedge \Phi') &= \mathsf{Sat}(\Phi) \cap \mathsf{Sat}(\Phi') \\ \mathsf{Sat}(\mathsf{P}_{\bowtie\lambda}[\Psi]) &= \{s \in \mathcal{S} \mid \mathsf{Pr}_s\{\omega \in \mathsf{Paths}(s) \mid \omega \models \Psi\} \bowtie \lambda\}. \end{aligned}$$

The algorithm for PCTL differs from that for CTL for formulas of the form $\mathsf{P}_{\bowtie\lambda}[\Psi]$. To determine the satisfying set, the probability $\mathsf{Pr}_s\{\omega \in \mathsf{Paths}(s) \models \Psi\}$, which wil be denoted

by $\mathsf{Prob}(s, \Psi)$, must be calculated for every state s in $\mathcal{S}$, yielding the $|\mathcal{S}|$-vector $\underline{\mathsf{Prob}(\Psi)}$. The elements of $\underline{\mathsf{Prob}(\Psi)}$ can then be element-wise compared to the bound $\bowtie \lambda$.

Next operator. For formulas of the form $\mathsf{P}_{\bowtie\lambda}[\mathrm{X}\,\Phi]$ the probability $\mathsf{Prob}(s, \mathrm{X}\,\Phi)$ for a state s can be is given by summation over the probabilities of outgoing transitions from that state to states that satisfy Φ, formally

$$\mathsf{Prob}(s, \mathrm{X}\,\Phi) = \sum_{s' \in \mathsf{Sat}(\Phi)} \mathbf{P}(s, s').$$

The vector of probabilities for all states, $\underline{\mathsf{Prob}(\mathrm{X}\,\Phi)}$, can be obtained by the matrix-vector multiplication $\underline{\mathsf{Prob}(\mathrm{X}\,\Phi)} = \mathbf{P} \cdot \underline{\Phi}$. The set of satisfying states if then given by $\mathsf{Sat}(\mathsf{P}_{\bowtie\lambda}[\mathrm{X}\,\Phi]) = \{s \in \mathcal{S} \mid \underline{\mathsf{Prob}(\Psi)}(s) \bowtie \lambda\}$.

Example 8. *Consider again the DTMC given in Figure 2.1. To check the property* $\mathsf{P}_{\geqslant 0.9}[\mathrm{X}(4 \vee 5))]$ *that states* 4 *or* 5 *are reachable in the next step with probability greater than, or equal to,* 0.9*, the vector of probabilities is calculated as*

$$\underline{\mathsf{Prob}(\mathrm{X}(4 \vee 5))} = \begin{bmatrix} 0 & 0.4 & 0.6 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.9 \\ 0 & 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0.8 & 0.2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0.8 \\ 0.2 \\ 1 \end{bmatrix}.$$

Comparing to the bound $\geqslant 0.9$ *gives* $\mathsf{Sat}(\mathsf{P}_{\geqslant 0.9}[\mathrm{X}(4 \vee 5))) = \{5\}$.

Bounded until. For formulas of the form $\mathsf{P}_{\bowtie\lambda}[\Phi\,\mathrm{U}^{\leqslant k}\,\Phi']$ where $k \neq \infty$ the probabilities are calculated as follows. First let the sets $\mathcal{S}^1 = \mathsf{Sat}(\Phi')$ and $\mathcal{S}^0 = \mathcal{S} \setminus (\mathsf{Sat}(\Phi) \cup \mathsf{Sat}(\Phi'))$ denote the sets of all states where $\Phi\,\mathrm{U}^{\leqslant k}\,\Phi'$ is satisfied with probability 1 and 0, respectively.

When the path property is a reachability property of the form $\mathrm{F}^{\leqslant k}\,\Phi \equiv \mathit{true}\,\mathrm{U}^{\leqslant k}\,\Phi$ the set $\mathcal{S}^0$ is simply the empty set. The probabilities for each state s are then given by the recursive equations

$$\mathsf{Prob}(s, \Phi\,\mathrm{U}^{\leqslant k}\,\Phi') = \begin{cases} \sum_{s' \in \mathcal{S}} \mathbf{P}(s,s')\mathsf{Prob}(s', \Phi\,\mathrm{U}^{\leqslant k-1}\,\Phi') & \text{if } s \in \mathcal{S} \setminus (\mathcal{S}^1 \cup \mathcal{S}^0) \text{ and } k > 0 \\ 1 & \text{if } s \in \mathcal{S}^1 \\ 0 & \text{otherwise.} \end{cases}$$

These can be computed by first constructing a new probability matrix $\mathbf{P}'$ that is equivalent to $\mathbf{P}$ excepting that all states in $\mathcal{S}^1$ and $\mathcal{S}^0$ are absorbing, and then computing the transient probabilities of being in a state satisfying Φ' after k steps. Intuitively, states in $\mathcal{S}^1$ are made absorbing because any paths starting from states in that set trivially satisfy the path formula $\Phi\,\mathrm{U}^{\leqslant k}\,\Phi'$, and conversely, states in $\mathcal{S}^0$ are made absorbing because any paths starting from a state in that set fail to satisfy the property. The probability matrix $\mathbf{P}'$ is constructed by setting $\mathbf{P}'(s,s) = 1$ for all $s \in (\mathcal{S}^1 \cup \mathcal{S}^0)$, $\mathbf{P}'(s,s') = 0$ for all $s \in (\mathcal{S}^1 \cup \mathcal{S}^0)$ and $s' \in \mathcal{S} \setminus (s^1 \cup s^0)$, and $\mathbf{P}'(s,s') = \mathbf{P}(s,s')$ for all other s and s'. The transient probabilities of being in a state that satisfies Φ' after at most k steps can then be calculated as the matrix-vector product $\underline{\mathsf{Prob}(\Phi\,\mathrm{U}^{\leqslant k}\,\Phi')} = \mathbf{P}'^{k} \cdot \underline{\Phi'}$. The satisfying set is then given by $\mathsf{Sat}(\mathsf{P}_{\bowtie\lambda}[\Phi\,\mathrm{U}^{\leqslant k}\,\Phi']) = \{s \in \mathcal{S} \mid \underline{\mathsf{Prob}(\Phi\,\mathrm{U}^{\leqslant k}\,\Phi')}(s) \bowtie \lambda\}$.

Example 9. *Again consider the DTMC in Figure 2.1, and the property* $\mathsf{P}_{\geqslant 0.4}[\mathit{true}\,\mathrm{U}^{\leqslant 2}\,4] \equiv \mathsf{P}_{\geqslant 0.4}[\mathrm{F}^{\leqslant 2}\,4]$. *The sets* $\mathcal{S}^1$ *and* $\mathcal{S}^0$ *are calculated as* $\mathcal{S}^1 = \{4\}$ *and* $\mathcal{S}^0 = \emptyset$, *and the probability matrix* $\mathbf{P}'$ *is then constructed by setting state* 4 *to be absorbing. The transient probabilities*

of being in a state satisfying 4 *after at most two steps are then given by*

$$\underline{\mathsf{Prob}(\mathit{true}\,\mathrm{U}^{\leqslant 2}\,4)} = \begin{bmatrix} 0 & 0.4 & 0.6 & 0 & 0 \\ 0 & 0 & 0.1 & 0 & 0.9 \\ 0 & 0 & 0.2 & 0.8 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^2 \begin{bmatrix} 0 \\ 0 \\ 0 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.48 \\ 0.08 \\ 0.96 \\ 1 \\ 0 \end{bmatrix},$$

and the satisfying set is $\mathsf{Sat}(\mathsf{P}_{\geqslant 0.4}[\mathit{true}\,\mathrm{U}^{\leqslant 2}\,4]) = \{1, 3, 4\}$.

Unbounded until. For formulas of the form $\mathsf{P}_{\bowtie\lambda}[\Phi\,\mathrm{U}\,\Phi']$, those where $k = \infty$, the probabilities are calculated as follows. Firstly the set $\mathcal{S}' \subseteq \mathcal{S}$ of all states from which some state satisfying Φ' is reachable with positive probability passing only through states that satisfy Φ is computed. The set of states $\mathcal{S}^0$, where $\Phi\,\mathrm{U}\,\Phi'$ is satisfied with probability 0, is then given by $\mathcal{S}^0 = \mathcal{S} \setminus \mathcal{S}'$. Next the the set $\mathcal{S}'' \subseteq \mathcal{S}$ of all states from which some state in $\mathcal{S}^0$ is reachable passing only through states that satisfy Φ is computed. The set $\mathcal{S}^1$ of states where $\Phi\,\mathrm{U}\,\Phi'$ is satisfied with probability 1 is then given by $\mathcal{S}^1 = \mathcal{S} \setminus \mathcal{S}''$. The probabilities for all states are then obtained as the solution to the linear equation system

$$\mathsf{Prob}(s, \Phi\,\mathrm{U}\,\Phi') = \begin{cases} 1 & \text{if } s \in \mathcal{S}^1 \\ 0 & \text{if } s \in \mathcal{S}^0 \\ \displaystyle\sum_{s' \in \mathcal{S}} \mathbf{P}(s, s')\mathsf{Prob}(s', \Phi\,\mathrm{U}\,\Phi') & \text{otherwise.} \end{cases}$$

Example 10. *Considering again the DTMC given in Figure 2.1, and the property* $\mathsf{P}_{\geqslant 0.4}[\mathrm{F}\,4]$. *The sets* $\mathcal{S}^0$ *and* $\mathcal{S}^1$ *are calculated as* $\mathcal{S}^0 = \{5\}$ *and* $\mathcal{S}^1 = \{3, 4\}$. *The resulting linear equa-*

tion system is

$$\mathsf{Prob}(1, \mathrm{F}\, 4) = 0.4 \cdot \mathsf{Prob}(2, \mathrm{F}\, 4) + 0.6 \cdot \mathsf{Prob}(3, \mathrm{F}\, 4)$$

$$\mathsf{Prob}(2, \mathrm{F}\, 4) = 0.1 \cdot \mathsf{Prob}(3, \mathrm{F}\, 4) + 0.9 \cdot \mathsf{Prob}(5, \mathrm{F}\, 4)$$

$$\mathsf{Prob}(3, \mathrm{F}\, 4) = 1$$

$$\mathsf{Prob}(4, \mathrm{F}\, 4) = 1$$

$$\mathsf{Prob}(5, \mathrm{F}\, 4) = 0$$

and through substitution the two unknowns are $\mathsf{Prob}(2, \mathrm{F}\, 4) = 0.1$ *and* $\mathsf{Prob}(1, \mathrm{F}\, 4) = 0.64$.

Rewards. For reward formulas of the form $\mathsf{R}_{\bowtie r}[\mathrm{F}\, \Phi]$, satisfiability of a state is determined by calculating the expected reward $\mathrm{E}[X_{\mathsf{Sat}(\mathrm{F}\, \Phi)}]$ with respect to some reward function $\mathbf{R}$, which will be denoted by $\mathsf{Exp}(s, \mathrm{F}\, \Phi)$, and comparing the resulting value to the bound $\bowtie r$. The sets $\mathcal{S}^1$ and $\mathcal{S}^0$ are computed as for unbounded until properties, and the set $S^{<1}$ of all states for which the probability of reaching a state satisfying Φ is less than 1 is given by $\mathcal{S}^{<1} = \mathcal{S} \setminus \mathcal{S}^1$. The expected rewards for all states are then obtained as the solution to the linear equation system

$$\mathsf{Exp}(s, \mathrm{F}\, \Phi) = \begin{cases} 0 & \text{if } s \in \mathsf{Sat}(\Psi) \\ \infty & \text{if } s \in \mathcal{S}^{<1} \\ \mathbf{R}(s) + \sum_{s' \in \mathcal{S}} \mathbf{P}(s, s')(\mathbf{R}(s, s') + \mathsf{Exp}(s', \mathrm{F}\, \Phi)) & \text{otherwise.} \end{cases}$$

Example 11. *Considering again the DTMC annotated with rewards of Figure 2.3b, and the property* $\mathsf{R}_{\geqslant 0.4}[\mathrm{F}\, 4]$. *The sets* $\mathcal{S}^0$ *and* $\mathcal{S}^1$ *are the same as in example 10 and* $\mathcal{S}^0 = \{5\}$

and $\mathcal{S}^1 = \{3, 4\}$*, and* $\mathcal{S}^{<1} = \{1, 2, 5\}$*. The resulting linear equation system is*

$$\begin{aligned}
\mathsf{Exp}(1, \mathrm{F}\, 4) &= \infty \\
\mathsf{Exp}(2, \mathrm{F}\, 4) &= \infty \\
\mathsf{Exp}(3, \mathrm{F}\, 4) &= 1 + 0.2 \cdot \mathsf{Exp}(3, \mathrm{F}\, 4) + 0.8 \cdot \mathsf{Exp}(4, \mathrm{F}\, 4) \\
\mathsf{Exp}(4, \mathrm{F}\, 4) &= 0 \\
\mathsf{Exp}(5, \mathrm{F}\, 4) &= \infty
\end{aligned}$$

and the single unknown is $\mathsf{Exp}(3, \mathrm{F}\, 4) = 1.25$*.*

Exact vs. numerical methods. While linear equation systems can be solved using exact solution methods, this is often too expensive for the analysis of larger models; Gaussian elimination is $O(n^3)$, for example. Therefore, most probabilistic model checking software uses value iteration methods such as Jacobi or Gauss-Seidel to obtain results where some user-specified precision is achieved.

2.5 Parametric model checking

Model checking PCTL reachability properties over a DTMC yields concrete values for the probability of taking some set of paths that satisfy the specification. When checking these properties over a PMC is it useful to obtain a closed-form solution (a rational function over the parameters of the model) for the *parametric reachability* probability – the probability of reaching some set of target states. This is useful when the exact probabilities are not known a priori, or when a property needs to be checked for a set of well-defined evaluations for the parameters of the model. For the former, when the concrete probability values are

known they can be simply substituted into the closed-form solution, which can then be efficiently evaluated; for the latter, model checking only needs to be applied once, and then again the closed form solution can be evaluated for each set of concrete values for the parameters.

An algorithm to obtain the rational function corresponding to the parametric reachability of some desirable states of a model was first introduced by Daws [41]. He interpreted the Markov chain under consideration as a finite automaton, in which transitions are labelled with symbols that correspond to rational numbers or variables. He then used state elimination [84] to obtain a regular expression for the language of the automaton. Evaluating these expressions into rational functions gave the probability of reaching target states. A simplification and refinement of the algorithm was introduced [70] and implemented [69] by Hahn et al., where rational functions were used to represent transition probabilities, instead of regular expressions.

The rest of this section is structured as follows. Firstly, properties for the parametric reachability of a set of target states are defined. Next the state elimination method of Hahn [70] for the calculation of these properties over PMCs is introduced. Finally, the extension of the algorithm to models annotated with rewards is described.

Parametric reachability. Given a PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$, let Pr be the parametric probability measure over Paths, the infinite paths of $\mathcal{D}_{\mathcal{X}}$, and $\Omega \subseteq \mathcal{S}$ be some set of target states. Define

$$X(\mathcal{D}_{\mathcal{X}})_i \colon \mathsf{Paths} \to \mathcal{S},$$

where $X(\mathcal{D}_{\mathcal{X}})_i(s_0 s_1 \ldots) = s_i$ to be the random variable taking values over the set of states that may be occupied at step $i \geqslant 0$ along some infinite path of the model, starting from

the initial state. The *parametric probabilistic reachability* is determined by computing the function

$$\mathsf{Reach}(\mathcal{D}_{\mathcal{X}}, \Omega) = \mathsf{Pr}\{\exists i \in \mathbb{N}.X(\mathcal{D}_{\mathcal{X}})_i \in \Omega\}.$$

Given a PMC augmented with a reward function $\mathbf{R}$, and some set of $\Omega \subseteq \mathcal{S}$, the property of interest is the *parametric expected accumulated reachability reward* [100]. As a simplification step, it is required that transition rewards are translated into state rewards. This is always possible when considering expected accumulated rewards for models without non-determinism [125]. The parametric expected accumulated reachability reward is obtained by computing the function

$$\mathsf{Acc}(\mathcal{D}_{\mathcal{X}}, \Omega) = \mathrm{E}\left[\sum_{i=0}^{\inf\{j|X(\mathcal{D}_{\mathcal{X}})_j \in \Omega\}} \mathbf{R}(X(\mathcal{D}_{\mathcal{X}})_i).\right].$$

State elimination. The algorithm of Hahn [70] shown in Figure 2 calculates the parametric reachability probability $\mathsf{Reach}(\mathcal{D}_{\mathcal{X}}, \Omega)$ for a PMC $\mathcal{D}_{\mathcal{X}}$ given some set of target states Ω by stepwise eliminating states from the model. At each step a state is removed from the model, and the new structure has all successors of this state as (potentially new) successors of the predecessors of this state. The probabilities, and if applicable, rewards on the edges are adjusted.

The algorithm proceeds as follows, where input is a PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{X})$ and a set of target states $\Omega \subset \mathcal{S}$. Initially, preprocessing is applied (Algorithm 1), and without loss of generality all states from which the set of target states is unreachable are removed. The function $\mathsf{Eliminate}(\mathcal{D}_{\mathcal{X}}, s)$ sets the probability of all incoming and outgoing transitions from s to be 0, eliminating these transitions and s from the underlying graph of $\mathcal{D}_{\mathcal{X}}$. Again without loss of generality, all outgoing transitions from states in Ω are then removed,

and a new state s_t is introduced such that there is a transition with probability 1 from all states in s_t to that state. A state s_e is chosen for elimination from Elim, the set of states to be eliminated, and Algorithm 3 is called to eliminate s_e from $\mathcal{D}_\mathcal{X}$ as follows. For every predecessor/successor pair $(s_1, s_2) \in \mathsf{pre}(s_e) \times \mathsf{post}(s_e)$ for s_e the existing probability $\mathbf{P}(s_1, s_2)$ is incremented by the probability of reaching s_2 from s_1 via s_e. The fractional term results from evaluating the geometric sum

$$\sum_{i=0}^{\infty} \mathbf{P}(s_1, s_e)\mathbf{P}(s_e, s_e)^i\mathbf{P}(s_e, s_2) = \frac{\mathbf{P}(s_1, s_e)\mathbf{P}(s_e, s_2)}{1 - \mathbf{P}(s_e, s_e)}$$

that is the sum of the probabilities of taking paths where s_e is visited $1 \dots \infty$ times. The state is then eliminated from the underlying graph of $\mathcal{D}_\mathcal{X}$. This procedure is repeated until only s_0 and s_t remain, and $\mathsf{Reach}(\mathcal{D}_\mathcal{X}, \Omega)$, the probability of reaching Ω from s_0, is then given by $\mathbf{P}(s_0, s_t)$.

The order in which states are eliminated does not affect the correctness of the approach [70]. Different orders yield equivalent, but possibly different, rational functions for the parametric probabilistic reachability. In Chapter 6 it is shown how guiding the order in which state elimination is applied can improve the efficiency of analysing families of parameterwise different PMCs, and in Chapter 6 heuristics for selecting an order for elimination are discussed.

Example 12. *Figure 2.4(a) shows a PMC $\mathcal{D}_\mathcal{X}$ with states $\mathcal{S} = \{1, 2, 3, 4, 5\}$, initial state 1, and parameters $\mathcal{X} = \{p, q, r\}$. State elimination is applied to the PMC to obtain $\mathsf{Reach}(\mathcal{D}_\mathcal{X}, \{4\})$, the parametric probabilistic reachability of the target state 4, as follows. Firstly, state 5 and its incident transitions are removed from the model, since the target state 4 is unreachable from that state (Figure 2.4(b)). Next, all outgoing transitions from the target state 4 are set to 0, a new state s_t is introduced to the model, and the probability*

Algorithm 1 Parametric reachability probability for PMCs.

procedure ReachabilityProbability($\mathcal{D}_\mathcal{X}, \Omega$)
  requires: A PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ and a set of target states $\Omega \subseteq \mathcal{S}$.
  $\mathcal{S}, \mathbf{P} \leftarrow \mathsf{Preprocess}(\mathcal{D}_\mathcal{X}, \Omega)$
  $\mathsf{Elim} \leftarrow \mathcal{S} \setminus \{s_0, s_t\}$
  while $\mathsf{Elim} \neq \emptyset$ **do**
    $s_e \leftarrow \mathsf{choose}(\mathsf{Elim})$
    $\mathbf{P} \leftarrow \mathsf{StateElimination}(\mathcal{D}_\mathcal{X}, s_e)$
    $\mathsf{Elim} \leftarrow \mathsf{Elim} \setminus \{s_e\}$
  end while
  return $\mathbf{P}(s_0, s_t)$
end procedure

Algorithm 2 PMC preprocessing for state elimination.

procedure Preprocess($\mathcal{D}_\mathcal{X}, \Omega$)
  requires: A PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ and a set of target states $\Omega \subseteq \mathcal{S}$.
  for all $s \in (\mathcal{S} \setminus \{s_0\})$ **do**
    if not $\mathsf{reach}_{\mathcal{D}_\mathcal{X}}(s, \Omega)$ **then** $\mathsf{Eliminate}(\mathcal{D}_\mathcal{X}, s)$
  end for
  $\mathcal{S} \leftarrow \mathcal{S} \cup \{s_t\}$
  for all $s \in \Omega$ **do**
    for all $s' \in \mathsf{post}_{\mathcal{D}_\mathcal{X}}(s)$ **do** $\mathbf{P}(s, s') \leftarrow 0$
    $\mathbf{P}(s, s_t) \leftarrow 1$
  end for
  return $\mathcal{S}, \mathbf{P}$
end procedure

of transitioning from state 4 *to* s_t *is set to* 1 *(Figure 2.4(c)). Figures 2.4(d)-2.4(f) then show the elimination of states* 3, 4, *and* 2, *respectively. Once state elimination is complete the probability labelling the transition from the initial state* s_0 *to the introduced state* s_t *is the parametric probabilistic reachability of state* 4,

$$\mathsf{Reach}(\mathcal{D}_\mathcal{X}, \{4\}) = \frac{(1-p)(1-r) + (1-q)(1-r)}{1-r}.$$

Eliminating the states in a different order results in a different, though equivalent, expres-

Algorithm 3 Elimination of a single state of a PMC.

procedure StateElimination($\mathcal{D}_{\mathcal{X}}, s_e$)
  requires: A PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ and a state to eliminate $s_e \in \mathcal{S}$.
  for all $(s_1, s_2) \in \mathsf{pre}(s_e) \times \mathsf{post}(s_e)$ **do**
    $\mathbf{P}(s_1, s_2) \leftarrow \mathbf{P}(s_1, s_2) + \frac{\mathbf{P}(s_1, s_e)\mathbf{P}(s_e, s_2)}{1 - \mathbf{P}(s_e, s_e)}$
  end for
  $\mathsf{Eliminate}(\mathcal{D}_{\mathcal{X}}, s_e)$
  return P
end procedure

sion.

State elimination with rewards. Given a PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$, a set of target states $\Omega \subset S$, and a reward function $\mathbf{R}$ for $\mathcal{D}_{\mathcal{X}}$, Algorithm 4 computes the parametric expected accumulated reachability reward. The algorithm is an extension of Algorithm 2 where for each predecessor and successor pair of s_e, a state to be eliminated from $\mathcal{D}_{\mathcal{X}}$, the reward function is also updated as follows:

$$\mathbf{R}(s_1) \leftarrow \mathbf{R}(s_1) + \mathbf{P}(s_1, s_e)\frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)}\mathbf{R}(s_e).$$

The updated value for $\mathbf{R}(s_1)$ reflects the reward that would be accumulated if a transition would be taken from s_1 to s_e, where the expected number of self-loops would be

$$\frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)}.$$

Upon termination, $\mathsf{Acc}(\mathcal{D}_{\mathcal{X}}, \Omega)$, the parametric expected accumulated reachability reward, is then given by $\mathbf{R}(s_0)$.

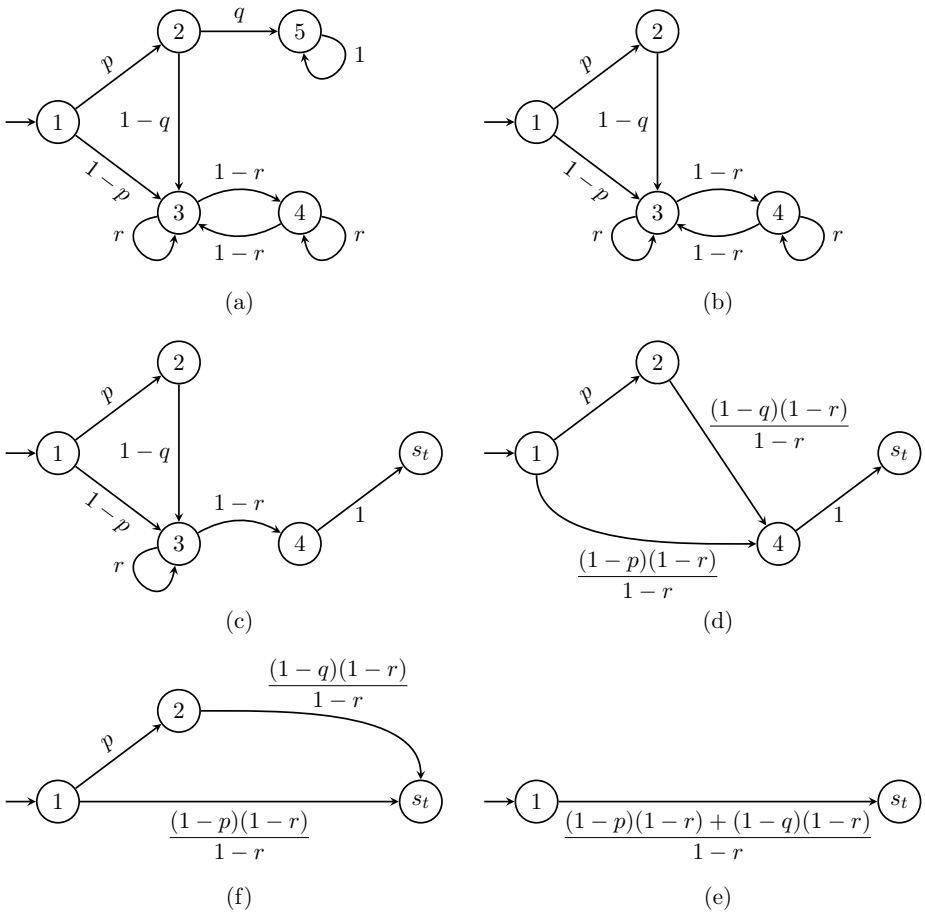


Figure 2.4: State elimination for a parametric Markov chain.

2.6 Statistical model checking

Model checking approaches are useful to guarantee the correctness of a system with respect to some specification, and many heuristics have been developed to reduce their memory footprint, for example symbolic approaches using binary decision diagrams or partial order reduction. However, it is still often the case that the state-space explosion problem impedes the analysis of larger systems. To overcome this, *statistical model checking* [150, 134] was

Algorithm 4 Parametric expected accumulated reachability reward for PMCs.

1: **procedure** ExpectedAccumulatedReachabilityReward($\mathcal{D}_\mathcal{X}, \Omega$, $\mathbf{R}$)
2: **requires:** A PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$,a set of target states $\Omega \subseteq \mathcal{S}$, and a reward function $\mathbf{R}\colon \mathcal{S} \to \mathcal{Q}_\mathcal{X}$.
3: $\mathcal{S}, \mathbf{P} \leftarrow \mathsf{Preprocess}(\mathcal{D}_\mathcal{X}, \Omega)$
4: $\mathsf{Elim} \leftarrow \mathcal{S} \setminus \{s_0, s_t\}$
5: **while** $\mathsf{Elim} \neq \emptyset$ **do**
6: $s_e \leftarrow \mathsf{choose}(\mathsf{Elim})$
7: $\mathbf{P}, \mathbf{R} \leftarrow \mathsf{StateEliminationReward}(\mathcal{D}_\mathcal{X}, s_e)$
8: $\mathsf{Elim} \leftarrow \mathsf{Elim} \setminus \{s_e\}$
9: **end while**
10: **return** $\mathbf{R}(s_0)$
11: **end procedure**

Algorithm 5 Elimination of a single state of a PMC incorporating rewards.

1: **procedure** StateEliminationReward($\mathcal{D}_\mathcal{X}, s_e$)
2: **requires:** A PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$,a state to eliminate $s_e \in \mathcal{S}$ and a reward function $\mathbf{R}\colon \mathcal{S} \to \mathcal{Q}_\mathcal{X}$.
3: **for all** $(s_1, s_2) \in \mathsf{pre}(s_e) \times \mathsf{post}(s_e)$ **do**
4: $\mathbf{P}(s_1, s_2) \leftarrow \mathbf{P}(s_1, s_2) + \frac{\mathbf{P}(s_1, s_e)\mathbf{P}(s_e, s_2)}{1 - \mathbf{P}(s_e, s_e)}$
5: $\mathbf{R}(s_1) \leftarrow \mathbf{R}(s_1) + \mathbf{P}(s_1, s_e)\frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)}\mathbf{R}(s_e)$
6: **end for**
7: $\mathsf{Eliminate}(\mathcal{D}_\mathcal{X}, s_e)$
8: **return** $\mathbf{P}, \mathbf{R}$
9: **end procedure**

proposed as an alternative to exhaustive state-space exploration. Here hypothesis testing and Monte Carlo simulations are used to determine if a finite number of executions of a model can demonstrate to some degree of certainty that the model satisfies or violates some given property [108]. The technique is a compromise between classical model checking and testing, and is less memory and time intensive than exhaustive techniques. Therefore, it can be used to obtain approximate results for large models where it would be otherwise infeasible to obtain results within a reasonable time or resource bound.

2.7 Tools for probabilistic model checking

Probabilistic model checkers are tools for the formal modelling and analysis of stochastic models. The tools take as input a model encoded in some appropriate formal language, and then automate the process of checking desirable properties expressed in PCTL, or other suitable logics, against the formal model. The results obtained throughout this thesis were obtained using the publicly available probabilistic model checkers PRISM [101], ePMC [71], and Storm [45].

In Chapters 3, 4, and 5 the input language developed for the PRISM model checker is used to encode the models that are developed there. A description of this language is given in the following section. The prototypical model checking tool developed for the work presented in Chapter 6 takes as input a low-level representation of model as a set of states and a set of transitions, that are defined in a file format widely accepted by probabilistic model checkers. A script is provided to automate the generation of low-level representations of the models described in Sections 4 and 5, should they be required for analysis with other tools

2.7.1 PRISM guarded command language

The PRISM language is a state-based language, based on the Reactive Modules formalism of Alur and Henzinger [4], and was used to encode the models that are presented in Chapters 3, 4, and 5. The language was originally developed for the PRISM model checker, but can also be parsed by the model checkers ePMC and Storm. Formally, the language can be defined as follows.

Definition 11. *A model $\mathcal{M}$ is a set of* modules *[120], where each module* M *is a tuple* $\mathsf{M} = (\mathcal{V}, I, \mathcal{C})$. $\mathcal{V} = \{v_1, \ldots, v_k\}$ *is a set of local variables over the domain consisting of finitely bound integers and Boolean values. With each local variable v there is an associated*

variable v' denoting the state of v in the next moment of time. The union of the sets of local variables for all the modules in $\mathcal{M}$ is denoted by $\mathcal{V}^{\mathcal{M}}$. I is a mapping from variables to their initial values. $\mathcal{C}$ is a set of commands *that define the behaviour of the module. A command $c \in \mathcal{C}$ is a pair $(g, \mathcal{T})$ where g is a predicate over $\mathcal{V}^{\mathcal{M}}$ and $\mathcal{T}$ is a set of possible transitions for the module. Each transition is a pair (p, A) where $p \in \mathbb{R}_{>0}$ is a constant defining the probability of that update occurring, and A is an assignment of values to each of the local variables in $\mathcal{V}$. Each assignment is of the form $\bigwedge_{i=1}^{k}(v_i' = ex_i)$, where each ex_i is an expression in terms of V and the domain of variables.*

The semantics of a model $\mathcal{M} = \{\mathsf{M}_1, \ldots, \mathsf{M}_n\}$ can be defined in terms of a DTMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, L)$. For DTMCs it is required that $p \in (0, 1]$ for every transition (p, A), and $\sum_{(p,A)\in\mathcal{T}} p = 1$ for every command $(g, \mathcal{T})$. The local state space $\mathcal{S}_i$ of a module $\mathsf{M}_i = (\mathcal{V}_i, I_i, \mathcal{C}_i)$ is the set of all valuations of $\mathcal{V}_i$. The global state space $\mathcal{S} = \mathcal{S}_1 \times \ldots \times \mathcal{S}_n$ of $\mathcal{M}$ is the product of all local state spaces. Labels for states are predicates over the set of variables $\mathcal{V}$. For details of the calculation of $\mathbf{P}$ the reader is referred to [120]. The semantics can also be given in terms of a PMC, by also allowing transition probabilities to be labelled with rational functions over a finite set of parameters (variables over $[0, 1]$).

Example 13. *Figure 2.5 shows the Knuth and Yao algorithm [94] for the simulation of a six-sided dice, by repeatedly tossing a coin. Here the model has been extended to simulate the tossing of a biased coin, where with probability x the coin toss results in heads, and with probability $1 - x$ the coin toss results in tails. Starting from step 0 at each step a coin is tossed and one of two possible choices are taken with probability x or $1 - x$. The algorithm terminates when one of the values ⚀…⚅ is reached. The algorithm can be modelled as a PMC with parameter x. Listing 2.1 shows an example of PRISM code that encodes such a PMC.*

The first line indicates that the model is a discrete-time Markov chain. Lines $3 - 21$

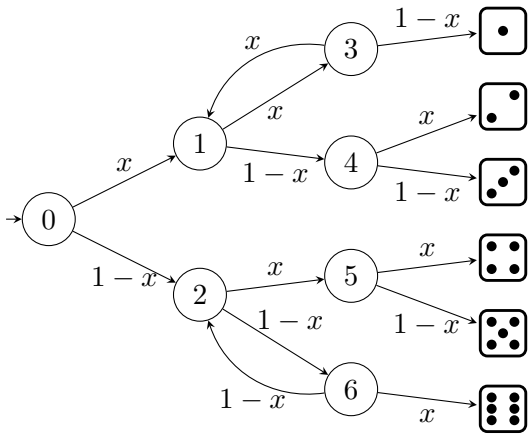


Figure 2.5: PMC model of a dice simulated using a biased coin.

describe a single PRISM module. Line 5 *defines an open parameter of the model,* `x`*, and so this model is a parametric Markov chain. Lines* 8 *and* 9 *defines integer variables* $\mathtt{s} \in [0 \dots 7]$ *and* $\mathtt{d} \in [0 \dots 6]$ *with initial values of* 0 *that are used to represent the state of the system. Here a value of* $k \in [0 \dots 6]$ *for* `s` *indicates that the system is in a state equivalent to state* k *in Figure 2.5, while a value of* 7 *for* `s` *indicates that the system is in a state that corresponds to one of the states* ⚀ … ⚅*. The value of* `d` *has a value of* 0 *when* `s` *has a value in* $[1 \dots 6]$*, and takes the value of the dice when* $\mathtt{s} = 7$ *and the system is in one of the states* ⚀ … ⚅*.*

Lines $12 - 19$ *are the commands that define the behaviour of the module, consisting of a predicate over the module variables and a set of possible transitions. For example, line* 12 *states that whenever the local variable* $\mathtt{s} = 0$*, in the next moment in time the system will move to a state where* $\mathtt{s} = 1$ *with probability* x*, or to a state where* $\mathtt{s} = 2$ *with probability* $1 - x$*. The value of the local variable* `s` *in the next state is denoted by* $\mathtt{s}'$*.*

```
dtmc

module die

    const double x;

    // local state
    s : [0..7] init 0;
    // value of the die
    d : [0..6] init 0;

    [] s=0 -> x : (s' = 1) + (1 - x) : (s' = 2);
    [] s=1 -> x : (s' = 3) + (1 - x) : (s' = 4);
    [] s=2 -> x : (s' = 5) + (1 - x) : (s' = 6);
    [] s=3 -> x : (s' = 1) + (1 - x) : (s' = 7) & (d' = 1);
    [] s=4 -> x : (s' = 7) & (d' = 2) + (1 - x) : (s' = 7) & (d' = 3);
    [] s=5 -> x : (s' = 7) & (d' = 4) + (1 - x) : (s' = 7) & (d' = 5);
    [] s=6 -> x : (s' = 2) + (1 - x) : (s' = 7) & (d' = 6);
    [] s=7 -> (s' = 7);

endmodule
```

Listing 2.1: PRISM code for an algorithm to simulate a biased dice using a coin.

Chapter 3

Probabilistic Model Checking of Ant-Based Positionless Swarming

In this chapter, a formal analysis of an algorithm inspired by the foraging behaviour of ants is conducted, where a swarm of flying *micro air vehicles* (MAVs) is deployed to locate a target at some unknown location. The algorithm was originally presented by Hauert et al. in [77], and coordinates the activity of a swarm of micro air vehicles (MAVs) attempting to locate a target in some unknown location. Once the target has been located, the swarm tries to establish a robust communication network between that target and a base station from where the swarm was launched.

Designing control mechanisms for swarms is a challenging problem. Individual behaviours must be formulated at the *microscopic* level and should result in the emergence of complex desired group behaviours at the *macroscopic* level. There are many examples found in nature of decentralised systems that solve complex problems [17]. A common approach in swarm robotics has been to develop control algorithms based on abstractions of these natural systems. In particular, much work has been conducted to develop control

algorithms based on the behaviours of social insects, such as foraging for food [29, 110], cooperative nest building [141], and efficient distribution of labour [18].

The control algorithm investigated here is inspired by the stigmergic foraging behaviour of army ants that lay and maintain pheromone paths from their nest to sources of food [19]. The ants initially make random walks from the nest, depositing light pheromone trails as they explore. If an ant finds a source of food it returns home, reinforcing the trail with larger deposits of pheromone that are often proportional to the quality of the food source. Deposited pheromone influences the behaviour of other ants, who are more inclined to follow, and also reinforce, those trails with larger deposits, creating a positive feedback loop. When a food source is depleted less ants will reinforce the trail, and a negative feedback loop resulting from pheromonal dissipation ensures that trails to depleted sources soon disappear. The combination of positive and negative feedback loops often lead to such trails converging to the shortest path between the nest and the source [52].

A model of this behaviour was originally developed in [47], where the results of running Monte Carlo simulations of ants moving through a discrete network of points were analysed. These findings were later discussed in detail in [17]. By taking inspiration from this behaviour, the algorithm presented in [77] aims to create and maintain shortest communication pathways between a base station and some target in an unknown location. Since stigmergic communication via the environment itself is often undesirable or not practical, the swarm itself forms a platform for the deposition of virtual pheromone.

This chapter describes the application of probabilistic temporal verification to the scenario presented in [77]. Parametrised probabilistic models are generated for its verification, encoded using the guarded command language described in Section 2.7.1. The models are validated by first checking reachability properties of interest in the models using statistical analysis of a subset of the possible runs of the system, and then comparing these results to

those obtained from simulations in [77]. Probabilistic reachability and reachability reward properties of interest are then exhaustively checked in the model using the probabilistic model checker PRISM. It is demonstrated how results that would facilitate the logistics of deploying a swarm of MAVs can be obtained a priori using these models.s

This scenario was selected as a case study for the following reasons. Firstly, the algorithm itself is interesting, and is inherently stochastic, making it an ideal candidate for formal probabilistic analysis. Secondly, the swarm of interest was composed of physically, and behaviourally, identical members, and the property of interest – locating the target – did not require any member of the swarm to be distinguished from the others. This homogeneity allows abstractions to be employed when creating formal models of such scenarios, and hence allows larger models to be analysed.. A general discussion of the use of abstractions was given in the introduction, and its application to this scenario is discussed in more detail in Section 3.3. Thirdly, the system had several clearly defined parameters that would determine the structure of any formal model developed for its analysis; examples include the size of the swarm, the maximum distance at which the target might be located, and a parameter defining the exploration behaviour of the swarm. Finally, the exploration algorithm considered a grid of discrete locations in the environment that motivated a natural abstraction for recording the locations of members of the swarm.

Related work is discussed in Section 3.1. Section 3.2 introduces the ant-based swarming scenario described in [77]. In Section 3.3 the generation of the parametrised input models is detailed, and both the abstractions used, and assumptions made, when designing the discrete formal model are discussed. The results of checking probabilistic reachability and reachability reward based properties in the models is given in Section 3.4. Concluding remarks and suggestions for further work are given in Section 3.5.

3.1 Related work

Laibinis et al. [103] formalised the behaviour of a colony of foraging ants using Event-B [14], a formalism appropriate for the correct-by-construction approach to the design of distributed systems. An abstract specification of the behaviour of the ants and their environment was transformed into a more detailed model via successive refinements, where at each refinement step it was verified that desirable behaviours were preserved. The reachability objective for the colony of ants – to gather all available food from the environment – was formalised as a termination problem, and a proof for termination was then obtained. Similarly to the models presented here, decisions made by ants were influenced by perceived amounts of pheromone deposited by other ants in the environment. In contrast, the behavioural choices of the ants are resolved non-deterministically, not stochastically. While the authors indicate that stochastic reasoning could be introduced to the model, this would only allow qualitative analysis with respect to the global objective [72], which precludes analyses to obtain quantitative results like those presented here.

Process algebras have been used to reason about the behaviour of interacting social insects, and were first applied by Tofts in [142]. They were used in [143] to study the allocation of tasks to individual ants based on the availability of work, and in [22] to investigate correlations between the factors determining the behaviour of individual ants. In [139], Sumpter et al. investigated the activity patterns of a colony of ants. Interactions at the microscopic level between individual ants were defined using process algebra, which allowed for computer simulation and Markov chain analysis, while dynamical systems approaches were used to analyse the emergent global properties at the macroscopic level. All these studies focused on the evolution of the global behaviour over time, and spatial aspects of the environment were not of interest, and hence not encoded in the models, unlike those presented here. Another study in [114] used process algebras and fluid flow analysis to re-

spectively analyse and encode the behaviour of foraging ants that make choices over which paths to take in order to reach some food in the environment. The choices of which path to take are stochastic, and determined by deposited levels of pheromone, though pheromonal decay over time is not encoded in the model. In contrast to the work presented here, their model only encodes a topological map of the environment, and the analysis is restricted to two possible paths from the nest to the food.

In [137] a multi-agent systems approach combining X-machines [53] and population P systems [12] was used to formalise the behaviour of a colony of ants, foraging for food in a grid-like environment. Simulation was used to provide statistical data for the validation of the model, however no verification results were presented, and the pheromone induced stochastic behaviour of the ants included in the simulations was not encoded in the formal model. Multi-agent based simulation methods were also employed by Herd et al. in [80], where statistical model checking was used to analyse a probabilistic model of nature-inspired foraging robots first introduced by Liu et al. in [110]. Similarly to parts of this work, the analysis focused on the energy consumed by the robots, but only considered the global behaviour of the system over time, and again spatial aspects of the environment were not encoded in the models. The model of Liu et al. was also analysed by Konur et al. in [96], using parametric Markov chains in a similar approach to that presented here. Behdenna et al. [9] used first-order linear temporal logic to formalise the model, and proofs of desirable properties were automated. The analysis was restricted to very small numbers of robots, and stochastic properties of the model could not be investigated as the probabilities labelling transitions of the original model were abstracted away.

3.2 The ant-based swarming scenario

The scenario to which probabilistic model checking techniques are applied is presented in [77]. Here, a simulated swarm of *positionless* fixed-wing micro aerial vehicles (MAVs) is deployed by a human operator in order to establish a robust emergency communication network between a *target*, situated at some unknown location (but in some known cardinal direction), and the base station from where the swarm is launched.

Each MAV is positionless in that it relies solely upon proprioceptive sensors and local neighbourhood communication to position itself [138]. MAVs deposit "pheromones" as they explore, similarly to foraging army ants. Pheromone levels influence the navigation of MAVs in the swarm. Modifying the environment by the deposition of chemicals or objects is often undesirable, and map-based virtual pheromone depositing is impossible without global positioning [77]. As a solution to this issue MAVs can enter a *node* state in which they remain stationary at a location. When in this state they record the levels of virtual pheromone deposited by other exploring MAVs. Once a MAV has moved within communication range of the target a communication link is established and messages between the target and the base node are relayed via the network of MAVs in the node state. The establishment and maintenance of this communication network is studied in more detail in [77] – this work focuses on the exploration behaviour of the MAVs.

3.2.1 MAV behaviour

Figure 3.1 shows a Y-junction grid consisting of possible positions that MAVs will ideally adopt in their search for the target, and the paths that connect them, while Figure 3.2 shows the finite state machine describing the behaviour of an individual MAV. An MAV begins in landed state at the *base node*, denoted as $(0, 0)$ on the grid. MAVs are launched at regular intervals and are then in the exploring state. In the exploring state an MAV navigates

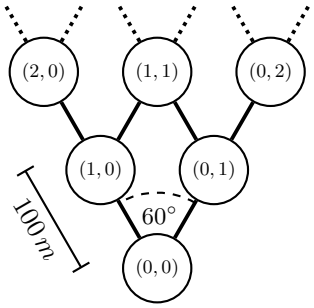


Figure 3.1: The Y-junction grid illustrating the ideal positions for MAVs. Each node is 100 m distant from each of its neighbours.

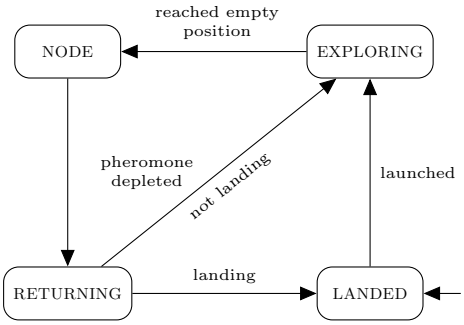


Figure 3.2: A finite state machine describing the behaviour of an MAV.

through the grid, travelling at a velocity of $10\,m/s$. When an MAV reaches a position in the grid where there is no other MAV it will change to node state and remain at that position, acting as a platform upon which other MAVs can "deposit" virtual pheromone. Upon changing to node state an MAV will initialise its pheromone levels to some given amount. An MAV in the node state at some position (i, j) is considered to be an *internal* node if there is an MAV in the node state at either of $(i+1, j)$ or $(i, j+1)$. Internal nodes supplement their levels of deposited pheromone at each time step by some amount. While in the node state each MAV broadcasts its pheromone level to other MAVs in the node state within its communication range of $100\,m$. When an MAV in the exploring state reaches a position in the grid where there is already an MAV in the node state, it continues moving outward and makes a probabilistic choice which branch to take, determined by the levels of pheromone deposited at the next positions on the left and right branches and a parameter that influences the choice of unexplored paths over explored paths, as transmitted by the MAV in the node state.

Pheromone levels dissipate gradually over time and when they are depleted, an MAV in the node state changes to the returning state (internal nodes are supplemented with

sufficient pheromone to guarantee that they cannot change to the returning state.) It then navigates back through the grid towards the base node similarly to an MAV in the exploring state but only moving along positions occupied by MAVs in the node state. Once it reaches the base node, if a signal to land is being broadcast by the base node then the MAV will land, otherwise it will change back to exploring state.

3.2.2 Path probabilities

In more detail, the choice between the left and right path from some position (i, j) is determined probabilistically according to the amount of deposited pheromone at $(i+1, j)$ and $(i, j+1)$ for MAVs in exploring state, or $(i-1, j)$ and $(i, j-1)$ for MAVs in the returning state. Given pheromone levels of $h_{(i+1,j)}$ and $h_{(i,j+1)}$, the probability of an MAV choosing the left or right path is calculated using (3.1) to (3.4), where $\mu \in (0, 1)$ is a real valued constant that determines the attractiveness of unexplored paths, and is set to 0.75 for the simulations in [77]. If there is no MAV in the node state at (i, j) then $h_{(i,j)} = 0$. Equations (3.3) and (3.4) are the calculations of the probabilities of taking the left or right path at (i, j) where the correction factor $c_L(i, j)$ defined in [77] is applied to the original probability calculation $p_L(i, j)$ given in [47]. This correction ensures that positions equidistant from the base node have an equal chance of being eventually reached, given equal amounts of pheromone on every path.

$$q_L(i,j) = \frac{\left(\mu + h_{(i+1,j)}\right)^2}{\left(\mu + h_{(i+1,j)}\right)^2 + \left(\mu + h_{(i,j+1)}\right)^2} \tag{3.1}$$

$$c_L(i,j) = \frac{i+1}{i+j+2} \tag{3.2}$$

$$p_L(i,j) = \frac{q_L(i,j)c_L(i,j)}{q_L(i,j)c_L(i,j) + (1 - q_L(i,j))(1 - c_L(i,j))} \tag{3.3}$$

$$p_R(i,j) = 1 - p_L(i,j) \tag{3.4}$$

3.3 Modelling the scenario

Next the design and automatic generation of parametrised models of the scenario, to which probabilistic analysis will be applied, are discussed. Formal models of the scenario were constructed using PRISM language, as detailed in Section 2.7.1.

3.3.1 Discretisation

Simulations were conducted in [77] using a time-step of $50\,ms$. Since an MAV travels at $10\,m/s$, and ideal positions for nodes are $100\,m$ distant from their neighbours, an MAV in the exploring or returning state takes $10\,s$ to move from one position to the next. For the formal models one transition in the model is considered to be equivalent to a time-step of $10\,s$. In the original scenario MAVs are launched from the base node by a human operator every $15 \pm 7.5\,s$, giving an interval in seconds of possible durations between launches of $[7.5, 22.5]$. By rounding the endpoints of the interval to the nearest multiple of 10, since one transition in our model is equivalent to $10\,s$, it can be determined that an MAV is launched from the base in the formal model once every one or two transitions. While additional transitions could be introduced to the model to encode this, by means of a distribution over the events where one, or two, MAVs are launched, a comparison of checking properties in

both the regular models, and those where these additional transitions were encoded, showed that there were only very minor differences between the results. Therefore, the model was further simplified so that exactly one MAV was launched per transition. This allowed much larger models to be analysed, since the state space of models with the additional encoding became too large.

When a MAV switches to node state at time t and position (i, j) it initialises the pheromone level $h_{(i,j)}(t)$ to h_{init}. The evolution of the pheromone levels at some position (i, j) is defined given by the equation

$$h_{(i,j)}(t+1) = \min(h_{(i,j)}(t) - \Delta_{h_{\text{dec}}} + n \cdot \Delta_{h_{\text{ant}}} + \Delta_{h_a}, h_{\max}) \tag{3.5}$$

where

- $h_{\max}$ is the maximum amount of pheromone that can be deposited at any node,
- $\Delta_{h_{\text{dec}}}$ is the rate at which deposited pheromone dissipates,
- $\Delta_{h_{\text{ant}}}$ is the rate at which pheromone is deposited on an MAV in the node state by an MAV in the exploring state or the returning state,
- n is the number of MAVs in the exploring state or the returning state at (i, j),
- $\Delta_{h_a} = \Delta_{h_{\text{init}}}$ if there is some MAV in the node state at $(i+1, j)$ or $(i, j+1)$, or 0 otherwise,
- $\Delta_{h_{\text{init}}}$ is the rate at which extra pheromone is deposited on internal nodes.

The simulations conducted in [77] used values $h_{\text{init}} = 0.7$ and $h_{\max} = 1$. The rates at which pheromone was deposited, or dissipated, given in terms of units per time-step with one time-step corresponding to $50\,ms$, were given as $\Delta_{h_{\text{ant}}} = 0.002$, $\Delta_{h_{\text{init}}} = 0.001$

and $\Delta_{h_{\text{dec}}} = 0.001$; in this model one transition is considered to be equivalent to $10\,s$ and therefore the values are multiplied by 200 to get the pheromone deposition/dissipation rates per transition in the model. To decrease the size of the models a range of discrete integer values is used to model pheromone levels. Given some $d \in \mathbb{N}$, the number of discrete values to be used to model pheromone levels, a pheromone value $h \in [0, h_{\max}]$ is mapped to the natural $\lfloor hd \rceil$, where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer. For the models analysed here a value of $d = 5$ is used; pheromone is deposited by MAVs in the exploring state or the returning state at a rate of $200 \cdot \Delta_{h_{\text{ant}}} \cdot 5 = 2$ per transition, internal nodes supplement their own pheromone levels at a rate of $200 \cdot \Delta_{h_{\text{init}}} \cdot 5 = 1$, and pheromone dissipates at a rate of $200 \cdot \Delta_{h_{\text{dec}}} \cdot 5 = 1$.

3.3.2 Abstractions and assumptions

Modelling each MAV individually would result in very large models. The model size is reduced by employing the following abstractions. First, since one transition in the formal model is equivalent to the duration of a flight between two adjacent nodes, and since all MAVs begin at the base node (position (i, j), see Figure 3.1), after each transition it can be assumed that the location of each MAV is always at some position (i, j), instead of in-between positions. Second, as demonstrated in [96] a *counting abstraction* can be used when modelling the behaviours of multiple identical processes. Since all MAVs are behaviourally identical, and their action decisions depend solely on their immediate state and percepts, a counter can be associated with every position (i, j) to record the number of MAVs in the returning state at that location. As detailed later in this section, the number of MAVs in the node or exploring state is always at most one, and therefore no such abstraction is required to model their behaviour.

For the simulations in [77], if the signal to land has not been given, then MAVs that

have returned to the base node instead resume exploration. Here the number of MAVs that may leave the base node at any moment in time is constrained to be at most one. This simplification allows us to greatly reduce the size of the model. Since each MAV is considered to land upon returning, and at most one MAV is launched each round, it can be concluded that at most one exploring MAV is at any given position at any time. This can be modelled using Boolean variables to record if an exploring MAV has moved from some position (i, j) to either of $(i + 1, j)$ or $(i, j + 1)$.

A strategy is given in [77] to automatically assign altitudes to individual MAVs, ensuring that MAVs in the exploring or the returning state maintain an altitude higher than MAVs in the node state. While this strategy did not prove to be successful in all cases, the chance of a collision occurring was sufficiently low (2.6% of 7500 MAVs collided over 500 trials) for us to assume that altitude differentiation always avoids collisions, and therefore choose to not encode this in our models.

3.3.3 Model generation

Given concrete values for the parameters $N \in \mathbb{N}$, the number of MAVs in the swarm, $U \in \mathbb{N}$, the maximum distance between the target and the base node (in hundreds of metres), and d, the number of discrete values used to record pheromone levels, the formal model $\mathcal{M}$ is encoded as a PMC with the single parameter μ, and can be defined using the state-based language discussed in Section 2.7.1, as a composition of modules:

$$\mathcal{M} = \{\mathsf{B}\} \cup \{\mathsf{E}_{i,j}, \mathsf{R}_{i,j} \mid i, j \in 0 \ldots U \text{ and } 0 < i + j < U\}$$
$$\cup \{\mathsf{F}_{i,j} \mid i, j \in 0 \ldots U \text{ and } i + j = U\},$$

where B is a module that models the movement of MAVs in exploring state from the base node, each $\mathsf{E}_{i,j}$ is a module that models the movement of MAVs in the exploring or node

states at (i,j), each $\mathsf{R}_{i,j}$ is a module that models the movement of MAVs in the returning state at (i,j), and each $\mathsf{F}_{i,j}$ is a module that models the movement of MAVs that have moved to, or beyond some point (i,j) that is at the maximum depth at which a target might be located. In PRISM it is possible to define many different parallel compositions of modules, but here all modules synchronise over all transitions – that is, a single transition is taken in each module at the same time. Each module in the model is now defined.

Base module. The base module is a tuple $\mathsf{B} = (\mathcal{V}_\mathsf{B}, I_\mathsf{B}, \mathcal{C}_\mathsf{B})$, where

$$\begin{aligned}\mathcal{V}_\mathsf{B} =& \{n^\mathsf{B}, \nwarrow_{0,0}, \nearrow_{0,0}\},\\ I_\mathsf{B} =& \{n^\mathsf{B} \mapsto N, \nwarrow_{0,0} \mapsto \mathit{false}, \nearrow_{0,0} \mapsto \mathit{false}\}.\end{aligned}$$

The finitely bound integer variable n^B records the number of MAVs at the base node. The boolean variables $\nwarrow_{0,0}$ and $\nearrow_{0,0}$ record the movement of MAVs in the exploring state, and are *true* if, and only if, in the last moment in time an MAV moved from the base node to $(1,0)$ or $(0,1)$, respectively. Should there be one or more MAVs at the base node then one will be launched and will move to $(1,0)$ with probability $p_L(0,0)$, or to $(0,1)$ with probability $p_R(0,0)$.

Exploring modules. For every $\mathsf{E}_{i,j} = \{\mathcal{V}_\mathsf{E}^{i,j}, I_\mathsf{E}^{i,j}, \mathcal{C}_\mathsf{E}^{i,j}\}$ we have local variables and initial values

$$\begin{aligned}\mathcal{V}_\mathsf{E}^{i,j} =& \{h_{i,j}, \bigcirc_{i,j}, \nwarrow_{i,j}, \nearrow_{i,j}\}\\ I_\mathsf{E}^{i,j} =& \{h_{i,j} \mapsto 0, n_{i,j} \mapsto 0, \nwarrow_{i,j} \mapsto \mathit{false}, \nearrow_{i,j} \mapsto \mathit{false}\},\end{aligned}$$

where $h_{i,j}$ is a finitely bound integer variable recording the levels of pheromone deposited at (i,j), $\bigcirc_{i,j}$ is a boolean variable that is *true* if there is an MAV in the node state at (i,j), and $\nwarrow_{i,j}$ and $\nearrow_{i,j}$ are Boolean variables which are are *true* if, and only if, an MAV moved from (i,j) respectively to $(i+1,j)$ or $(i,j+1)$ in the last moment in time. If no MAVs in the exploring state have moved to (i,j) then in the next moment in time no MAVs in the exploring state will be moving from (i,j); if $\bigcirc_{i,j} = \mathit{true}$ then pheromone updates will be applied, and if $h_{i,j} \leqslant 0$ then in the next moment in time the MAV will be in the returning state, otherwise the MAV remains in the node state. If an MAV in the exploring state has moved to (i,j) when $\bigcirc_{i,j} = \mathit{false}$ then in the next moment in time $\bigcirc_{i,j}$ will be *true* and $h_{i,j}$ will be initialised to $h_{\text{init}} \cdot d$; no MAV in the exploring state will be moving from (i,j) in the next moment in time. If an MAV in the exploring state has moved to (i,j) when $\bigcirc_{i,j} = \mathit{true}$, then in the next moment in time the exploring MAV will have moved to $(i+1,j)$ with probability $p_L(i,j)$, or to $(i,j+1)$ with probability $p_R(i,j)$, and remains in the exploring state; pheromone updates are applied and if $h_{i,j} \leqslant 0$ then in the next moment in time the MAV in the node state will be in the returning state, otherwise this MAV remains in the node state.

Returning modules. For every $\mathsf{R}_{i,j} = \{\mathcal{V}_{\mathsf{R}}^{i,j}, I_{\mathsf{R}}^{i,j}, \mathcal{C}_{\mathsf{R}}^{i,j}\}$ if $i \geqslant 0, j > 0$ there is a finitely bound integer variable $\swarrow_{i,j} \in \mathcal{V}_{\mathsf{R}}^{i,j}$ with $I_{\mathsf{R}}^{i,j}(\searrow_{i,j}) = 0$, and if $i > 0, j \geqslant 0$ there is a finitely bound integer variable $\searrow_{i,j} \in \mathcal{V}_{\mathsf{R}}^{i,j}$ with $I_{\mathsf{R}}^{i,j}(\searrow_{i,j}) = 0$, which record the number of MAVs in the returning state that moved from (i,j) respectively to $(i,j-1)$ and $(i-1,j)$ in the last moment in time. Unlike exploring MAVs, it is often the case that two or more returning MAVs will simultaneously move to the same position. Any MAV in the returning state at some location (i,j) will always move to $(i,j-1)$ or $(i-1,j)$ respectively if $i = 0$ or $j = 0$, otherwise it will move from (i,j) to $(i-1,j)$ with probability $p_L(i-1,j-1)$, or to $(i,j-1)$ with probability $p_R(i-1,j-1)$. If no MAVs in the returning state have moved to (i,j),

and there is no MAV in the node state whose pheromone levels have depleted, then in the next moment in time no MAVs in the returning state will be moving to either of $(i-1,j)$ or $(i,j-1)$. Let $E^{i,j}_{k,l}$ to be the event where given k MAVs in the returning state at (i,j), l MAVs move from (i,j) to $(i-1,j)$ in the next moment in time, and $k-l$ MAVs move from (i,j) to $(i,j-1)$ in the next moment in time. The probability of this event occurring is denoted by $P(E^{i,j}_{k,l})$, and is given by the probability mass function

$$P(E^{i,j}_{k,l}) = \binom{k}{l} p_L(i-1,j-1)^l p_R(i-1,j-1)^{k-l}.$$

For each case where there are $0 < k \leqslant N$ MAVs at (i,j) the event $E^{i,j}_{k,l}$ occurs with probability $P(E^{i,j}_{k,l})$ for $0 \leqslant l \leqslant k$, and a random variable over the possible events clearly follows the binomial distribution.

MAVs beyond the maximum depth. For every $\mathsf{F}_{i,j} = \{\mathcal{V}^{i,j}_{\mathsf{F}}, I^{i,j}_{\mathsf{F}}, \mathcal{C}^{i,j}_{\mathsf{F}}\}$ there is a finitely bound integer variable $n^{\mathsf{F}} \in \mathcal{V}^{i,j}_{\mathsf{F}}$ with $I^{i,j}_{\mathsf{F}}(n^{\mathsf{F}}) = 0$ that records the number of MAVs that are at or beyond this position and a finitely bound integer variable $t \in \mathcal{V}^{i,j}_{\mathsf{F}}$ with $\mathcal{C}^{i,j}_{\mathsf{F}}(t) = 0$ that is used to determine when MAVs should return from (i,j) or beyond. If $i \geqslant 0, j > 0$ there is a finitely bound integer variable $\swarrow_{i,j} \in \mathcal{V}^{i,j}_{\mathsf{F}}$ with $I^{i,j}_{\mathsf{F}}(\swarrow_{i,j}) = 0$ and if $i > 0, j \geqslant 0$ there is a finitely bound integer variable $\searrow_{i,j} \in \mathcal{V}^{i,j}_{\mathsf{F}}$ with $I^{i,j}_{\mathsf{F}}(\searrow_{i,j}) = 0$ which record the number of MAVs in the returning state that moved from (i,j) respectively to $(i,j-1)$ and $(i-1,j)$ in the last moment in time. The variable t is an approximation of the average number of transitions of the model that would be expected to occur between a MAV in the exploring state moving to (i,j) or beyond, and then returning from a position at or beyond (i,j). If no MAV moves to (i,j) then after each transition the variable t is decremented by 1. If there are MAVs at (i,j) and t has decreased to 0, then in the next moment in time n^{F} is decremented by 1 and a single MAV returns to $(i-1,j)$ or $(i,j-1)$

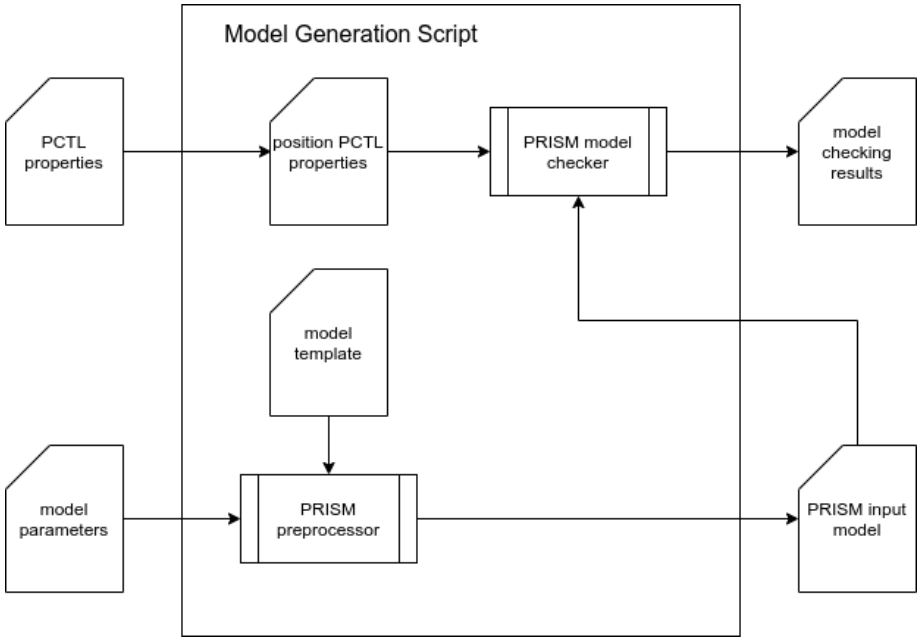


Figure 3.3: The automation of model generation and analysis for ant-based swarming models, given a set of parameters for the model and a set of PCTL properties to check.

with probability $p_L(i,j)$ or $p_R(i,j)$ respectively. For final positions where $i = 0, j = 0$, all MAVs returning from (i,j) will move to $(i,j-1)$ or $(i-1,j)$ respectively.

Automated model generation. Formal models are encoded using the PRISM language described in Section 2.7.1. A script was developed to automate both the generation of the input files for PRISM, and the checking of PCTL properties against these models using the model checker. Figure 3.3 illustrates the automation process. The script takes as input a set of parameters defining the model, and a set of PCTL properties to check against that model.

The model parameters are concrete values for the parameters $N \in \mathbb{N}$, the number of MAVs in the swarm, $U \in \mathbb{N}$, the maximum distance between the target and the base node (in hundreds of metres), and d, the number of discrete values used to record pheromone

levels. The PCTL properties are a set of reachability properties that should be checked against the model. Given an input property the script generates a corresponding property for each position (i, j) in the Y-junction grid. This is described in more detail in the next section.

The PRISM preprocessor[1] is a tool to automate the generation of PRISM models that contain a lot of repetition. The preprocessor accepts template models with undefined constants that can be given concrete values when they are processed. It also allows the use of looping constructs to aid procedural generation. The script uses the preprocessor to instantiate a concrete PRISM input model, using the model parameters and a template model defining how each of the modules described earlier in this section should be generated. The script then checks the PCTL properties against the model, using the PRISM model checker. The results are then aggregated and written to an output file.

3.4 Experiments

In this section the formal model is instantiated, and validated using statistical methods. Properties that could facilitate the deployment of such a swarm are then checked against the instantiated formal model, using both exhaustive and statistical methods.

3.4.1 Model validation

To validate the model statistical model checking was applied using the statistical engine of PRISM by checking properties against systems of large swarms of MAVs where full verification could not be practically applied. For a detailed description of statistical model checking see Section 2.6.

The results obtained were compared to those taken from the simulations conducted

[1] https://www.prismmodelchecker.org/prismpp/

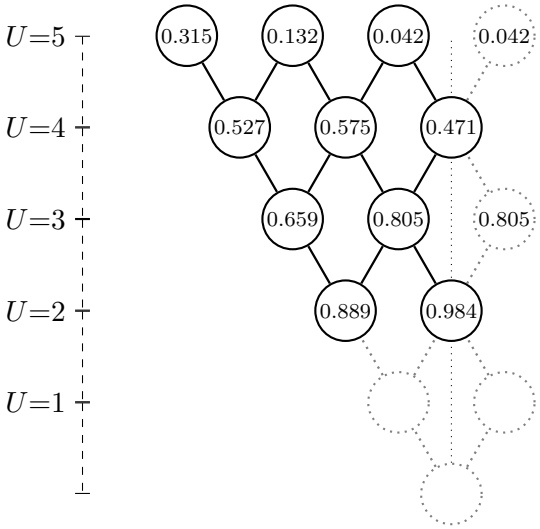


Figure 3.4: The subset of $\mathcal{L}_{\mathsf{user}}$ for which the PCTL property is checked. Here the results correspond to checking the property for $N = 5$.

in [77]. The mean probability of establishing contact with a target within 30 minutes was calculated over a series of 500 simulations for varying swarm sizes. Targets were located at some randomly determined location within a 60 degree arc in a known cardinal direction from the base node at a distance of $\approx$ 200-500 m. In the formal model it was assumed that an MAV has established communication with a target if it has moved to a position at most 100 m distant from the target. Since a target can be located up to $\approx 500\,m$ from the base node, models were generated with $N = 5$ and $U = 5$ for all models. The set of all possible locations at which a target may be located is defined as

$$\mathcal{L}_{\mathsf{user}} = \{(i, j) \mid i, j \in 0 \ldots 5 \text{ and } 1 < i + j \leqslant 5\}.$$

For each $(i, j) \in \mathcal{L}_{\mathsf{user}}$ the statistical engine of PRISM was used to calculate the probability of an MAV moving to (i, j) within 30 minutes (equivalent to 180 transitions in the formal model), by formally specifying this as a probabilistic reachability property in PCTL.

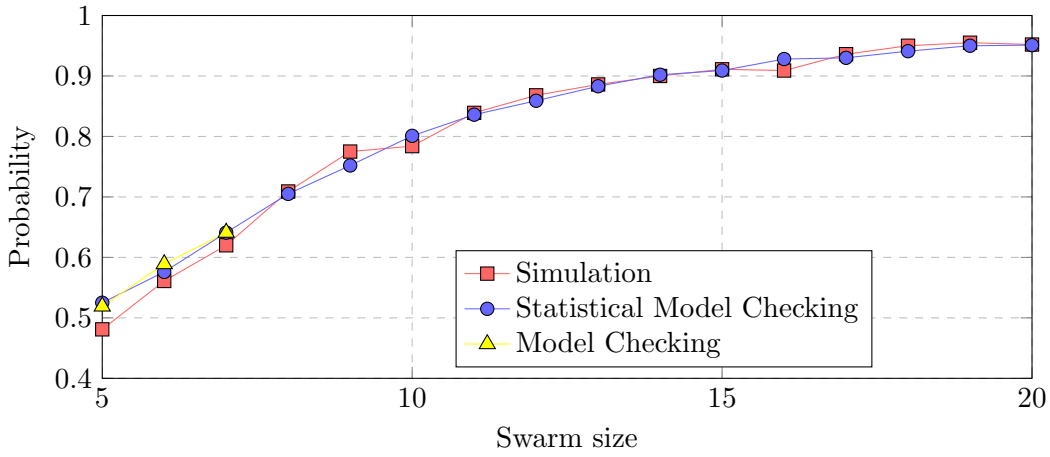


Figure 3.5: The mean probability of finding the target within 30 minutes over 500 trials for simulation and 500 samples for statistical model checking.

Since the grid of positions is symmetrical PCTL properties are only checked for a subset of $\mathcal{L}_{\mathsf{user}}$, as shown in Figure 3.4 – the probability of a MAV moving to some (i, j) is equivalent to the probability of an MAV moving to (j, i). For each(i, j) define $\mathsf{reached}_{i,j}$ as

$$\mathsf{reached}_{i,j} \equiv \begin{cases} \nwarrow_{i-1,j} \vee \nearrow_{,i} j-1 & \text{if } i,j > 0 \\ \nearrow_{i,j-1} & \text{if } i = 0, j > 0 \\ \nwarrow_{i-1,j} & \text{if } i > 0, j = 0. \end{cases}$$

The mean probability over all locations $\bar{P}(\mathsf{reached})$ is then calculated as follows, where we assume the location of the target is taken from a uniform distribution over $\mathcal{L}_{\mathsf{user}}$:

$$\bar{P}(\mathsf{reached}) = \frac{1}{|\mathcal{L}_{\mathsf{user}}|} \sum_{(i,j)\in\mathcal{L}_{\mathsf{user}}} \mathsf{P}[\mathsf{F}^{\leqslant 180}\, \mathsf{reached}_{i,j}].$$

Figure 3.5 compares the results of calculating $\bar{P}(\mathsf{reached})$ for values of $N \in [5 \ldots 20]$ to the results presented in [77]. For each instance of the PMC we induce a DTMC by fixing

the parameter μ to be 0.75, the value used for the simulations in [77]. Statistical model checking results were obtained using 500 simulation samples with an average confidence interval of $\pm 2\%$ based on a 99.0% confidence level. Experiments were conducted on a PC with a 2.20GHz Intel Xeon E5-2420 CPU, 196GB RAM, running Scientific Linux 6.6. There is clearly a strong correlation between both sets of results. For some swarm sizes, namely for $N \in \{5, 9, 10, 16\}$, there was a more pronounced difference between the two values. However, some minor discrepancies were expected due to the relatively low number of simulations/samples used to obtain the results. Exhaustive model checking results are shown only for $N \in [5 \dots 7]$ since the reachable state space of models for $N \geqslant 8$ was too large for the properties to be checked.

3.4.2 Reachability and reachability reward targets

As described in Section 2.3, other measurable aspects of model behaviours can be reasoned about by associating real valued rewards with states and transitions. Properties of interest relating to the expected reward accumulated along some path can then be checked in the model.

By defining a simple reward function for the formal model that associated a reward of 1 with every state and then checking the property $\mathsf{R}[\mathsf{F}\,\mathsf{reached}_{i,j}]$ for every $(i, j) \in \mathcal{L}_{\mathsf{user}}$ the total expected time for the swarm to almost surely establish contact with a target at (i, j) (with probability 1) can be obtained. These values could facilitate the logistics of the deployment of swarms of real MAVs where guaranteed contact with a target is required, and where there is a limited number of MAVs.

The four plots of Figure 3.6 show the total expected time in hours for a deployment of N MAVs, depth U, and lateral distance of the target from the base node, to establish communication with the target target with probability 1. The results obtained by checking

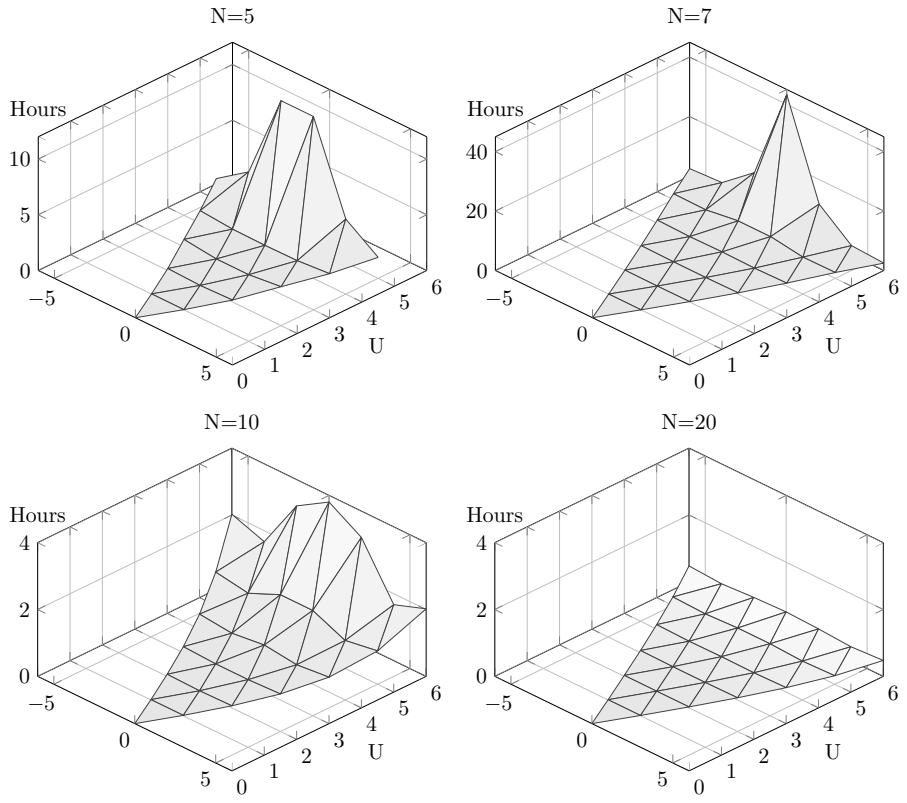


Figure 3.6: Results for checking the property $\mathsf{R}[\mathrm{F}\,\mathsf{reached}_{i,j}]$ for each (i, j).

the property $\mathsf{R}[\mathrm{F}\,\mathsf{reached}_{i,j}]$ for each (i, j). Each vertex on the triangulated meshes for each plot corresponds to a possible position for the target on the Y-junction grids first illustrated in Figure 3.1, and indicates the result that was obtained for checking the property against that position. Larger swarm sizes result in a flattening of the mesh at all points, indicating that the expected time to locate the target decreases as the swarm size increases irrespective of the location of the target.

The four plots of Figure 3.7 show the probability of establishing communication with the target within thirty minutes by a deployment of N MAVs, depth U, and lateral distance of the target from the base node. The results are obtained by checking the property

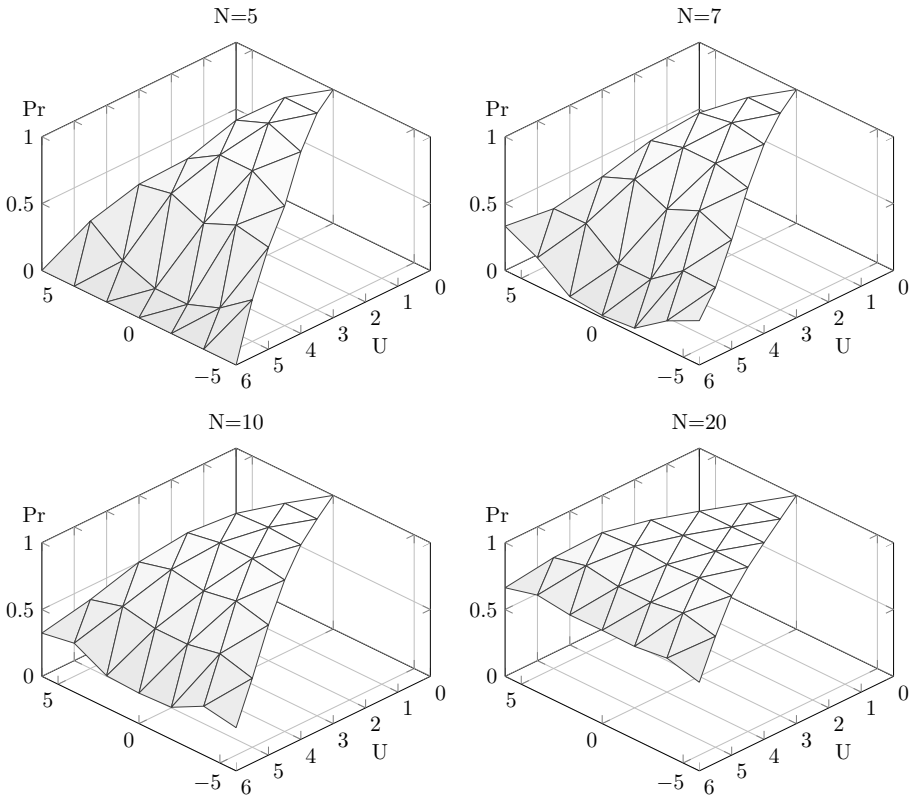


Figure 3.7: Results for checking the property $\mathsf{P}[\mathrm{F}^{\leqslant 180}\,\mathsf{reached}_{i,j}]$ for each (i, j).

$\mathsf{P}[\mathrm{F}^{\leqslant 180}\,\mathsf{reached}_{i,j}]$ for each (i, j). Results for $N > 7$ were obtained using statistical methods over 4000 samples. Again each vertex on the triangulated meshes corresponds to a result obtained by checking the property for a possible position for the target on the Y-junction grids. Larger swarm sizes result in all points of the mesh being elevated, indicating that the probability of locating the target within 180 time steps decreases as the swarm size increases, irrespective of the location of the target.

3.5 Conclusions and further work

Formal probabilistic models were constructed by making some simplifying assumptions given in Section 3.3.2, and a close correspondence between these models and the simulations conducted in the original scenario was clearly shown. The models were then used to check both probabilistic reachability and reachability reward properties, the results of which could be used to plan the deployment of a swarm of MAVs where establishing contact with a user must be guaranteed, or achieved with a probability that exceeds some given threshold. Since battery life greatly impacts the flight duration of MAVs the a priori calculation of the total expected flight time, or total expected distance travelled, for the swarm would ensure that sufficient resources could be made available to ensure that it achieves its objectives. Different reward functions could be developed for a more detailed analysis of the energy consumption of the swarm, since the rate of power consumption for each MAV would differ according to its current mode and rate of communication with other MAVs. Reward functions such as these are applied later in Chapter 5, where the power consumption of a network of synchronising nodes is analysed.

Another natural extension of this work would be to use parametric model checking to investigate the relationship between the parameter μ, that defines the attractiveness of unexplored paths in the grid, and the likelihood of the swarm finding the user at some location. Optimal parameter values for μ could be determined for the maximisation or minimisation of probabilistic reachability or reachability reward properties of interest. Parametric model checking techniques for such an analysis, where changing a parameter does not result in a change in the underlying graph of the PMC have been well studied. The analysis of model parameters that do induce structural changes, for instance the number of MAVs in the swarm N, or the maximum distance at which a user might be located U, has received less attention, and new ideas for their analysis are presented later in Chapter 6.

Chapter 4

Investigating Parametric Influence on Discrete Synchronisation Protocols

Synchronisation is an emergent phenomenon observable throughout the natural world. A noteworthy example is the mutual synchronisation of some species of firefly. Each firefly has an internal biological clock which determines the frequency of its flashes. When a firefly perceives the flashing of others in a colony it updates the rhythm of its own flashes accordingly to bring its own cycle closer to that of its observable neighbours, leading to the emergence of self-organized synchronisation without any centralized control. Natural synchronising systems have inspired the development of protocols for achieving coordination in a diverse range of distributed dynamic systems; in particular swarm robotic systems and WSNs. Applications include detecting faults in members of a robotic swarm [33], synchronising the duty cycles of sensor nodes in a network [148], auto-tuning mobile networks to save energy [16], and coordinating data dissemination for a WSN [27].

The cyclic behaviour of systems where synchrony spontaneously occurs can be modelled as networks of coupled oscillators with similar frequencies. Oscillators are devices that generate oscillations, continual variations in a measure between two or more different states. Essentially an oscillator produces some form of energy at levels that follow a regular rhythmic pattern. Oscillators are *coupled* when some process results in the transferral of energy between them. When the mutual agitation of oscillators takes place only at discrete instances in time the oscillators are *pulse-coupled* [99, 117]. At some distinguished point in the oscillation cycle a pulse-coupled oscillator *fires* and influences other nearby oscillators, and resets its own phase. An oscillator that is perturbed by another oscillator shifts or resets the phase of its own oscillation cycle to more closely match that of its neighbour. Over time this can lead to all oscillators matching phase, and synchronisation is achieved if all oscillators fire synchronously.

Figure 4.1 illustrates the evolution over time of the phase of two pulse-coupled oscillators A and B. Initially, oscillator A is at the start of its cycle, while oscillator B is half way through its cycle. When the phase of each oscillator reaches the threshold (indicated by the top line of both grids) it resets and fires, perturbing the phase of the other oscillator. After each interaction the difference in the phases of the oscillators decreases until they share the same phase and become synchronised.

In nature, the oscillation cycle of oscillators often includes a *refractory period*. The refractory period is an interval in the oscillation cycle during which its phase cannot be perturbed by other firing oscillators. The refractory period can prevent spurious mutual stimulation of the oscillators, which could lead to perpetual asynchrony.

Figure 4.2 again illustrates the evolution over time of oscillators A and B but now the cycle of each oscillator includes a refractory period, indicated by the shaded areas of the grids and denoted by R. Again oscillator A begins at the start of its cycle, while oscillator

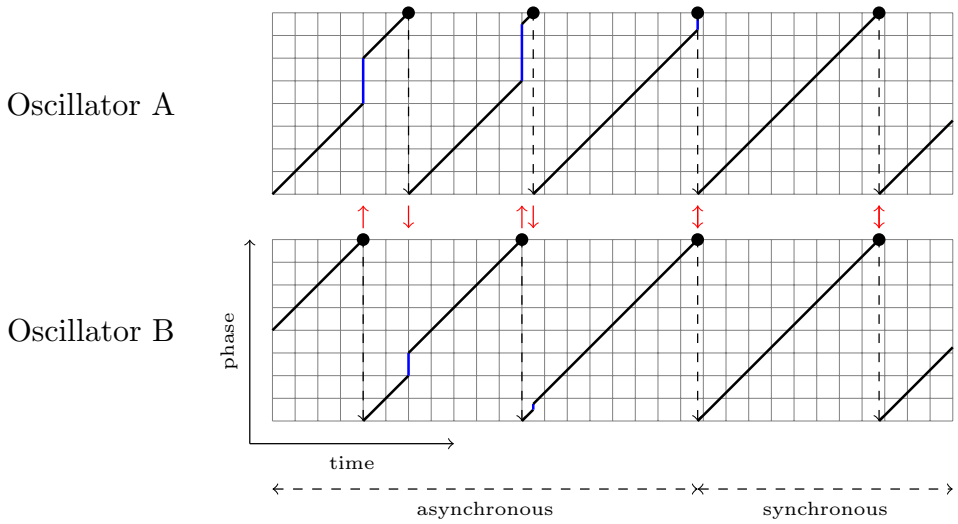


Figure 4.1: Synchronisation of two pulse-coupled oscillators.

B begins half way through its cycle. The initial firing of oscillator A does not perturb the phase of oscillator B, which is within its refractory period. For this example, this leads to synchrony being achieved earlier than was the case for the oscillators without refractory periods in Figure 4.2, however it is sometimes the case that longer refractory periods result in longer times to synchronise. The introduction of a refractory period to oscillators in artificial systems not only helps to achieve synchrony, but can also be thought of as a period during which robots in a swarm, or nodes in a WSN, can turn off their wireless antennas and save energy [147].

The contribution of this chapter is the development of a formal and general model for oscillator synchronisation, which is parametrised by a synchronisation model and a configuration for both oscillators and the network. In contrast to previous applications of model checking to detect synchronisation, this model is discrete. That is, the oscillators interact at discrete moments in time, and their oscillation cycles are defined as sequences of discrete states. For very small devices with limited resources, for instance sensing nodes in a WSN,

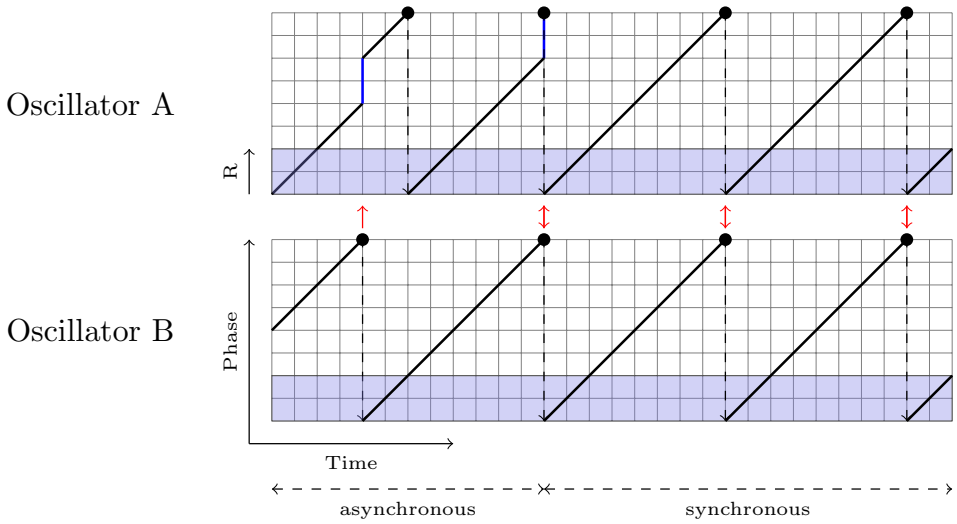


Figure 4.2: Synchronisation of two pulse-coupled oscillators with refractory periods.

it is important to minimise the cost of low-level functionalities, such as synchronisation. Even a floating point number may need too much memory, compared to an implementation with, for example, a four-bit vector. Hence, in our model the oscillators synchronise over a finite set of *discrete* clock values. A PMC can be automatically generated from an instantiation of the formal model, and properties of interest relating to the likelihood of synchronisation, and time taken to achieve it, can be formalised and checked against the model. An analysis of two different models of synchronisation are considered, and for each a family of PMCs for different parameter instances are automatically generated.

This scenario was selected as a case study for the following reasons. As in the previous chapter, the system of interest is composed of homogeneous interacting entities (nodes in a WSN, or robots in a swarm), and the property of interest, the likelihood that a network synchronises, does not require any individual entity to be distinguished from the others. This ensures that entities sharing the same state can be reasoned about as a group, not as individuals, resulting in much smaller formal models, and hence less resources being

required for their analysis. Again, the model introduced here has clearly defined parameters that determine both the structure of the formal model, and stochastic choices over how the model evolves over time.

Related work is discussed in Section 4.1. Section 4.2 presents the dynamics of individual oscillators with discrete oscillation cycles. The general, parametrised population model for a network of oscillators is then formally defined in Section 4.3, and a description of how the corresponding DTMC is constructed is given in Section 4.4. The results of checking synchronisation properties for two concrete instantiations of our formal model are discussed in Section 4.5. A refinement of the population model that allows larger networks to be analysed is then presented and evaluated in Section 4.6. Finally, concluding remarks and suggestions for further work are given in Sect. 4.7.

4.1 Related work

The synchronisation of pulse-coupled oscillators is a well-studied phenomenon. In [117] Mirollo and Strogatz give an intuitive description of how synchrony develops in a system of pulse-coupled oscillators. As the system evolves over time clusters of oscillators sharing the same activation are formed as the firing of oscillators brings their neighbours' firing cycles closer to those of their own. Since all oscillators in a cluster fire at the same time the collective firing strength increases, bringing other oscillators to the firing threshold. Groups of oscillators grow by *absorbing* other oscillators. Larger groups may absorb smaller groups and this process repeats until there is only one group of synchronized oscillators. Mirollo and Strogatz's model was derived from Peskin's model of a cardiac pacemaker [123]. They proved that under certain conditions a network of mutually coupled oscillators always converges – their position within the oscillation cycle eventually coincides. One such assumption was that the network was fully coupled. Later work by Lucarelli and Wang

showed that this assumption could be relaxed, by proving that oscillators would always achieve synchrony if the coupling graph of the network was connected [113].

Such models have been shown to be applicable to the clock synchronisation of wireless sensor nodes and swarms of robots. A pulse-coupled oscillator model was used by Tyrell et al. in [144] to synchronise a network of WSN nodes over the physical layer. The model was shown to be effective, even under the integration of realistic effects such as transmission delays. A similar approach was used in [148] to synchronise the duty cycles of nodes in a WSN, however here the nodes used delayed information from the past to adjust their clocks when receiving synchronisation messages. The method was shown to work well even under adverse network conditions such as message delays and loss, and was tested on an indoor sensor network test bed. An agent based approach was employed by Bojic et al. in [16], where autonomous agents operating on nodes in a network aim to synchronise their actions, and fine tune a telecommunication network. Taniguchi et al. [140] showed that synchronising nodes in a network facilitated data gathering without centralised control, where all nodes leave their idling state at the same time, obtain information from their environment, and transmit this information back to some distinguished base node.

The introduction of a refractory period for oscillators, an interval in the oscillation cycle where that oscillator cannot be perturbed, can help to achieve synchrony and reduce power consumption. Wang et al. [147] investigated refractory periods, by studying the power consumption of nodes in a WSN that synchronise and turn off their wireless antennas to save energy in unison. Refractory periods were also investigated by Breza in [27], where bio-inspired synchronisation protocols were combined with gossiping protocols to coordinate data dissemination in a WSN.

Synchronisation has also been shown to be useful in the domain of swarm robotics. It was used by Christensen et al. in [33] to detect faults in a robotic swarm. Members

of the swarm periodically emit a flash that is perceivable by others, and when all flashes are synchronised a faulty robot that fails to flash can be detected. The synchronisation of mobile swarm robots was also investigated in detail by Perez-Diaz [122], where the effects of robot speed on the rates at which synchronisation was achieved were analysed.

A general formal model of oscillator synchronisation was introduced in [6], where oscillators were modelled as a subclass of timed automata [3], and a model checking algorithm was used to determine the reachability of a synchronised state for distinguished runs of the model. Unlike the work presented here, their analysis was restricted to a network of only three oscillators.

More recently, Heidarian et al. used model checking to analyse clock synchronisation [78]. They analysed the behaviour of synchronisation protocols for WSNs based on time allocation slots for up to four nodes and different topologies, from fully connected networks to line topologies. They modelled the protocol as timed automata [3], and used the model-checker UPPAAL [10] to examine its worst-case behaviour. In contrast to the models presented in this chapter, they use continuous time models, and in particular, they did not model pulse-coupled oscillators.

4.2 Discrete oscillator model

A fully-connected network of pulse-coupled oscillators is considered, where all oscillators have identical dynamics over discrete time. The *phase* of an oscillator i at time t is denoted by $\phi_i(t)$. The phase of an oscillator progresses through a sequence of discrete integer values bounded by some $T \geqslant 1$. The phase progression over time of a single uncoupled oscillator is determined by the successor function, where the phase increases until $\phi_i(t) = TT$, at which point the oscillator *fires*, in the next moment in time where $\phi_i(t+1)$ becomes 1, and the oscillator attempts to broadcast a firing signal to all other oscillators *coupled* to

it. The phase progression of an uncoupled oscillator is cyclic with period T and we refer to one cycle as an *oscillation cycle*.

An oscillator's firing signal perturbs the phase of all oscillators to which it is couples. We use $\alpha_i(t)$ to denote the number of other oscillators that are coupled to oscillator i and will broadcast their firing signal at time t. The strength of the coupling between the oscillators determines the magnitude of the induced perturbation to phase, and is given by the parameter $\epsilon \in \mathbb{R}$. Furthermore, $\mu \in [0, 1]$ is the probability that a *broadcast failure* occurs when an oscillator fires – the attempt to broadcast its firing signal fails (the oscillator still resets its phase to 1). Note that μ is a global parameter, hence the chance of broadcast failure is identical for all oscillators. Observe that $\alpha_i(t)$ is defined globally even though the model is not deterministic.

Definition 12. *The* perturbation function *is an increasing function*

$$\Delta : [1 \dots T] \times \mathbb{N} \times \mathbb{R}_{>0} \to \mathbb{N}$$

that maps the phase of an oscillator i, the number of oscillators that have fired and perturbed i, and a real valued constant defining the strength of the coupling between oscillators, to an integer value corresponding to the induced perturbation to phase.

A *refractory period* can be introduced into the oscillation cycle of each oscillator. A refractory period is an interval of discrete values $[1 \dots R] \subseteq [1 \dots T]$ where $0 \leqslant R \leqslant T$ is the size of the refractory period, such that if $\phi_i(t)$ is inside the interval, for some oscillator i at time t, then i cannot be perturbed by other oscillators to which it is coupled. If $R = 0$ then the interval has zero length and there is no refractory period at all. To be consistent with the literature only refractory periods that occur at the start of the oscillation cycle are considered [147].

Definition 13. *The* refractory function

$$\mathsf{ref} : [1 \dots T] \times \mathbb{N} \to \mathbb{N}$$

is defined as:

$$\mathsf{ref}(\phi, \delta) = \begin{cases} \phi & \textit{if } \phi \in [1 \dots R] \\ \phi + \delta & \textit{otherwise.} \end{cases}$$

The phase is increased by δ if ϕ is outside of the refractory period, otherwise it remains unchanged.

The phase evolution of an oscillator i over time is then defined as follows, where the *update function* and *firing predicate*, respectively denote the updated phase of oscillator i at time t in the next moment in time, and the firing of oscillator i at time t,

$$\begin{aligned} \mathsf{update}_i(t) &= 1 + \mathsf{ref}(\phi_i(t), \Delta(\phi_i(t), \alpha_i(t), \epsilon)), \\ \mathsf{fire}_i(t) &= \mathsf{update}_i(t) > T, \\ \phi_i(t+1) &= \begin{cases} 1 & \text{if } \mathsf{fire}_i(t) \\ \mathsf{update}_i(t) & \text{otherwise.} \end{cases} \end{aligned}$$

4.3 Population model

Let $\mathcal{O} = \{1, \dots, N\}$ be a fully connected network of N identical oscillators with phases in the range $1, \dots, T$, whose dynamics are determined by the perturbation function Δ, and have a refractory period defined by R. The strength of the coupling between the oscillators is given by $\epsilon \in [0, 1]$, and the probability of broadcast failure is given by $\mu \in [0, 1]$. The

population model of the network $\mathcal{O}$ is a tuple

$$\mathcal{P} = (\Delta, N, T, R, \epsilon, \mu).$$

Since all oscillators in the model are behaviourally identical there is no need to distinguish between oscillators sharing the same phase, and reasoning can be applied to groups of oscillators sharing the same local state, instead of individuals. The global state of the model is encoded as a tuple

$$\langle k_1, \ldots, k_T \rangle$$

where each k_Φ is the number of oscillators sharing phase ϕ. The population model does not account for the introduction of additional oscillators to a network, or the loss of existing coupled oscillators, and the population N remains constant.

Definition 14. *A global state of a population model* $\mathcal{P} = (\Delta, N, T, R, \epsilon, \mu)$ *is a* T*-tuple* $\langle k_1, \ldots, k_T \rangle \in [0 \ldots N]^T$*, where* $\sum_{\Phi=1}^{T} k_\Phi = N$*. The set of all global states of* $\mathcal{P}$ *is* $\Gamma(\mathcal{P})$*, or simply* Γ *when* $\mathcal{P}$ *is clear from the context.*

Example 14. *Figure 4.3 shows three global states for an instantiated population model,*

$$\mathcal{P} = (\Delta, 5, 6, 2, 0.15, 0.1),$$

where the synchronisation model of [117] is instantiated by defining the perturbation function as

$$\Delta(\Phi, \alpha, \epsilon) = \lfloor \Phi \, \alpha \, \epsilon \rceil \, .$$

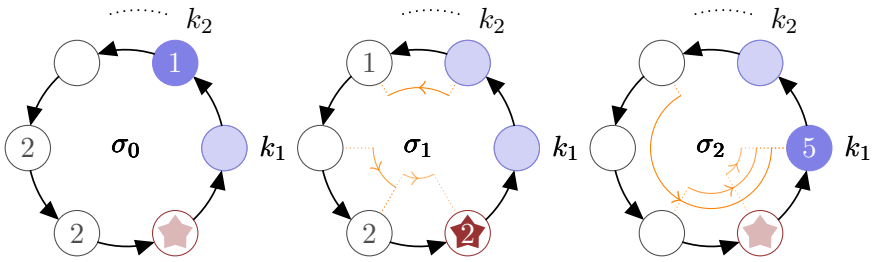


Figure 4.3: Evolution of the global state over three discrete time steps.

The label for each node k_Φ is the number of oscillators with phase Φ. The label is omitted if $k_\Phi = 0$. The starred node records the number of oscillators with phase T. Oscillators at node k_6 are about to fire, and oscillators at nodes k_1 and k_2 are in their refractory period, and cannot be perturbed by the firing of other oscillators. The global states are

$$\sigma_0 = \langle 0, 1, 0, 2, 2, 0 \rangle,$$
$$\sigma_1 = \langle 0, 0, 1, 0, 2, 2 \rangle,$$
$$\sigma_2 = \langle 5, 0, 0, 0, 0, 0 \rangle.$$

Later it is discussed how transitions between these global states are made. Note that directional arrows indicate cyclic direction, and do not represent transitions.

A non-empty set of *failure vectors* is associated with every global state σ, where each failure vector is a tuple of broadcast failures that could occur in σ.

Definition 15. *A* failure vector *is a T-tuple $\mathbf{f} \in ([0 \dots N] \cup \{\star\})^T$. The set of all possible failure vectors is denoted by $\mathcal{F}$.*

Given a failure vector $\mathbf{f} = \langle f_1, \dots, f_T \rangle$, $f_\Phi \in [0 \dots N]$ indicates the number of broadcast failures that occur for all oscillators with a phase of Φ. If $f_\Phi = \star$ then no oscillators with a phase of Φ fire, for all $1 \leqslant \Phi \leqslant T$. Semantically, $f_\Phi = 0$ and $f_\Phi = \star$ differ in that the

former indicates that all (if any) oscillators with phase Φ fire and no broadcast failures occur, while the latter indicates that all (if any) oscillators with a phase of Φ do not fire. If no oscillators fire at all in a global state then there is only one possible failure vector, namely $\{\star\}^T$.

4.3.1 Transitions

The calculation of the set of all possible failure vectors for a global state, and the resulting identification of all successor states of that state, is described later in this section. Firstly, it must be shown how the single successor state of a global state can be calculated, given a single failure vector for that state.

Absorptions. For real deployments of synchronisation protocols it is often the case that the duration of a single oscillation cycle will be at least several seconds [33, 121]. The perturbation induced by the firing of a group of oscillators may lead to groups of other oscillators to which they are coupled firing in turn. The firing of these other oscillators may then cause further oscillators to fire, and so forth, leading to a "chain reaction", where each group of oscillators triggered to fire is *absorbed* by the initial group of firing oscillators. Since the whole chain reaction of absorptions may occur within just a few milliseconds, and in our model the oscillation cycle is a sequence of discrete states, when a chain reaction occurs the phases of all perturbed oscillators are updated at one single time step.

Since only fully connected networks of oscillators are considered here, two oscillators sharing the same phase will have their phase updated to the same value in the next time step. They will always perceive the same number of other oscillators firing. Therefore, for each phase Φ the following function is defined,

$$\alpha^{\Phi} : \Gamma \times \mathcal{F} \to [1 \ldots N],$$

where $\alpha^{\Phi}(\sigma, \mathbf{f})$ is the number of oscillators with a phase greater than Φ perceived to be firing by oscillators with phase Φ in some global state σ, incorporating the broadcast failures defined in the failure vector $\mathbf{f}$. This allows the aforementioned chain reactions of firing oscillators to be encoded. Note that this encoding of chain reactions results in a global semantics that differs from typical parallelisation operations, for example, the construction of the crossproduct of the individual oscillators.

Definition 16. *Given a global state $\sigma = \langle k_1, \ldots, k_T \rangle$ and a failure vector $\mathbf{f} = \langle f_1, \ldots, f_T \rangle$, the following mutually recursive definitions show how $\alpha^1(\sigma, \mathbf{f}), \ldots, \alpha^T(\sigma, \mathbf{f})$ are calculated, and how functions introduced in the previous section are modified to indicate the update in phase, and firing, of* all *oscillators sharing the same phase Φ. Observe that to calculate any $\alpha^{\Phi}(\sigma, \mathbf{f})$ only definitions for phases greater than Φ are referenced and the base case is $\Phi = T$, that is, values are computed from T down to* 1.

$$\mathsf{update}^{\Phi}(\sigma, \mathbf{f}) = 1 + \mathsf{ref}(\Phi, \Delta(\Phi, \alpha^{\Phi}(\sigma, \mathbf{f}), \epsilon))$$

$$\mathsf{fire}^{\Phi}(\sigma, \mathbf{f}) = \mathsf{update}^{\Phi}(\sigma, \mathbf{f}) > T$$

$$\alpha^{\Phi}(\sigma, \mathbf{f}) = \begin{cases} 0 & \text{if } \Phi = T \\ \alpha^{\Phi+1}(\sigma, \mathbf{f}) + k_{\Phi+1} - f_{\Phi+1} & \text{if } \Phi < T, f_{\Phi+1} \neq \star \text{ and } \mathsf{fire}^{\Phi+1}(\sigma, \mathbf{f}) \\ \alpha^{\Phi+1}(\sigma, \mathbf{f}) & \text{otherwise} \end{cases}$$

Transition function. The transition function that maps phase values to their updated values in the next time step is now defined. Since there is no differentiation between oscillators with the same phase, only a single value for their perturbation needs to be calculated.

Definition 17. *The* phase transition function $\tau \colon \Gamma \times [1 \ldots T] \times \mathcal{F} \to \mathbb{N}$ *maps a global state, a phase Φ, and some possible failure vector $\mathbf{f}$ for σ, to the updated phase in the next*

discrete time step, with respect to the broadcast failures defined in $\mathbf{f}$*, and is defined as*

$$\tau(\sigma, \Phi, \mathbf{f}) = \begin{cases} 1 & \textit{if } \mathsf{fire}^{\Phi}(\sigma, \mathbf{f}) \\ \mathsf{update}^{\Phi}(\sigma, \mathbf{f}) & \textit{otherwise.} \end{cases}$$

Let $\mathcal{U}_{\Phi}(\sigma, \mathbf{f})$ be the set of phase values Ψ where all oscillators with phase Φ in σ will have their phase updated to Ψ in the next time step, with respect to the broadcast failures defined in $\mathbf{f}$. Formally,

$$\mathcal{U}_{\Phi}(\sigma, \mathbf{f}) = \{\Psi \mid \Psi \in [1 \dots T] \wedge \tau(\sigma, \Psi, \mathbf{f}) = \Phi\}.$$

The successor state of a global state σ can now be calculated, and the evolution of the model over time can be defined.

Definition 18. *The* successor function $\vec{\mathsf{succ}}\colon \Gamma \times \mathcal{F} \to \Gamma$ *maps a global state* σ *and a failure vector* $\mathbf{f}$ *to a global state* σ'*, and is defined as*

$$\vec{\mathsf{succ}}(\langle k_1, \dots, k_T \rangle, \mathbf{f}) = \Big\langle \sum_{\Psi \in \mathcal{U}_1(\sigma, \mathbf{f})} k_{\Psi}, \dots, \sum_{\Psi \in \mathcal{U}_T(\sigma, \mathbf{f})} k_{\Psi} \Big\rangle.$$

Example 15. *Consider again Figure 4.3 with global states* $\sigma_0 = \langle 0, 1, 0, 2, 2, 0 \rangle$*,* $\sigma_1 = \langle 0, 0, 1, 0, 2, 2 \rangle$*,* $\sigma_2 = \langle 5, 0, 0, 0, 0, 0 \rangle$*. Observe that in the global state* σ_0 *no oscillators will fire since* $k_6 = 0$*. Therefore there is one possible failure vector for* σ_0*, namely* $\mathbf{f} = \{\star\}^6$*. Since no oscillators fire the dynamics of the oscillators are determined solely by the standalone dynamics, and all oscillators simply increase their phase by* 1 *in the next time*

step. Now consider the global state σ_1 *and*

$$\mathbf{f} = \langle \star, \star, 0, 0, 0, 1 \rangle,$$

a possible failure vector for σ_1*, indicating that oscillators with phases in* $[3..6]$ *will fire and one broadcast failure will occur for one of the two oscillators that will fire with phase* 6*. Despite the broadcast failure occurring, a chain reaction will occur as the perturbation induced by the single oscillator with phase* 6 *that does not suffer a broadcast failure is sufficient to cause the two oscillators with phase* 5 *to fire also. The combined perturbation induced by the firing of all three oscillators will cause the final oscillator with phase* 3 *to fire. All oscillators are therefore absorbed into the initial group of firing oscillators. Since* $\mathsf{fire}^6(\sigma_1, \mathbf{f})$ *holds*

$$\alpha^5(\sigma_1, \mathbf{f}) = \alpha^6(\sigma_1, \mathbf{f}) + k_6 - f_6 = 0 + 2 - 1 = 1.$$

Similarly since $\mathsf{fire}^5(\sigma_1)$ *holds,*

$$\alpha^4(\sigma_1, \mathbf{f}) = \alpha^5(\sigma_1, \mathbf{f}) + k_5 = 1 + 2 = 3.$$

Then by continuing to calculate $\alpha^\Phi(\sigma_1, \mathbf{f})$ *for* $3 \geqslant \Phi \geqslant 1$*, it can be concluded that*

$$\mathcal{U}_1(\sigma_1, \mathbf{f}) = \{6, 5, 4, 3\},$$
$$\mathcal{U}_6(\sigma_1, \mathbf{f}) = \mathcal{U}_5(\sigma_1, \mathbf{f}) = \mathcal{U}_4(\sigma_1, \mathbf{f}) = \emptyset.$$

Then since $R = 2$,

$$\mathcal{U}_3(\sigma_1, \mathbf{f}) = \{2\},$$
$$\mathcal{U}_2(\sigma_1, \mathbf{f}) = \{1\}.$$

Finally, the successor of σ_1 is calculated as

$$\sigma_2 = \vec{\mathsf{succ}}(\langle 0, 0, 1, 0, 2, 2\rangle, \mathbf{f}) = \langle k_6 + k_5 + k_4 + k_3, k_1, k_2, 0, 0, 0\rangle = \langle 5, 0, 0, 0, 0, 0\rangle.$$

Lemma 1. *The number of oscillators is invariant during transitions, i.e., the successor function only creates tuples that are states of the given model. Formally, let $\sigma = \langle k_1, \ldots, k_T\rangle$ and $\sigma' = \langle k'_1, \ldots, k'_T\rangle$ be two states of a population model $\mathcal{P}$ such that $\sigma' = \vec{\mathsf{succ}}(\sigma, \mathbf{f})$, where $\mathbf{f}$ is some possible failure vector for σ. Then*

$$\sum_{\Phi=1}^{T} k_\Phi = \sum_{\Phi=1}^{T} k'_\Phi = N.$$

Proof. By construction, it is clear that for all $1 \leqslant \Phi \leqslant T$,

$$\tau(\sigma, \Phi, \mathbf{f}) \in [1 \ldots T].$$

Hence for all Ψ with $1 \leqslant \Psi \leqslant T$, there is a Φ such that $\Psi \in \mathcal{U}_\Phi(\sigma, \mathbf{f})$. This implies

$$\bigcup_{\Phi=1}^{T} \mathcal{U}_\Phi(\sigma, \mathbf{f}) = [1 \ldots T].$$

Furthermore, there cannot be more than one Φ such that $\Psi \in \mathcal{U}_\Phi(\sigma, \mathbf{f})$, since τ is functional.

Therefore,

$$\sum_{\Phi=1}^{T} k'_\Phi = \sum_{\Phi=1}^{T} \sum_{\Psi \in \mathcal{U}_\Phi(\sigma,\mathbf{f})} k_\Psi = \sum_{\Phi=1}^{T} k_\Phi = N,$$

and the lemma is proved. □

4.3.2 Failure vector calculation

The set of all possible failure vectors for a global state is constructed by considering every group of oscillators in decreasing order of phase. At each stage it is determined if the oscillators would fire. If they fire then each outcome where any, all, or none of the firings result in a broadcast failure is considered. A corresponding value is then added to a partially calculated failure vector and the next group of oscillators with a lower phase is considered. If the oscillators do not fire then there is nothing left to do, since by Definition 12 Δ is increasing, therefore all oscillators with a lower phase will also not fire. The partial failure vector can then be padded with $\star$ appropriately to indicate that no failure could happen since no oscillators fired.

Table 4.1 illustrates how a possible failure vector for global state σ_1 in Figure 4.3 is iteratively constructed. The first three columns respectively indicate the current iteration i, the global state σ_1 with the currently considered oscillators indicated, and the elements

Table 4.1: Construction of a possible failure vector for a global state $\sigma_1 = \langle 0, 0, 1, 0, 2, 2 \rangle$.

iteration (i)	σ_1	failure vector $\mathbf{f}$	fired	branches
0	$\langle 0, 0, 1, 0, 2, 2 \rangle$	$\langle\rangle$	–	*false*
1	$\langle 0, 0, 1, 0, 2, \underline{2} \rangle$	$\langle 1 \rangle$	*true*	*true*
2	$\langle 0, 0, 1, 0, \underline{2}, 2 \rangle$	$\langle 0, 1 \rangle$	*true*	*true*
3	$\langle 0, 0, 1, \underline{0}, 2, 2 \rangle$	$\langle 0, 0, 1 \rangle$	*true*	*false*
4	$\langle 0, 0, \underline{1}, 0, 2, 2 \rangle$	$\langle 0, 0, 0, 1 \rangle$	*true*	*true*
5	$\langle 0, \underline{0}, 1, 0, 2, 2 \rangle$	$\langle \star, \star, 0, 0, 0, 1 \rangle$	*false*	–

of the failure vector $\mathbf{f}$ computed so far. The fourth column is *true* if the oscillators with phase $T + 1 - i$ would fire given the broadcast failures in the partial failure vector. All outcomes of any or all firings resulting in broadcast failure must be considered. The final column therefore indicates whether the value added to the partial failure vector in the current iteration is the only possible value (*false*), or if a choice from one of several possible values (*true*).

Initially there is an empty partial failure vector. At the first iteration there are 2 oscillators with a phase of 6. These oscillators will fire so each case where $0, 1$ or 2 broadcast failures occur must be considered. Here 1 broadcast failure is chosen, which is then added to the partial failure vector. At iteration 2, oscillators with a phase of 5 fire, and again each case with $0, 1$ or 2 broadcast failures occur must be considered; here 0 is chosen. At iteration 3 oscillators with a phase of 4 would have fired, but since there are no oscillators with a phase of 4 there is only one possible value to add to the partial failure vector, namely 0. At iteration 4 a single oscillator with a phase of 3 fires, and the case where the firing did not result in a broadcast failure is chosen. In the final iteration oscillators with a phase of 2 do not fire, it can be concluded that oscillators with a phase of 1 also do not fire, and the partial failure vector can be padded appropriately with $\star$.

Let g be a family of functions indexed by Φ, where each g_Φ takes as parameters some global state σ, and V, a vector of length $T - \Phi$. V represents all broadcast failures for all oscillators with a phase greater than Φ. The function g_Φ then computes the set of all possible failure vectors for σ with suffix V. The notation $v^\frown v'$ is used to indicate vector concatenation.

Definition 19. *For* $1 \leqslant \Phi \leqslant T$ *let*

$$g_\Phi \colon \Gamma \times [0 \dots N]^{T-\Phi} \to \mathbb{P}([0 \dots N] \cup \{\star\})^T)$$

be the family of functions indexed by Φ*, where* $\sigma = \langle k_1, \ldots, k_T \rangle$ *and*

$$g_\Phi(\sigma, V) = \begin{cases} \bigcup_{i=0}^{k_\Phi} g_{\Phi-1}(\sigma, \langle i \rangle^\frown V) & \text{if } 1 < \Phi \leqslant T \text{ and } \mathsf{fire}^\Phi(\sigma, \{\star\}^\Phi{}^\frown V) \\ \bigcup_{i=0}^{k_1} \{\langle i \rangle^\frown V\} & \text{if } \Phi = 1 \text{ and } \mathsf{fire}^1(\sigma, \langle \star \rangle^\frown V) \\ \left\{\{\star\}^\Phi{}^\frown V\right\} & \text{otherwise.} \end{cases}$$

The set of all possible failure vectors for σ *is then given by* $g_T(\sigma, \langle\rangle)$*, and for every failure vector* $\mathbf{f} = \langle f_1, \ldots, f_T \rangle$ *in that set* $f_i \leqslant k_i$ *for* $1 \leqslant i \leqslant T$ *and* $f_i = \star$ *implies* $f_j = \star$ *for* $1 \leqslant j < i$*.*

Definition 20. *Given a global state* $\sigma \in \Gamma$ *let* $\mathcal{F}_\sigma$ *be the set of all possible failure vectors for that state, where* $\mathcal{F}_\sigma = g_T(\sigma, \langle\rangle)$*, and let* $\mathsf{Next}(\sigma)$ *be the set of all successor states of* σ*, where* $\mathsf{Next}(\sigma) = \{\vec{\mathsf{succ}}(\sigma, \mathbf{f}) \mid \mathbf{f} \in \mathcal{F}_\sigma\}$*.*

Note that for some global states $|\mathsf{Next}(\sigma)| < |\mathcal{F}_\sigma|$, since it may be the case that $\vec{\mathsf{succ}}(\sigma, \mathbf{f}) = \vec{\mathsf{succ}}(\sigma, \mathbf{f}')$ for some $\mathbf{f}, \mathbf{f}' \in \mathcal{F}_\sigma$ with $\mathbf{f} \neq \mathbf{f}'$.

Given a global state σ and a failure vector $\mathbf{f} \in \mathcal{F}_\sigma$, the probability of a transition being made to the global state $\vec{\mathsf{succ}}(\sigma, \mathbf{f})$ in the next time step can now be computed. Recall that μ is the probability with which a broadcast failure occurs. Firstly, let $\mathsf{PMF}\colon [1 \ldots N] \times [1 \ldots N] \to [0, 1]$ be the probability mass function, where $\mathsf{PMF}(n, b)$ gives the probability of b broadcast failures occurring given that n oscillators fire, given by

$$\mathsf{PMF}(n, b) = \mu^b (1 - \mu)^{n-b} \binom{n}{b}.$$

The function mapping a broadcast failure vector $\mathbf{f}$ for σ, to the probability of the

failures in $\mathbf{f}$ occurring can then be defined as

$$P_\tau \colon \Gamma \times \mathbf{f} \to [0, 1],$$

where

$$P_\tau(\langle k_1, \ldots, k_T \rangle, \langle f_1, \ldots, f_T \rangle) = \prod_{\Phi=1}^{T} \begin{cases} \mathsf{PMF}(k_\Phi, f_\Phi) & \text{if } f_\Phi \neq \star \\ 1 & \text{otherwise.} \end{cases}$$

Lemma 2. *For any global state σ, P_τ is a discrete probability distribution over $\mathcal{F}_\sigma$. Formally, $\sum_{\mathbf{f} \in \mathcal{F}_\sigma} P_\tau(\sigma, \mathbf{f}) = 1$.*

Proof. Given a global state $\sigma = \langle k_1, \ldots, k_T \rangle$ a tree of depth T can be constructed where each leaf node is labelled with a possible failure vector for σ, and each node Λ at depth Φ is labelled with a vector of length Φ corresponding to the last Φ elements of a possible failure vector for σ. The label of a node Λ is denoted by $V(\Lambda)$. Each node Λ_e is labelled with $\langle e \rangle^\frown V(\Lambda)$. The tree is iteratively constructed, starting with the root node, root, at depth 0, which is labelled with the empty tuple $\langle\rangle$. For each node Λ at depth $0 \leqslant \Phi < T$ the children of Λ are constructed as follows:

1. If oscillators with phase Φ fire the sample space $\mathcal{E} = \{0, \ldots, k_\Phi\}$ is a set of disjoint events, where each $e \in \mathcal{E}$ is the event where e broadcast failures occur, given that k_Φ oscillators fired. For each $e \in \mathcal{E}$ there is a child Λ_e of Λ with label $\langle e \rangle^\frown V(\Lambda)$, and the edge from Λ to Λ_e is labelled with $\mathsf{PMF}(k_\Phi, e)$.

2. If oscillators with phase Φ do not fire then Λ has a single child $\Lambda_\star$ labelled with $\langle \star \rangle^\frown V(\Lambda)$, and the edge from Λ to $\Lambda_\star$ is labelled with 1.

The label of an edge from a node Λ to its child Λ' is denoted by $L(\Lambda, \Lambda')$. For case 2 if

oscillators with phase Φ do not fire then oscillators with any phase $\Psi < \Phi$ will also not fire, since from Definition 12 Δ is an increasing function. Hence, all descendants of Λ will also have a single child, with an edge labelled with 1, and each node is labelled with the label of its parent, prefixed with $\langle\star\rangle$.

After constructing the tree there is a vector of length T associated with each leaf node, corresponding to a failure vector for σ. The set $\mathcal{F}_\sigma$ of all possible failure vectors for σ is therefore the set of all vectors labelling leaf nodes. The product of all labels on edges along the path from Λ back to the root is denoted by $P^{\downarrow}(\Lambda)$. Given a global state $\sigma = \langle k_1, \ldots, k_T\rangle$ and a failure vector $\mathbf{f} = \langle f_1, \ldots, f_T\rangle \in \mathcal{F}_\sigma$ labelling some leaf node Λ at depth T,

$$P^{\downarrow}(\Lambda) = 1 \cdot \prod_{\Phi=1}^{T} \begin{cases} \mathsf{PMF}(k_\Phi, f_\Phi) & \text{if } f_\phi\star \\ 1 & \text{otherwise} \end{cases} = P_\tau(\sigma, \mathbf{f}).$$

Let Depth^Φ denote the set of all nodes at depth Φ. The equality $\sum_{d\in\mathsf{Depth}^\Phi} P^{\downarrow}(d) = 1$ is shown by induction on Φ. For $\Phi = 0$, i.e. $\mathsf{Depth}^\Phi = \{\mathsf{root}\}$, the property holds by definition. Now assume that $\sum_{d\in\mathsf{Depth}^\Phi} P^{\downarrow}(d) = 1$ holds for some $0 \leqslant \Phi < T$. Let Λ be some node in Depth^Φ, and let Child^Λ be the set of all children of Λ. Consider the following two cases: If oscillators with phase Φ do not fire then $|\mathsf{Child}^\Lambda| = 1$, and $L(\Lambda, c) = 1$ for the only $c \in \mathsf{Child}^\Lambda$. If oscillators with phase Φ fire observe that PMF is a probability mass function for a random variable defined on the sample space $\mathcal{E} = \{0, \ldots, k_\Phi\}$. In either case $\sum_{c\in\mathsf{Child}^\Lambda} L(\Lambda, c) = 1$. Note that

$$\mathsf{Depth}^{\Phi+1} = \bigcup_{d\in\mathsf{Depth}^\Phi} \mathsf{Child}^d,$$

and recall that $L(d,c)P^{\downarrow}(d) = P^{\downarrow}(c)$. Therefore,

$$\sum_{d \in \mathsf{Depth}^{\Phi+1}} P^{\downarrow}(d) = \sum_{d \in \mathsf{Depth}^{\Phi}} \sum_{c \in \mathsf{Child}^{d}} L(d,c)P^{\downarrow}(d) = \sum_{d \in \mathsf{Depth}^{\Phi}} \left(P^{\downarrow}(d) \sum_{c \in \mathsf{Child}^{d}} L(d,c) \right).$$

Since $\sum_{c \in \mathsf{Child}^{d}} L(d,c) = 1$ for each $d \in \mathsf{Depth}^{\Phi}$, and from the induction hypothesis,

$$\sum_{d \in \mathsf{Depth}^{\Phi}} \left(P^{\downarrow}(d) \sum_{c \in \mathsf{Child}^{d}} L(d,c) \right) = \sum_{d \in \mathsf{Depth}^{\Phi}} P^{\downarrow}(d) = 1.$$

It has already been shown that $P^{\downarrow}(\Lambda) = P_{\tau}(\sigma, \mathbf{f})$ for any leaf node Λ labelled with a failure vector $\mathbf{f}$, and since the set of all labels for leaf nodes is $\mathcal{F}_{\sigma}$ it can be concluded that

$$\sum_{\mathbf{f} \in \mathcal{F}_{\sigma}} P_{\tau}(\sigma, \mathbf{f}) = \sum_{d \in \mathsf{Depth}^{T}} P^{\downarrow}(d) = 1.$$

This proves the lemma. □

Example 16. *Consider again the global states*

$$\sigma_1 = \langle 0, 0, 1, 0, 2, 2 \rangle,$$
$$\sigma_2 = \langle 5, 0, 0, 0, 0, 0 \rangle,$$

given in Figure 4.3, of the population model instantiated in Example 14, and the failure vector

$$\mathbf{f} = \langle \star, \star, 0, 0, 0, 1 \rangle$$

given in Example 15, noting that $\mathbf{f} \in \mathcal{F}_{\sigma_1}$, $\overrightarrow{\mathsf{succ}}(\sigma_1, \mathbf{f}) = \sigma_2$, *and* $\mu = 0.1$. *The probability*

of a transition being made from σ_1 to σ_2 is calculated as

$$\begin{aligned} P_\tau(\langle 0,0,1,0,2,2\rangle, \langle \star,\star,0,0,0,1\rangle) &= 1 \cdot 1 \cdot \mathsf{PMF}(1,0) \cdot \mathsf{PMF}(0,0) \cdot \mathsf{PMF}(2,0) \cdot \mathsf{PMF}(2,1) \\ &= 1 \cdot 1 \cdot 0.9 \cdot 1 \cdot 0.81 \cdot 0.18 = 0.13122. \end{aligned}$$

The evolution of the global state of a population model over time.can now be defined. A path of a population model $\mathcal{P}$ is an infinite sequence of global states $\sigma_0\sigma_1\sigma_2\cdots$, where $\sigma_{i+1} \in \mathsf{Next}(\sigma_i)$ for all $i \geqslant 0$.

4.3.3 Synchronisation

When all oscillators in a global state of a population model have the same phase in that state the state is *synchronised*. Formally, a global state $\sigma = \langle k_1, \ldots, k_T\rangle$ is *synchronised* if, and only if, there is some $\Phi \in [1 \ldots T]$ such that $k_\Phi = N$. Hence, for all $\Phi' \neq \Phi$, $k_{\Phi'} = 0$. A path of the run *synchronises* if, and only if, there exists some state along that path that is synchronised. A population model $\mathcal{P}$ synchronises if, and only if, all possible runs of the model synchronise. In Figure 4.3 global state σ_2 is synchronised, since $k_1 = N$.

4.4 Model generation

Given a population model $\mathcal{P} = (\Delta, N, T, R, \epsilon, \mu)$, a PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ is constructed as follows. First, the set of model parameters is set to $\mathcal{X} = \{\mu\}$. The set of states $\mathcal{S}$ is the set $\Gamma \cup \{s_0\}$. In the initial state s_0 all oscillators are considered to be *unconfigured*, and have not yet been assigned a value for their phase.

Multinomial coefficients are used to calculate the likelihood of permutations of a population of oscillators, for which the standard definition is provided.

Definition 21. *The* multinomial coefficient *is an extension of the binomial coefficient*

that gives the number of ordered permutations of the elements of a multiset. Given a finite multiset $\mathcal{M}$, a permutation is an ordered arrangement of its elements, where each element appears a number of times equal to its multiplicity in $\mathcal{M}$. The number of permutations of $\mathcal{M}$ is given by

$$\binom{n}{m_1, m_2, \ldots, m_i} = \frac{n!}{m_1! m_2! \cdots m_i!} = \binom{m_1}{m_1}\binom{m_1+m_2}{m_2}\cdots\binom{m_1+m_2+\ldots+m_i}{m_i},$$

where $m_1, \ldots m_i$ are the multiplicities of the elements of $\mathcal{M}$ and n is the sum of those multiplicities.

For each $s \in \mathcal{S}$, where $s \in \Gamma$ and $s = \langle k_1, \ldots, k_T \rangle$, we define

$$\mathbf{P}(s_0, s) = \frac{1}{T^N}\binom{N}{k_1, \ldots, k_T} \tag{4.1}$$

to be the probability of moving from s_0 to a state where k_Φ arbitrary oscillators are configured with the phase value Φ for $1 \leqslant \Phi \leqslant T$. The multinomial coefficient defines the number of possible assignments of phases to distinct oscillators that result in the global state σ. The fractional coefficient normalises the multinomial coefficient with respect to the total number of possible assignments of phases to all oscillators. For every $s \in \mathcal{S} \setminus \{s_0\}$ the probability of taking a transition to s', a possible successor of s, is

$$\mathbf{P}(s, s') = P_\tau(s, \mathbf{f}), \tag{4.2}$$

where $s' = \vec{\mathsf{succ}}(s, \mathbf{f})$ for some $\mathbf{f} \in \mathcal{F}_s$. For all other $s \in \mathcal{S} \setminus \{s_0\}$ and $s' \in \mathcal{S}$, where $s \neq s'$ and $s' \notin \mathsf{Next}(s)$, $\mathbf{P}(s, s') = 0$. The set of labels of each $s = \langle k_1, \ldots, k_T \rangle \in \mathcal{S} \setminus \{s_0\}$ is set

to $L(s) = \{\mathsf{synchronised}\}$ if, and only if,

$$\bigvee_{\Phi=1}^{T} k_\Phi = N$$

holds for that state, or is set to $L(s) = \emptyset$ otherwise. Observe that the set of all states labelled with synchronised form a bottom strongly connected component of the underlying graph $\mathcal{G}_{\mathcal{D}_\mathcal{X}}$, since the successor of any synchronised state is also synchronised.

Automated model generation. The PMC is encoded using the PRISM language described in Section 2.7.1, where a global state is a valuation for T finitely bound integer variables ranging over $[1 \ldots N]$. A script was developed to automate both the generation of the input files for PRISM, and the checking of PCTL properties against these models using the model checker. Figure 4.4 illustrates the automation process. The script takes as input a set of parameters defining the model, and a set of PCTL properties to check against that model.

The model parameters given as input to the system consist of sets of valuations for the parameters N, T, R, ϵ, and μ, and a definition for the perturbation function Δ. Given a valuation for each of the model parameters, the formal model is programmatically generated, where the probabilities for transitions between states are calculated directly as defined in equations (4.1) and (4.2). A model is generated for all parameter instantiations taken from the cartesian product of the sets of valuations for each of the parameters, and the PCTL properties are checked against the model using PRISM. User-specified output – for example the result, or model checking time – is written to a file that can be used by statistical analysis tools. A sample file generated for $N = 4$, $T = 5$, $\epsilon = 0.1$, $R = 1$, and with the perturbation function defined in Section 4.5.1, is given in Appendix A.

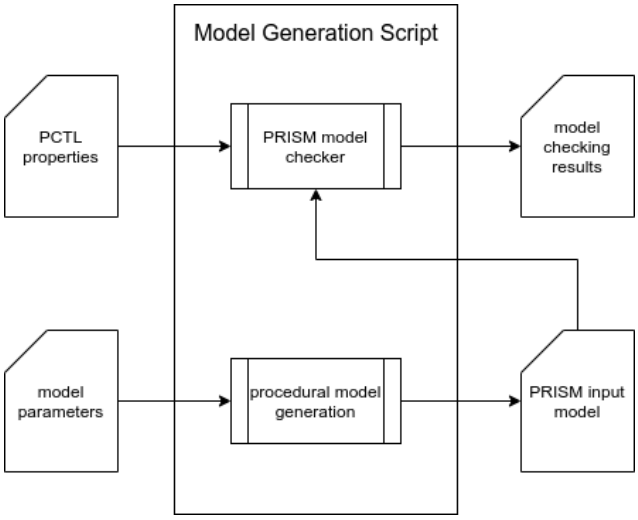


Figure 4.4: The automation of model generation and analysis for synchronisation models, given value ranges for model parameters, and a set of PCTL properties to check.

4.5 Evaluation

This section discusses the properties of two synchronisation models taken from the literature. The formal model defined in Section 4.3 is instantiated for each synchronisation model by providing a concrete definition for the perturbation function. Properties of interest are then checked for different model instances using the model checker PRISM, and the effects of varying parameters relating to the network (N, ϵ), oscillator dynamics (T, R), and environmental factors (μ), are discussed. Other case studies could also be considered for different models of synchronisation where the dynamics of oscillators, and their interactions, can be described by some perturbation function.

The properties of interest are firstly, the probability a model eventually synchronises, and secondly, the time needed to achieve synchronisation. The first property can be for-

malised in PCTL as

$$\mathsf{P}[\mathrm{F}\ \mathsf{synchronised}] \equiv \mathsf{P}\left[\mathrm{F}\left(\bigvee_{\Phi=1}^{T} k_\Phi = N\right)\right],$$

the probability that an execution path of the model leads to a state that is labelled with synchronised – that is, all oscillators in that state have the same phase.

To formalise the second property – the expected time taken to achieve synchronisation, if it is achieved at all – a reward function is first defined for the PMC, as described in Section 2.3. Given the PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ for a population model $\mathcal{P}$ let $\mathbf{R}$ be a reward function for $\mathcal{D}_\mathcal{X}$, where the reward for each state $s \in \mathcal{S}$ is given by

$$\mathbf{R}(s) = \begin{cases} \frac{1}{T} & \text{if } s \neq s_0 \text{ and } s \not\models \mathsf{synchronised} \\ 0 & \text{otherwise.} \end{cases}$$

For all $s, s' \in \mathcal{S}, \mathbf{R}(s, s') = 0$. By assigning a reward of $\frac{1}{T}$ to each state where oscillators are configured, and where the system is not synchronised, the accumulated reward along an execution path yields the number of oscillation cycles taken to reach a synchronised state. The corresponding property can now be formalised in PCTL as

$$\mathsf{R}[\mathrm{F}\ \mathsf{synchronised}] \equiv \mathsf{R}\left[\mathrm{F}\left(\bigvee_{\Phi=1}^{T} k_\Phi = N\right)\right].$$

Recall from the definition of reachability reward properties in Section 2.3 that a reward of ∞ is accumulated along a path that never reaches a set of target states. Hence, this is the result that is obtained when checking the property in a model that does not synchronise.

The results of model checking two instantiations of the formal model are now presented. For a network of synchronising sensor nodes, there are several attributes of interest that

can be weighted against each other:

1. the probability of achieving synchrony,

2. the time taken to achieve synchrony,

3. the power consumption of the network.

Direct results for the first two desirable properties can be obtained by respectively checking the properties $\mathsf{P}[\mathsf{F}\,\mathsf{synchronised}]$ and $\mathsf{R}[\mathsf{F}\,\mathsf{synchronised}]$ in the formal model. Reasoning about the expected power consumption of a single node, or the network as a whole, requires more specialised reward functions, and is explored in detail in Chapter 5. In WSNs, communication is costly with respect to power consumption. Communication is either active when sending a message, for example when a node fires and broadcasts its firing signal, or passive, when receiving messages from other nodes. Hence, during periods where a sensor does neither, the antenna can be shut down to save energy. In the formal model, this interval of inactivity corresponds to the refractory period. That is, the longer the refractory period is, the less energy will be consumed.

4.5.1 Mirollo and Strogatz synchronisation model.

The formal model is first instantiated as a discretisation of the Mirollo and Strogatz [117] (M&S) model of synchronisation. The perturbation function is set to

$$\Delta(\Phi, \alpha, \epsilon) = \lfloor \Phi\alpha\epsilon \rceil \,.$$

Observe that the perturbation induced by the firing of another oscillator increases linearly with the phase of the perturbed oscillator. With this function fixed the models were checked with respect to the two properties of interest, for the parameter space defined by

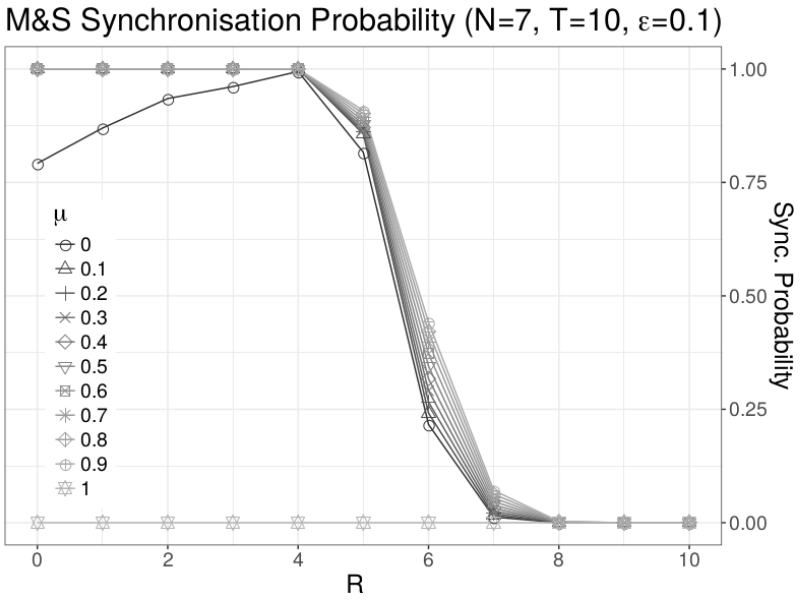


Figure 4.5: Mirollo and Strogatz synchronisation: synchronisation probabilities for different refractory periods.

considering all possible valuations from

$$N \in [3 \dots 7], T \in [4 \dots 10], R \in [0 \dots T], \epsilon \in \{0.1, 0.2, \dots 1\}, \mu \in \{0.1, 0.2, \dots, 1\}.$$

Observe that for each instance we induce a DTMC by providing a concrete value for the single parameter μ.

Figure 4.5 plots the probability of synchronisation for different rates of broadcast failure against the refractory period for $N = 7$, $T = 10$, and $\epsilon = 0.1$. We can extrapolate a trade-off between a high refractory period and high synchronisation probability. As long as the refractory period is less than half the oscillation cycle, synchronisation will be achieved in almost all cases. Higher values for R result in a rapid drop in the probability of synchronisation. The exceptions are the edge cases $\mu = 0$, and $\mu = 1$. Unsurprisingly,

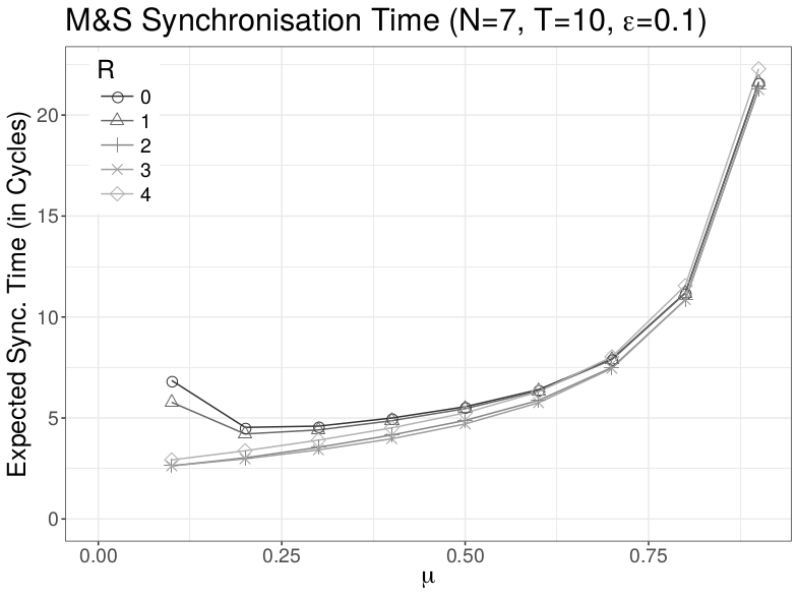


Figure 4.6: Mirollo and Strogatz synchronisation: synchronisation times for different rates of broadcast failure.

if all firings result in broadcast failures ($\mu = 1$), the probability of synchronisation is almost zero. In fact, the only paths that synchronise in this case are those where the first state along that path where the oscillators are configured (i.e. the successor state of s_0) is synchronised. The comparably bad synchronisation probabilities for $\mu = 0$ may seem surprising. If $\mu = 0$ then all transitions in the model are taken deterministically, excepting the initial transitions from s_0 to states where oscillators are configured. This can lead to unwanted cyclic behaviour, an artefact of the discreteness of the phase values, where very minor perturbations to phase are ignored due to the rounding of values to the nearest integer in the perturbation function, resulting in groups of oscillators staying unsynchronised forever. Similar phenomena have also been observed in other approaches used to model emergent synchronisation [56]. When some level of uncertainty is introduced to the model perpetually asynchronous cycles no longer occur.

Figure 4.6 plots the expected time taken for the model to synchronise. A higher refractory period results in less time being taken to achieve synchronisation when the probability for broadcast failure is low. In general, a longer refractory period up to half the cycle length improves the rate of convergence to synchrony, which is consistent with the findings of [42]. Furthermore, for high values of μ the differences in the times required to achieve synchronisation for different refractory period lengths are negligible. Hence, a refractory period of slightly less than half the cycle, with a low value for the coupling constant ϵ, is optimal for this model of synchronisation, if shorter synchronisation times are desirable. As ϵ is increased the results remain similar, but with a decrease in the time taken to achieve synchronisation.

Mirollo and Strogatz proved that a system of interacting pulse-coupled oscillators with no refractory period would always synchronise under the assumption of a convex perturbation function, and higher values for ϵ would results in faster synchronisation rates. The findings presented here are consistent with their results, however the effect of a refractory period on the likelihood of synchronisation was also investigated. The exception is the results obtained when the chance of broadcast failure is 0, or the coupling strength was set to very low values. This is because unlike the oscillators analysed in the work of Mirollo and Strogatz [115], oscillators in the formal model synchronise over discrete values, and very minor induced perturbations are ignored due to the rounding of phase values to the nearest discrete value.

4.5.2 Mean phase synchronisation model.

The formal model is now instantiated for a model of synchronisation similar to the work of Breza [27]. For this model of synchronisation, when the phase of an oscillator is perturbed by the firing of another oscillator it updates its phase to be the average of its current phase

and the phase of the firing oscillator, which is fixed as T in the formal model. The model as referred to as the *mean phase* (MP) synchronisation model.

Since the perturbation to phase is determined solely by the current phase of the firing oscillator, and the oscillator receiving the firing message, the is no notion of coupling strength between oscillators. Hence, in this model ϵ is ignored. However, the formal model can still be instantiated to formalise such a protocol. The perturbation function is instantiated as:

$$\Delta(\Phi, \alpha, \epsilon) = \left\lfloor 2^{-\alpha}(T(2^{\alpha} - 1) + \Phi) \right\rceil - \Phi. \tag{4.3}$$

Informally, the function calculates the result of iteratively taking the mean of the phase and T, for α iterations, and returns the difference between this and the original phase. Note that the perturbation induced by the firing of another oscillator is inversely proportional to 2^{Φ}.

Example 17. *Consider the following situation where all oscillators are synchronising over* $T = 10$ *discrete values, the perturbation function is instantiated as that in (4.3), and a group of oscillators sharing phase* 2 *are perturbed by the firing of a group of three oscillators where no broadcast failures occur. The perturbation induced by the firing would be given by*

$$\begin{aligned} \Delta(2, 3, \epsilon) &= \left\lfloor 2^{-3}(10(2^3 - 1) + 2) \right\rceil - 2 \\ &= \left\lfloor \frac{72}{8} \right\rceil - 2 \\ &= 7, \end{aligned}$$

which is what we would expect, since iteratively taking the average of the phase 2 *with* 10 *three times yields* 6*, then* 8*, and finally* 9*; the perturbation is therefore* $9 - 2 = 7$.

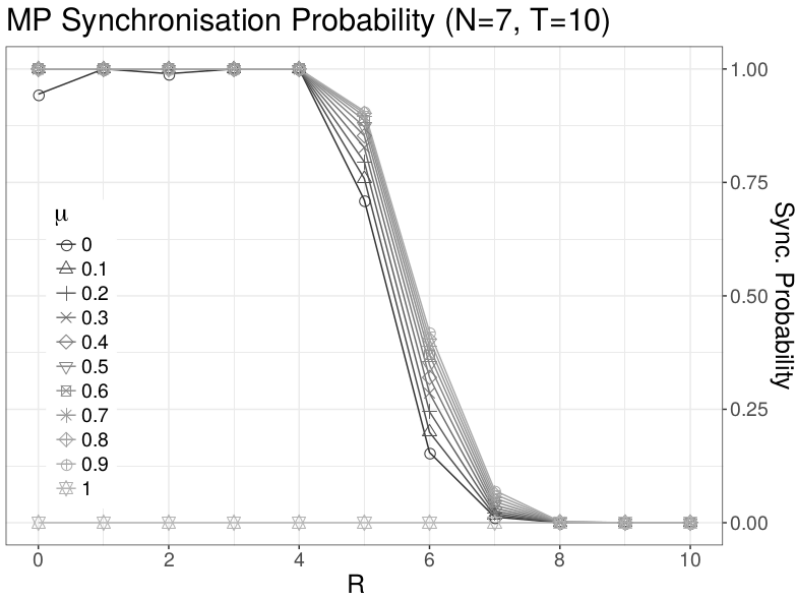


Figure 4.7: Mean phase synchronisation: synchronisation probabilities for different refractory periods.

Models were generated for the parameter values examined for the M&S model of synchronisation in Section 4.5.1, and again the models were analysed with respect to the two properties of interest.

Figure 4.7 shows the probability of achieving synchronisation for different rates of broadcast failure and lengths of refractory period. It has similar characteristics to Figure 4.5. For almost all values of μ the oscillators will always synchronise when the length of the refractory period is less than half the length of the oscillation cycle. Again as expected, $\mu = 1$ results in almost no synchronising runs, and $\mu = 0$ creates cyclic behaviour that leads to perpetual asynchrony. Here the MP synchronisation model is slightly more robust than the M&S synchronisation model with loosely coupled oscillators. However, if the coupling strength of the latter is increased, it performs even better. The results are consistent with the findings of [27], who showed that refractory periods longer than half of

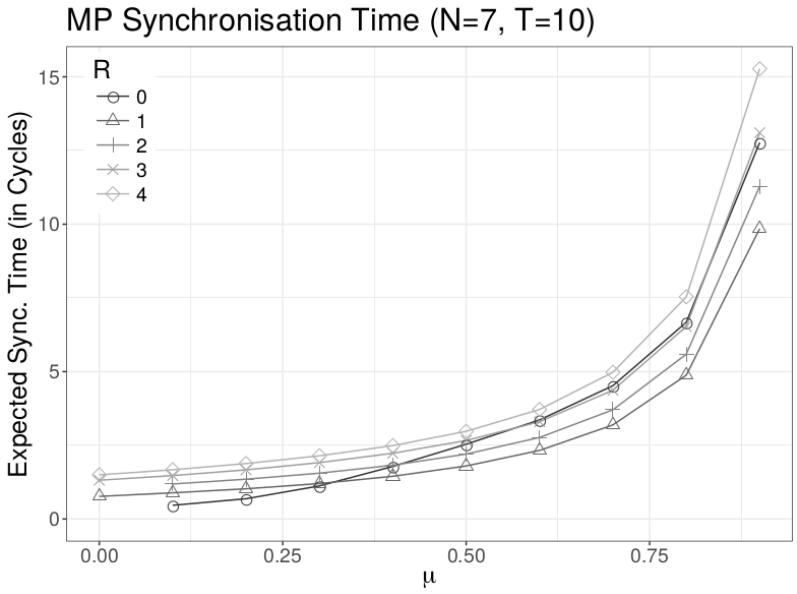


Figure 4.8: Mean phase synchronisation: synchronisation times for different rates of broadcast failure.

they cycle length result in systems where sychrony is often not achieved.

Figure 4.8 shows the results of checking the time taken to achieve synchronisation. In most cases, a short, but non-zero, refractory period results in shorter synchronisation times. In general, it seems optimal to choose a short, non-zero length refractory period, if shorter synchronisation times are desirable. When the probability of broadcast failure is low, however, there are negligible differences for refractory periods of different lengths. If robust communication is expected for a deployed network then a longer refractory period should be chosen, to conserve energy.

4.5.3 Summary of the two synchronisation methods.

Both the Mirollo and Strogatz and mean phase synchronisation models perform well for networks where the oscillators have refractory periods of length less than half the cycle

length. However, the mean phase model is more robust with respect to broadcast failure and also achieves shorter synchronisation times, making it a good choice for the synchronisation of networks where many messages are expected to be lost, perhaps due to adverse environmental effects, or for networks where synchronisation must be achieved rapidly.

4.5.4 Network synchronisation scalability.

Figures 4.9 and 4.10 plot the time taken to achieve synchronisation against the size of the network for different broadcast failure probabilities, for the Mirollo and Strogatz and mean phase synchronisation models respectively, where $T = 10$ and $R = 1$. For the M&S model when $\mu > 0.3$, increasing the number of nodes in the network results in shorter synchronisation times, while higher rates of broadcast failure yields longer synchronisation times. It is conjectured that the surprising peaks for $\mu \leqslant 0.3$ are again an artefact of rounding values to the nearest integer, resulting in cyclic behaviour, similar to that observed for the M&S model when $\mu \in \{0, 1\}$. For the mean phase synchronisation model, longer times to achieve synchronisation are again observed for higher rates of broadcast failure. Similarly, increasing the population size results in shorter times to achieve synchronisation. However, in this case the rate at which synchronisation time decreases, given an increase in the size of the population, is more pronounced. Unlike the M&S model there are no peaks in the graphs indicating undesirable asynchronous cyclic behaviour. For the M&S model, low coupling strength resulted in minor perturbations to phase being ignored due to rounding. In the mean phase model this does not occur, since the fractional part of the calculated mean phase is always greater than, or equal to, 0.5.

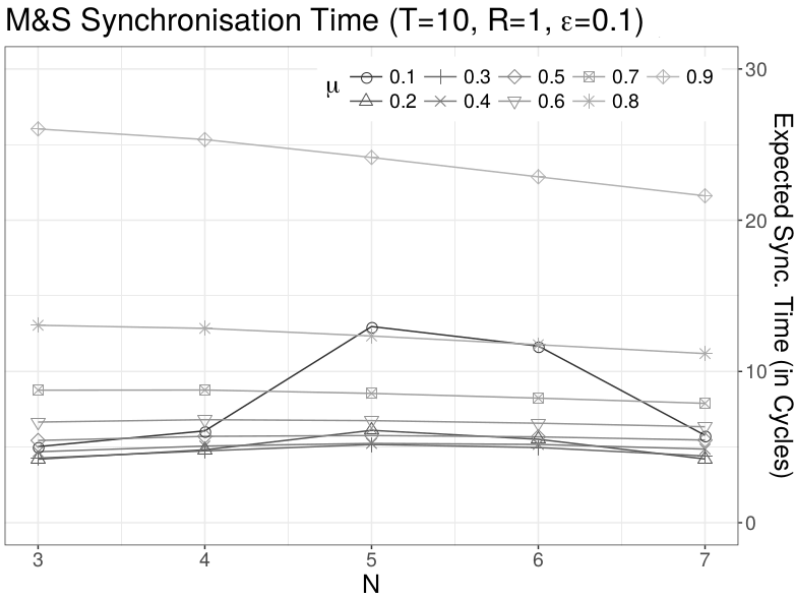


Figure 4.9: Synchronisation times for different number of oscillators for the Mirollo and Strogatz synchronisation model.

4.5.5 Model checking scalability.

Using formal models to analyse networks of indistinguishable oscillators is a promising approach. Network sizes of up to 7 oscillators were investigated, while other formal analyses in the literature turned out to be infeasible for more than four nodes for fully-connected networks [78]. The memory used, and time taken, by PRISM for exploring the reachable state space of a single model, and for checking the probabilistic reachability property in that model, are shown in Table 4.2. The increase in memory usage is as expected, and differences between the two models are relatively small. The properties can be checked in under a second. While this approach allows state space explosion to be postponed, we cannot escape it completely. The major bottleneck is not the model checking time itself, but rather state space exploration time. For individual models this was relatively short, but accumulated greatly over the parameter space that was investigated, where thousands

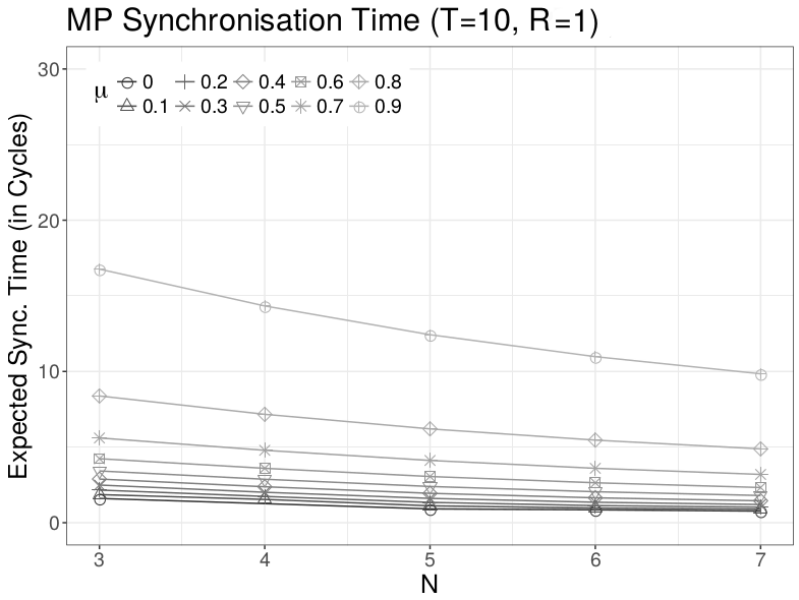


Figure 4.10: Synchronisation times for different number of oscillators for the mean phase synchronisation model.

of individual models were constructed.

4.6 Population model refinement

In Section 4.3 formal models were constructed for populations of up to 7 synchronising oscillators. Desirable properties could not be checked for larger populations, due to the limited resources of the machine used for their analysis. To facilitate the analysis of larger networks, a refinement of the population model is now presented. It is shown that the refinement results in a significant decrease in the size of the model, and is equivalent to the original model with respect to the reachability of global states where one or more oscillators are firing.

First, states where one or more oscillators are about to fire, and states where no oscilla-

Table 4.2: Memory for model checking and time for model construction, for $T = 10$, $R = 1$, $\epsilon = 0.1, \mu = 0.1$.

	M&S		MP	
N	Memory(MB)	Time(s)	Memory(MB)	Time(s)
3	124.63	0.09	131.30	0.09
4	161.33	0.37	162.42	0.43
5	262.62	1.65	261.39	1.61
6	592.94	5.28	610.20	5.42
7	1604.76	17.13	1495.59	16.88

tors will fire at all, are distinguished, and will be referred to as *firing states* and *non-firing* states, respectively.

Definition 22. *Given a population model $\mathcal{P} = (\Delta, N, T, R, \epsilon, \mu)$, a global state $\langle k_1, \ldots, k_T \rangle \in \Gamma$ is a* firing state *if, and only if, $k_T > 0$. The set of all firing states of $\mathcal{P}$ is denoted by Γ^{F}, and the set of all non-firing states of $\mathcal{P}$ is denoted by $\Gamma^{\mathsf{NF}} = \Gamma \setminus \Gamma^{\mathsf{F}}$.*

First, given a PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ let

$$|\mathbf{P}| = |\{(s, s') \mid s, s' \in \mathcal{S}^2 \text{ and } \mathbf{P}(s, s') \neq 0\}|,$$
$$|\mathcal{D}_{\mathcal{X}}| = |\mathcal{S}| + |\mathbf{P}|$$

denote the number of non-zero transitions in $\mathbf{P}$, and the total number of states and non-zero transitions in $\mathcal{D}_{\mathcal{X}}$, respectively. Observe that $|\mathbf{P}|$ is the number of edges in $\mathcal{G}_{\mathcal{D}_{\mathcal{X}}} = (\mathcal{S}, \mathcal{E})$, the underlying graph of $\mathcal{D}_{\mathcal{X}}$ and is not used to denote the determinant of the matrix $\mathbf{P}$.

Theorem 1. *For every population model $\mathcal{P} = (\Delta, N, T, R, \epsilon, \mu)$ and corresponding PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$, there is a reduced model $\mathcal{D}'_{\mathcal{X}} = (\mathcal{S}', s_0, \mathbf{P}', L', \mathcal{X})$ where $|\mathcal{D}'_{\mathcal{X}}| < |\mathcal{D}_{\mathcal{X}}|$ and unbounded-time reachability properties with respect to synchronised firing states in $\mathcal{D}_{\mathcal{X}}$ are preserved in $\mathcal{D}'_{\mathcal{X}}$ for all total well-defined evaluations for $\mathcal{X}$. In particular, the states*

and transitions in $\mathcal{D}_{\mathcal{X}}$ *are reduced in* $\mathcal{D}'_{\mathcal{X}}$ *such that* $\mathcal{S}' = \mathcal{S} \setminus \Gamma^{\mathsf{NF}}$ *and*

$$|\mathcal{S}'| = 1 + \frac{T^{(N-1)}}{(N-1)!},$$
$$|\mathbf{P}'| \leqslant |\mathbf{P}| - 2|\Gamma^{\mathsf{NF}}|$$

where $\cdot^{(n)}$ *denotes the rising factorial.*

The proof of the theorem follows, but first some preliminary properties of non-firing states, and their relation to firing states, are defined.

Lemma 3. *Every non-firing state* $s^{\mathsf{NF}} \in \Gamma^{\mathsf{NF}}$ *has exactly one successor state, and in that state all oscillator phases have increased by* 1*.*

Proof. Given a non-firing state $s^{\mathsf{NF}} = \langle k_1, \ldots, k_T \rangle$ observe that as $k_T = 0$ there is only one possible failure vector for s^{NF}, namely $\{\star\}^T$. The set of all successor states of s^{NF} is then the singleton

$$\{\vec{\mathsf{succ}}(s^{\mathsf{NF}}, \{\star\}^T)\}.$$

By construction we can then see that

$$\begin{aligned} \mathit{update}^{\Phi}(s^{\mathsf{NF}}, \{\star\}^T) &= \Phi + 1 && \text{for } 1 \leqslant \Phi \leqslant T \\ \mathcal{U}_{\Phi}(s^{\mathsf{NF}}, \{\star\}^T) &= \{\Phi - 1\} && \text{for } 1 < \Phi \leqslant T \\ \mathcal{U}_1(s^{\mathsf{NF}}, \{\star\}^T) &= \emptyset. \end{aligned}$$

The single successor state is then given by

$$\vec{\mathsf{succ}}(s^{\mathsf{NF}}, \{\star\}^T) = \langle 0, k_1, \ldots, k_T - 1 \rangle.$$

□

Corollary 1. *An immediate corollary of Lemma 3 is that a transition from any non-firing state is taken deterministically, since for any* $\sigma \in \Gamma^{\mathsf{NF}}$ *we have* $P_\tau(\sigma, \{\star\}^T) = 1$.

4.6.1 Reachable state reduction.

Given a path $s_0 \cdots s_{n-1} s_n$ where $s_i \in \Gamma^{\mathsf{NF}}$ for $0 < i < n$ and $s_0, s_n \in \Gamma^{\mathsf{F}}$, we omit transitions (s_i, s_{i+1}) for $0 \leqslant i < n$, and instead introduce a direct transition from s_0, the first firing state, to s_n, the next firing state in the sequence. For any $s = \langle k_1, \ldots, k_T \rangle \in \mathcal{S}$ let

$$\delta_s = \max\{\Phi \mid k_\Phi > 0 \text{ and } 1 \leqslant \Phi \leqslant T\}$$

be the highest phase of any oscillator in s. The successor state of a non-firing state is then the state where all phases have increased by $T - \delta_s$. Observe that $T - \delta_s = 0$ for any $s \in \Gamma^{\mathsf{F}}$.

Definition 23. *The* deterministic successor function $\overset{\curvearrowright}{\mathsf{succ}} : \Gamma \to \Gamma^{\mathsf{F}}$, *given by*

$$\overset{\curvearrowright}{\mathsf{succ}}(\langle k_1, \ldots, k_T \rangle) = \{0\}^{T-\delta_s \frown} \langle k_1, \ldots, k_{\delta_s} \rangle,$$

maps a non-firing, or firing, state s *to the next firing state reachable by taking* $T - \delta_s$ *deterministic transitions. Observe that* $\delta_{s^{\mathsf{F}}} = T$ *for any firing state* s^{F}, *and hence that* $\overset{\curvearrowright}{\mathsf{succ}}(s^{\mathsf{F}}) = s^{\mathsf{F}}$.

The definition for the set of all successor states for some global state $s \in \mathcal{S}$ is updated to incorporate the deterministic successor function.

Definition 24. *Given a global state* $s \in \mathcal{S}$ *let* $\overset{\curvearrowright}{\mathsf{Next}}(s)$ *be the set of all successor states of*

s, where

$$\overset{\curvearrowright}{\mathsf{Next}}(s) = \{\overset{\curvearrowright}{\mathsf{succ}}(\overrightarrow{\mathsf{succ}}(s, \mathbf{f})) \mid \mathbf{f} \in \mathcal{F}_s\}.$$

Definition 25. *Given a firing state $s^{\mathsf{F}} \in \Gamma^{\mathsf{F}}$ let $\mathsf{Pred}(s^{\mathsf{F}})$ be the set of all non-firing predecessors of s^{F}, where s^{F} is reachable from the predecessor by taking some positive number of transitions deterministically. Formally,*

$$\mathsf{Pred}(s^{\mathsf{F}}) = \{s \mid s \in \Gamma^{\mathsf{NF}} \text{ and } \overset{\curvearrowright}{\mathsf{succ}}(s) = s^{\mathsf{F}}\}.$$

All states $s \in \mathsf{Pred}(s^{\mathsf{F}})$ are referred to as deterministic predecessors *of s^{F}.*

Then given $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ with $\mathcal{S} = \{s_0\} \cup \Gamma$ let

$$\mathcal{S}' = \mathcal{S} \setminus \bigcup_{s \in \Gamma^{\mathsf{F}}} \mathsf{Pred}(s)$$

to be the reduction of $\mathcal{S}$ where all non-firing states from which a firing state can be reached deterministically are removed.

Lemma 4. *For any PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ with $\mathcal{S} = \Gamma \cup \{s_0\}$, the reduction $\mathcal{S}'$ is equal to $\Gamma^{\mathsf{F}} \cup \{s_0\}$.*

Proof. Let $P = \bigcup_{s \in \Gamma^{\mathsf{F}}} \mathsf{Pred}(s)$ be the set of all predecessors of firing states in Γ^{F}. Since $\mathcal{S} = \Gamma \cup \{s_0\}$ and $\mathcal{S}' = \mathcal{S} \setminus P$ we can see that $\mathcal{S}' = \Gamma^{\mathsf{F}} \cup \{s_0\}$ if, and only if, $P = \Gamma^{\mathsf{NF}}$. From Definition 25 it follows that $P \subseteq \Gamma^{\mathsf{NF}}$. In addition, for any $s \in \Gamma^{\mathsf{NF}}$ there is some state s' such that $s \in \mathsf{Pred}(s')$ and $s' = \overset{\curvearrowright}{\mathsf{succ}}(s) \in \Gamma^{\mathsf{F}}$, hence $\Gamma^{\mathsf{NF}} \subseteq P$ and the lemma is proved. □

Lemma 5. *For any PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ with $\mathcal{S} = \Gamma \cup \{s_0\}$, the number of states*

in the reduction of $\mathcal{S}$ is given by

$$|\mathcal{S}'| = 1 + \frac{T^{(N-1)}}{(N-1)!},$$

where $\cdot^{(n)}$ denotes the rising factorial.

Proof. Observe that there are $\binom{N+T-1}{N}$ ways to assign T distinguishable phase values to N indistinguishable oscillators [58]. Since $\mathcal{S} = \Gamma \cup \{s_0\}$, and Γ is the set of all possible configurations for oscillators, it is clear that

$$|\mathcal{S}| = \binom{N+T-1}{N} + 1.$$

For any non-firing state $s^{\mathsf{NF}} = \langle k_1, \ldots, k_T \rangle \in \Gamma^{\mathsf{NF}}$ it is known from Definition 14 that

$$\sum_{\Phi=1}^{T} k_\Phi = N$$

and from Definition 22 that $k_T = 0$, so it must be the case that

$$\sum_{\Phi=1}^{T-1} k_\Phi = N.$$

That is, there must be

$$\binom{N+T-2}{N}$$

ways to assign $T-1$ distinguishable phases to N indistinguishable oscillators, and so

$$|\Gamma^{\mathsf{NF}}| = \binom{N+T-2}{N}.$$

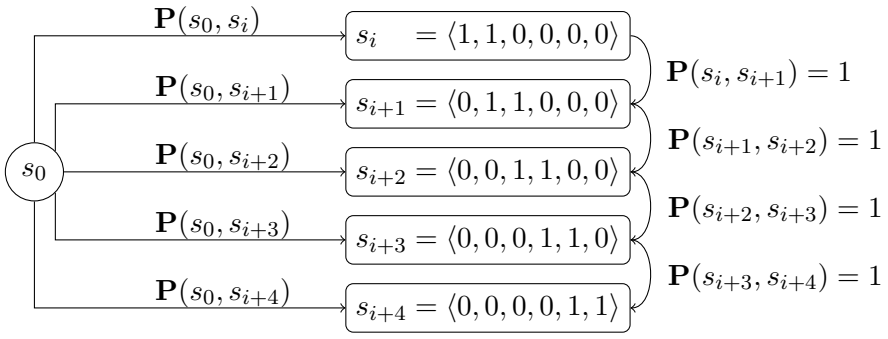


Figure 4.11: Five possible initial configurations in $\mathcal{S}$ for $N = 2$, $T = 6$.

Since $\mathcal{S}' = \mathcal{S} \setminus \Gamma^{\mathsf{NF}}$ from Lemma 4 it must be the case that

$$
\begin{aligned}
|\mathcal{S}'| &= |\mathcal{S}| - |\Gamma^{\mathsf{NF}}| &&= 1 + \binom{N+T-1}{N} - \binom{N+T-2}{N} \\
&&&= 1 + \frac{(N+T-1)!}{N!(T-1)!} - \frac{(N+T-2)!}{N!(T-2)!} \\
&&&= 1 + \frac{(N+T-2)!}{(N-1)!(T-1)!} \\
&&&= 1 + \frac{T^{(N-1)}}{(N-1)!}.
\end{aligned}
$$

□

4.6.2 Transition matrix reduction.

The reduction in the number of non-zero transitions in the model is now described. First, the removal of transitions from the initial state to non-firing states is illustrated using a simple example, and secondly, the removal of transitions from firing states to any successor non-firing states is shown.

Figure 4.11 shows five possible initial configurations $s_i, \ldots, s_{i+4} \in \mathcal{S}$ for $N = 2$ oscillators with $T = 6$ values for phase, where a transition is taken from s_0 to each s_j with probability $\mathbf{P}(s_0, s_j)$. Any infinite run of $\mathcal{D}_\mathcal{X}$ where a transition is taken from s_0 to one of the configured states $s_i, \ldots, s_{i+3}$ will pass through s_{i+4}, since all transitions (s_{i+j}, s_{i+j+1}) for $0 \leqslant j \leqslant 3$ are taken deterministically. Also, observe that states $s_i, \ldots, s_{i+3}$ are not in $\mathcal{S}'$, since s_{i+4} is reachable from each by taking some number of deterministic transitions. The probability of moving from s_0 to s_{i+4} in $\mathbf{P}'$ is therefore set to be the sum of the probabilities of moving from s_0 to s_{i+4} and from s_0 to each of its predecessors in $\mathbf{P}$. Generally, given a state $s \in \mathcal{S}'$ where $s \neq s_0$ the probability to take the transition from the initial state to s is set as

$$\mathbf{P}'(s_0, s) = \mathbf{P}(s_0, s) + \sum_{s' \in \mathsf{Pred}(s)} \mathbf{P}(s_0, s').$$

Now the probability with which a transition is taken from a firing state to each of its possible successors is calculated. For each firing state $s^\mathsf{F} \in \mathcal{S}'$ and each possible successor $s \in \overset{\curvearrowright}{\mathsf{Next}}(s^\mathsf{F})$ of s^F, let $\mathcal{F}_{s^\mathsf{F} \to s}$ be the set of all possible failure vectors for s^F for which the successor of s^F is s,

$$\mathcal{F}_{s^\mathsf{F} \to s} = \{\mathbf{f} \in \mathcal{F}_{s^\mathsf{F}} \mid \overset{\curvearrowright}{\mathsf{succ}}(\overrightarrow{\mathsf{succ}}(s^\mathsf{F}, \mathbf{f})) = s\}.$$

The probability with which a transition from s^F to s is taken is then set to

$$\mathbf{P}'(s^\mathsf{F}, s) = \sum_{\mathbf{f} \in \mathcal{F}_{s^\mathsf{F} \to s}} P_\tau(s^\mathsf{F}, \mathbf{f}).$$

Lemma 6. *For a PMC $\mathcal{D}_\mathcal{X} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ with $\mathcal{S} = \{s_0\} \cup \Gamma$, and its reduction*

$\mathcal{D}'_{\mathcal{X}} = (\mathcal{S}', s_0, \mathbf{P}', L', \mathcal{X})$, *the transitions in* $\mathbf{P}$ *are reduced in* $\mathbf{P}'$ *such that*

$$|\mathbf{P}'| \leqslant |\mathbf{P}| - 2|\Gamma^{\mathsf{NF}}|.$$

Proof. From Lemma 4 $|\mathcal{S}'| = |\mathcal{S} \setminus \Gamma^{\mathsf{NF}}|$, and hence $|\Gamma^{\mathsf{NF}}|$ transitions from s_0 to non-firing states are not in $\mathbf{P}'$, and from Lemma 3 there is one transition from each non-firing state to its unique successor state that is not in $\mathbf{P}'$. Since no additional transitions are introduced in the reduction it is clear that $|\mathbf{P}'| \leqslant |\mathbf{P}| - 2|\Gamma^{\mathsf{NF}}|$. □

4.6.3 Preservation of reachability properties

Lemma 7. *For every PMC* $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$, *unbounded-time reachability properties with respect to synchronised firing states in* $\mathcal{D}_{\mathcal{X}}$ *are preserved in its reduction* $\mathcal{D}'_{\mathcal{X}}$ *for all total well-defined evaluations for* $\mathcal{X}$.

Proof. This can be proved by showing that for every $\bowtie \in \{<, \leqslant, \geqslant, >\}$ and every $\lambda \in [0, 1]$, if $s_0 \models \mathsf{P}_{\bowtie\lambda}[\mathsf{F}\ \mathsf{synch}]$ holds in $\mathcal{D}_{\mathcal{X}}$ then it also holds in $\mathcal{D}'_{\mathcal{X}}$. Assume that we have υ, some total well-defined evaluation for $\mathcal{X}$, yielding the DTMCs

$$\mathcal{D}_{\mathcal{X}_\upsilon} = (\mathcal{S}, s_0, \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon], L, \mathcal{X}_\upsilon),$$
$$\mathcal{D}'_{\mathcal{X}_\upsilon} = (\mathcal{S}', s_0, \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon]', L', \mathcal{X}_\upsilon).$$

From the semantics of PCTL over a DTMC we have

$$s_0 \models \mathsf{P}_{\bowtie\lambda}[\mathsf{F}\ \ \mathsf{synch}] \quad \Leftrightarrow \quad \mathsf{Pr}\{\omega \in \mathsf{Paths} \mid \omega \models \mathrm{F}\ \ \mathsf{synch}\} \bowtie \lambda,$$

and need to show that

$$\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega \in \mathsf{Paths}^{\mathcal{D}_{\mathcal{X}_v}} \mid \omega \models \mathrm{F}\ \ \mathsf{synch}\} = \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega' \in \mathsf{Paths}^{\mathcal{D}'_{\mathcal{X}_v}} \mid \omega' \models \mathrm{F}\ \ \mathsf{synch}\},$$

where $\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}$ and $\mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}$ denote the probability measures with respect to the sets of infinite paths from s_0 in $\mathcal{D}_{\mathcal{X}_v}$ and $\mathcal{D}'_{\mathcal{X}_v}$ respectively.

Given a firing state $s^{\mathsf{F}} \in \mathcal{S}$ let $\mathsf{Paths}_{s^{\mathsf{F}}}^{\mathcal{D}_{\mathcal{X}_v}}$ denote the set of all infinite paths of $\mathcal{D}_{\mathcal{X}_v}$ starting in s_0 where the first firing state reached along that path is s^{F}. All such sets for all firing states in $\mathcal{S}$ form a partition, such that

$$\bigcup_{s^{\mathsf{F}} \in \Gamma^{\mathsf{F}}} \mathsf{Paths}_{s^{\mathsf{F}}}^{\mathcal{D}_{\mathcal{X}_v}} = \mathsf{Paths}^{\mathcal{D}_{\mathcal{X}_v}}.$$

Now observe that any infinite path of $\mathcal{D}_{\mathcal{X}_v}$ can be written in the form

$$\omega = s_0 \omega_1^{\mathsf{NF}} s_1^{\mathsf{F}} \omega_2^{\mathsf{NF}} s_2^{\mathsf{F}} \cdots$$

where s_i^{F} is the i^{th} firing state in the path and each

$$\omega_i^{\mathsf{NF}} = s_i^1 s_i^2 \cdots s_i^{k_i}$$

is a possibly empty sequence of k_i non-firing states. Then for every such path in $\mathcal{D}_{\mathcal{X}_v}$ there is a corresponding path of $\mathcal{D}'_{\mathcal{X}_v}$ without non-firing states, and of the form

$$\omega' = s_0 s_1^{\mathsf{F}} s_2^{\mathsf{F}} s_3^{\mathsf{F}} \cdots,$$

since $s_i^j \in \mathsf{Pred}(s_i^{\mathsf{F}})$ for all $i \geqslant 1$ and all $1 \leqslant j \leqslant k^i$. As only deterministic transitions have

been removed in $\mathcal{D}'_{\mathcal{X}_v}$ observe that

$$\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{s_1^{\mathsf{F}}\omega_2^{\mathsf{NF}}s_2^{\mathsf{F}}\cdots\} = \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{s_1^{\mathsf{F}}s_2^{\mathsf{F}}s_3^{\mathsf{F}}\cdots\}.$$

Hence, only the finite paths from s_0 to s_1^{F} need to be considered. To that end, observe that there are $|\mathsf{Pred}(s_1^{\mathsf{F}})|$ possible prefixes for each path from s_0 to s_1^{F} where the initial transition is taken from s_0 to some non-firing predecessor of s_1^{F}, plus the single prefix where the initial transition is taken to s_1^{F} itself. Overall there are exactly $|\mathsf{Pred}(s_1^{\mathsf{F}})|+1$ distinct finite prefixes that have ω' as their corresponding path in $\mathcal{D}'_{\mathcal{X}_v}$. The set of these prefixes for a path ω' of $\mathcal{D}'_{\mathcal{X}_v}$ is denoted by $\mathsf{Pref}(\omega')$. Since the measure of each finite prefix extends to a measure over the set of infinite paths sharing that prefix, it is sufficient to show that the sum of the probabilities for these finite prefixes is equal to the probability of the unique prefix s_0, s_1^{F} of ω', that is

$$\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\mathsf{Pref}(\omega') = \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{s_0, s_1^{\mathsf{F}}\}.$$

This can be written as

$$\begin{aligned}\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\mathsf{Pref}(\omega') &= \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon](s_0, s_1^{\mathsf{F}}) + \sum_{s\in\mathsf{Pred}(s_1^{\mathsf{F}})} \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon](s_0, s)\cdot 1^{k_s} \\ &= \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon](s_0, s_1^{\mathsf{F}}) + \sum_{s\in\mathsf{Pred}(s_1^{\mathsf{F}})} \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon](s_0, s),\end{aligned}$$

where k_s is the number of deterministic transitions that lead from s to s_1^{F} in $\mathcal{D}_{\mathcal{X}_v}$. Now recall that for any $s \in \mathcal{S}' \setminus \{s_0\}$,

$$\mathbf{P}'(s_0, s) = \mathbf{P}(s_0, s) + \sum_{s'\in\mathsf{Pred}(s)} \mathbf{P}(s_0, s).$$

Table 4.3: Reduction in state space and transitions for different model instances.

		$\mathcal{D}_{\mathcal{X}}$		$\mathcal{D}'_{\mathcal{X}}$		Reduction (%)	
N	T	States	Transitions	States	Transitions	States	Transitions
3	6	113	188	22	52	80.5	72.3
5	6	505	1030	127	389	74.9	62.2
8	6	2575	7001	793	3154	69.2	54.9
3	8	241	410	37	97	84.6	76.3
5	8	1585	3250	331	1097	79.1	66.2
8	8	12871	34615	3433	14519	73.3	58.1
3	10	441	752	56	156	87.3	79.3
5	10	4005	8114	716	2484	82.1	69.4
8	10	48621	128936	11441	50883	76.5	60.5

It has been shown that $\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\mathsf{Pref}(\omega') = \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{s_0, s_1^{\mathsf{F}}\}$ and the lemma is proved. □

4.6.4 Proof and empirical analysis

Proof of Theorem 1. Follows from Lemmas 5 and 6 for the reduction of states and transitions respectively, and from Lemma 7 for the preservation of unbounded time reachability properties. □

Table 4.3 shows the number of reachable states and transitions of the PMC, and corresponding reduction, for different population sizes (N) and oscillation cycle lengths (T), using the Mirollo and Strogatz model of synchronisation discussed in Chapter 4.5.1. The number of reachable states is stable under changes to the parameters R, ϵ, and μ, since every possible firing state is always reachable from the initial state. For the results shown here the parameters were arbitrarily set to $R = 1$, $\epsilon = 0.1$. The underlying graph of the model, and hence the number of transitions, is stable under changes to the parameter μ, and is not of interest here.

Table 4.4 shows the number of transitions of the PMC, and corresponding reduction, for various population model instances, and again uses the Mirollo and Strogatz model of

Table 4.4: Reduction in transitions for different population model instances.

				Transitions		
N	T	R	ϵ	$\mathcal{D}_{\mathcal{X}}$	$\mathcal{D}'_{\mathcal{X}}$	Reduction (%)
5	10	1	0.1	8114	2484	69.4
5	10	3	0.1	7928	2391	69.8
5	10	5	0.1	7568	2211	70.8
5	10	7	0.1	6976	1915	72.5
5	10	9	0.1	6006	1430	76.2
5	10	1	0.01	6006	1430	76.2
5	10	1	0.05	6426	1640	74.5
5	10	1	0.1	8114	2484	69.4
5	10	1	0.25	8950	2902	67.6
5	10	1	0.5	9382	3118	66.7

synchronisation. Increasing the length of the refractory period (R) results in an increase in the reduction of transitions in the model. A longer refractory period leads to more firing states where the firing of a group of oscillators is ignored. This results in successor states having oscillators with lower values for phase, and hence a longer sequence of deterministic transitions (later removed in the reduction) leading to the next firing state. Conversely, increasing the strength of the coupling between oscillators (ϵ) results in a decrease in the reduction of transitions in the model. For the Mirollo and Strogatz model of synchronisation used here, increasing the coupling strength results in a linear increase in the perturbation to phase induced by the firing of an oscillator. This results in successor states of firing states having oscillators with higher values for phase, and hence a shorter sequence of deterministic transitions leading to the next firing state.

4.7 Conclusion

A formal, parametric model for networks of pulse-coupled oscillators was presented, where oscillation cycles are defined as sequences of discrete states. The model was instantiated

for two models of synchronisation used for the coordination of wireless sensor networks or robotic swarm systems. The parameter space was explored for each synchronisation model. For each parameter valuation a model was automatically generated, encoded as a PMC. The parametric influence on both the rate at which synchronisation occurs, and the time taken for it to occur, were investigated, and trade-offs for parameter choices to minimise the energy consumption of a network were discussed.

The formal model was then further refined. The state-space explosion typically encountered when model-checking was mitigated by collapsing sequences of transitions where there are no interactions between oscillators and by removing all states of the model where no oscillators fire. This resulted in a model that was equivalent to the original with respect to the reachability of global firing states, but where the state space and number of non-zero transitions of the model were reduced, allowing larger networks of oscillators to be analysed.

A population model is appropriate when nodes in the network are indistinguishable with respect to their behaviour, and when the network is fully coupled. To analyse different network topologies, for instance a network of fully connected subcomponents, each subcomponent could be encoded as a single population model, and the product of all such subcomponent models could then be analysed. While such an approach sounds promising, encoding interactions between subcomponents would need to be defined, and further refinements to the model would clearly be necessary to offset the inevitable explosion in the size of the explorable state space.

As discussed in Section 4.5, the total time taken to explore the state space for a single parameter instantiation, and the total time taken to check desirable properties in those models, accumulates greatly when exploring a large parameter space. For some parametric models, small changes to a model parameter may result in relatively minor changes in the

underlying graph – for example, the parameter R of the population model presented here, where small changes to R result in the introduction, or removal, of a relatively small number of transitions in the model. If this is the case, results from the analysis of the original model can be used when analysing the model induced by the change in the parameter. This is described in more detail in Chapter 6. Exploring the parameter space for model parameters that do not induce structural changes can also be time consuming if the parameter space is very large. Chapter 7 describes new techniques that allow such parameter spaces to be explored more efficiently, in terms of resources used and time taken.

In the next chapter the binary notion of synchronisation introduced in this chapter is extended to facilitate reasoning about different degrees of synchronisation for global states of the model. It is then shown that by considering a subset of the state space of the formal model presented in this chapter, properties of interest relating to the resynchronisation of a destabilised network can be checked for much larger networks. Finally, new reward functions are defined for the reduced model that allow formal reasoning about the expected power consumption of a synchronising network (informally discussed in this chapter).

Chapter 5

Power Consumption in Networks of Pulse-Coupled Oscillators

Nature-inspired synchronisation protocols have been widely adopted to achieve consensus within wireless sensor networks. In this chapter, an analysis of the the power consumption of such protocols is conducted. In particular, the energy required to synchronise all nodes across a network. The model of nature-inspired, pulse-coupled oscillators is used to achieve network-wide synchronisation, and the formal model introduced in the previous chapter is annotated with rewards for recording energy usage. An exhaustive analysis is then carried out to calculate the resources consumed on each possible system execution. A broad range of parameter instantiations are investigated, along with the trade-offs between power consumption and time to synchronise. The results provides a principled basis for the formal analysis of a broader range of large-scale network protocols.

Minimising power consumption is a critical design consideration for WSNs [129, 2]. Once deployed a WSN is generally expected to function independently for long periods of time. In particular, regular battery replacement can be costly and impractical for

remote sensing applications. Hence, it is important to reduce the power consumption of the individual nodes by choosing low-power hardware and/or energy efficient protocols. However, to make informed choices, it is also necessary to have good estimations of the power consumption for individual nodes. While the general power consumption of the hardware can be extracted from data sheets, estimating the overall power consumption of different protocols is more demanding.

Surveys conducted by Irani and Pruhs [87] and Albers [2] investigated algorithmic problems in power management, in particular power-down mechanisms at the system and device level. Soua and Minet provided a general taxonomy for the analysis of wireless network protocols with respect to energy efficiency [136] by identifying the contributing factors of energy wastage, for example packet collisions and unnecessary idling. More specifically, Oller et al. analysed whether *wake-up radio* based medium-access control protocols are more energy efficient than *duty-cycle* based protocols [119]. Detrimental effects such as packet collisions can be overcome by allocating time slots for node communication. That is, nodes in a network need to synchronise their clocks and use time slots for communication to avoid packet collisions [128, 149].

The biologically inspired synchronisation protocols discussed in the previous chapter are well-suited for WSNs, since centralised control is not required to achieve synchrony. The protocols build on the underlying mathematical model of pulse-coupled oscillators[113, 117, 123]; integrate-and-fire oscillators with pulsatile coupling, such that when an oscillator fires it induces some phase-shift response determined by a *phase response function*. Over time mutual interactions can lead to all oscillators firing synchronously.

The population model presented in the previous chapter is extended with rewards to associate different current draws with its states, thus enabling us to measure the energy consumption of the overall network. An exhaustive analysis is then conducted using the

probabilistic model checker PRISM, to investigate the average and worst-case energy consumption for both the synchronisation of arbitrarily configured networks, and the restabilisation of a network, where a subset of oscillators desynchronised.

Exact time synchronisation, where all clocks always agree on their value, is never achieved for real-world deployments of synchronising devices [15]. Hardware imperfections result in different clock frequencies, environmental factors influence radio transmission, and network congestion leads to package collisions and loss [85]. Consequently, the precision of synchronisation is not usually required to be exact, and it is often sufficient for all oscillators to fire within some defined time window [15]. The size of this window depends on the application. Some applications may require a very small window, for instance distributed sensing of mobile objects, while others may prefer energy efficiency at the cost of synchronisation precision [128]. To this end we extend the binary notion of synchronisation discussed in the previous chapter, and derive a new synchronisation metric from the complex order parameter of Kuramoto [98], that captures the degree of synchrony of a fully connected network of oscillators as a real value in the interval $[0, 1]$.

The structure of the chapter is as follows. In Section. 5.1 related work is discussed. Section 5.2 introduces the derived synchronisation metric. In Section 5.3 the reduced formal population model of Chapter 4 is annotated with rewards corresponding to synchronisation time and power consumption. Subsequently, in Section 5.4 it is shown how rewards for the unreduced model can be translated into rewards for the reduced model. In Section 5.5 it is shown how for some properties, larger networks of nodes can be analysed by considering only a subset of the state space of the model. Then in Section 5.6 the results for certain parameter instantiations are evaluated, and their trade-offs are discussed. with respect to power consumption. Section 5.7 concludes the chapter.

5.1 Related work

Heidarian et al. used model checking to analyse clock synchronisation for medium access protocols [78]. They considered both fully-connected networks and line topologies with up to four nodes. Model checking of biologically inspired coupled oscillators has also been investigated by Bartocci et al. [6]. They present a subclass of timed automata suitable to model biological oscillators, and a model checking algorithm. However, their analysis was restricted to a network of three oscillators.

Wang et al. proposed an energy-efficient strategy for the synchronisation of pulse coupled oscillators [147]. In contrast to this work, they consider real-valued clocks and *delay-advance phase response* functions, where both positive and negative phase shifts can occur. That is, if an oscillator is perturbed within the first half of its oscillation cycle the perturbation is negative, while in the second half of the cycle the perturbation is positive. A result of their choice of phase response function is that synchronisation time is independent of the length of the refractory period, in contrast to the discrete model presented in the previous chapter. Furthermore, they assume that the initial phase difference between oscillators has an upper bound. They achieve synchrony for refractory periods larger than half the cycle, while the models presented here do not always synchronise in these cases, as a bound is not imposed on the phase difference of the oscillators. All possible differences in phase are considered, since we examine the energy consumption for the resynchronisation of a subset of oscillators.

Konishi and Kokame conducted an analysis of pulse coupled oscillators where a perceived pulse immediately resets oscillators to the start of their cycle [95]. Their goal was to maximise refractory period length, while still achieving synchronisation within some number of clock cycles. Similarly to this work, they restricted their analysis to a fully coupled network. They assumed that the synchronisation protocol was implemented as part

of the physical layer of the network stack by using capacitors to generate pulses, therefore their clocks were continuous and had different frequencies. Here the assumption is that the synchronisation protocol resides on a higher layer, where the clock values are discretised and oscillate with the same frequency.

5.2 Synchronisation metric

As discussed in the introduction to this chapter, exact time synchronisation where all clocks always agree on their value, is often neither achieved, nor required, for real-world deployments of synchronising devices, and instead it is often sufficient for all the oscillators in a network to fire within some defined window of time. In this section a new metric for sychronisation is defined; an extension of the work of Kuramoto [98]. The appropriateness of the metric is later verified by the analysis conducted in Section 5.6.1.

In the previous chapter a binary metric of synchrony for a population model was introduced. Given a population model $\mathcal{P} = (\Delta, N, T, R, \epsilon, \mu)$, a PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ was constructed, and a global state $\sigma = \langle k_1, \ldots, k_T \rangle \in \Gamma$ was labelled with

$$\mathsf{synchronised} \equiv \bigvee_{\Phi=1}^{T} k_\Phi = N$$

if all oscillators in that state shared the same phase. Here, a state is either synchronised, or not synchronised. However, it is clear that some global states appear to be closer to achieving a truly synchronised state than others. Consider the global states $\sigma_1 = \langle 0, 2, 0, 2, 0, 2 \rangle$ and $\sigma_2 = \langle 0, 0, 0, 0, 1, 5 \rangle$ of some population model for a network of $N = 6$ nodes with an oscillation cycle over $T = 6$ discrete values. Using the binary notion of synchrony all that is known is that both states are not synchronised (i.e. neither state is $\langle 0, 0, 0, 0, 0, 6 \rangle$). However, it is clear that for nearly all models of synchronisation encoded

as some perturbation function, σ_2 appears to be closer to converging to a state where all oscillators share the same phase, since five oscillators already share the same phase.

The binary notion of synchrony can be extended by introducing a *phase coherence* metric for the level of synchrony of a global state. The metric is derived from the *order parameter* introduced by Kuramoto [98] as a measure of synchrony for a population of coupled oscillators. We represent the phases of the oscillators as vectors on the unit circle in the complex plane.

Definition 26. *The function* $\phi^{\mathbb{C}} : [1 \dots T] \to \mathbb{C}$ *maps a phase value to a corresponding vector on the unit circle in the complex plane, and is defined as* $\phi^{\mathbb{C}}(\Phi) = e^{i\theta_\Phi}$*, where* $\theta_\Phi = \frac{2\pi}{T}(\Phi - 1)$*.*

Example 18. *Given the global state* $\langle 1, 0, 1, 0, 1, 0, 1, 0 \rangle$ *of a population model where* $N = 4$ *and* $T = 8$*, Figure 5.1 shows the corresponding vectors on the unit circle in the complex plane. Phase vectors are calculated for oscillator phases* 1*,* 3*,* 5*, and* 7 *as*

$$\begin{aligned}
\theta_1 &= \frac{2\pi}{8}(1-1) = 0 & \phi^{\mathbb{C}}(1) &= e^{i\theta_1} = e^{i\cdot 0} = 1,\\
\theta_3 &= \frac{2\pi}{8}(3-1) = \frac{\pi}{2}, & \phi^{\mathbb{C}}(3) &= e^{i\theta_3} = e^{\frac{i\pi}{2}} = i,\\
\theta_5 &= \frac{2\pi}{8}(5-1) = \pi, & \phi^{\mathbb{C}}(5) &= e^{i\theta_5} = e^{i\pi} = -1,\\
\theta_7 &= \frac{2\pi}{8}(7-1) = \frac{3\pi}{2}, & \phi^{\mathbb{C}}(7) &= e^{i\theta_7} = e^{\frac{3\pi}{2}} = -i.
\end{aligned}$$

A measure of synchrony $\eta \in [0, 1]$ can be obtained by calculating the magnitude of the complex number corresponding to the mean of the phase vectors. A global state has a maximal value of $\eta = 1$ when all oscillators are synchronised and share the same phase Φ, mapped to the vector defined by $\phi^{\mathbb{C}}(\Phi)$. It then follows that the mean vector is also $\phi^{\mathbb{C}}(\Phi)$

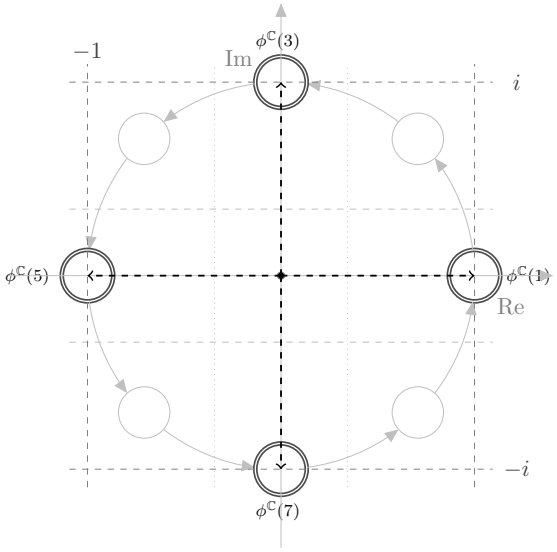


Figure 5.1: Argand diagram of the phase vectors for the global state $\langle 1,0,1,0,1,0,1,0\rangle$.

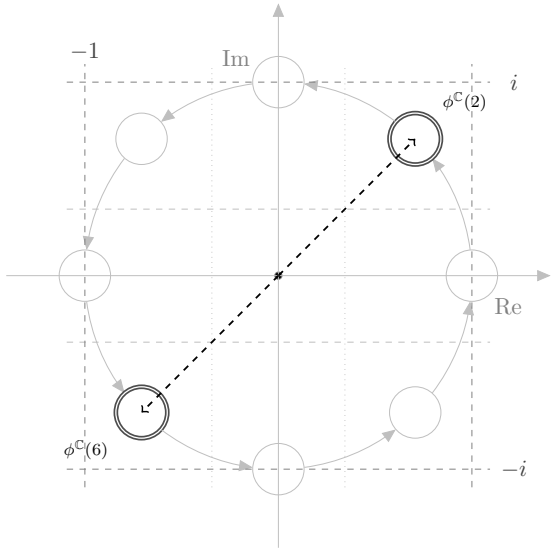


Figure 5.2: Mutual counterpoise achieved for the global state $\langle 0,1,0,0,0,1,0,0\rangle$.

and $|\phi^{\mathbb{C}}(\Phi)| = 1$. A global state has $\eta = 0$ when the mean of all phase vectors is the zero vector. This occurs when the phase vectors achieve mutual counterpoise, for example when $\frac{N}{2}$ oscillators share some phase value Φ and the remaining $\frac{N}{2}$ oscillators have a phase value whose corresponding vector on the complex plane is the negation of the complex conjugate of $\phi^{\mathbb{C}}(\Phi)$.

Example 19. *Given the global state* $\langle 0,1,0,0,0,1,0,0\rangle$ *of a population model where* $N = 2$ *and* $T = 8$*, Figure 5.2 shows the corresponding positions on the unit circle in the complex plane for phases* 2 *and* 5*. Here mutual counterpoise is achieved, as* $\frac{N}{2} = 1$ *oscillator has phase value* 2*, the remaining* $\frac{N}{2} = 1$ *oscillator has phase value* 5 *and* $\phi^{\mathbb{C}}(2) = e^{\frac{i\pi}{4}} = \frac{1+i}{\sqrt{2}}$ *is the complex conjugate of* $\phi^{\mathbb{C}}(5) = e^{\frac{i5\pi}{4}} = -\frac{1+i}{\sqrt{2}}$.

Definition 27. *The* phase coherence function $\mathsf{PCF} : \Gamma \to [0,1]$ *assigns a measure of*

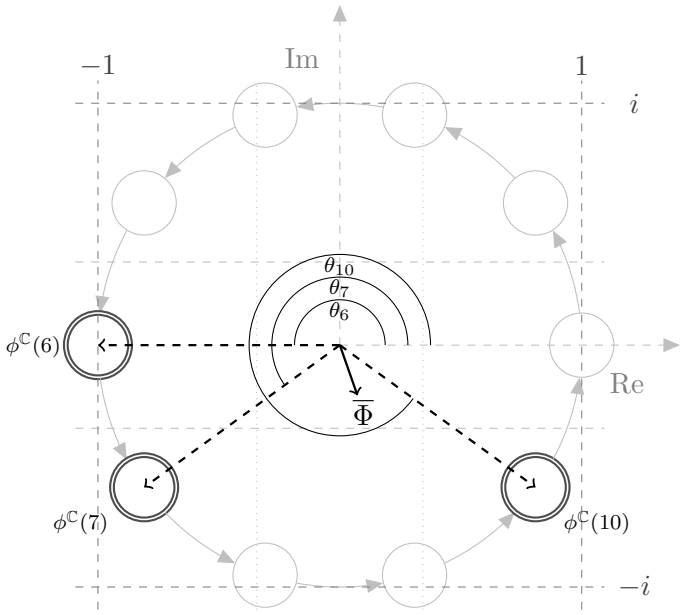


Figure 5.3: Argand diagram of the phase vector for the global state $\langle 0,0,0,0,0,2,1,0,0,5\rangle$, and mean phase vector $\overline{\Phi}$.

synchrony to each global state, and is defined as

$$\mathsf{PCF}(\langle k_1,\ldots,k_T\rangle) = \left|\frac{1}{N}\sum_{\Phi=1}^{T} k_\Phi \phi^{\mathbb{C}}(\Phi)\right|.$$

Example 20. *Figure 5.3 shows a plot on the complex plane of the phase vectors for some global state*

$$\sigma = \langle 0,0,0,0,0,2,1,0,0,5\rangle$$

of a population model where $N = 8$, $T = 10$ *The phase vectors are given by* $\phi^{\mathbb{C}}(6) = e^{i\pi}$ *for* 2 *oscillators with phase* 6, $\phi^{\mathbb{C}}(7) = e^{\frac{6i\pi}{5}}$ *for* 1 *oscillator with phase* 7, *and* $\phi^{\mathbb{C}}(10) = e^{\frac{9i\pi}{5}}$

for 5 *oscillators with phase* 10*. The phase coherence can then be determined as*

$$\mathsf{PCF}(\sigma) = \left|\frac{1}{8}(2e^{i\pi} + e^{\frac{6i\pi}{5}} + 5e^{\frac{9i\pi}{5}})\right| = 0.4671.$$

The mean phase vector is indicated on the diagram by $\overline{\Phi}$*.*

Later in Section 5.6 this metric is used to analyse the expected power consumption of a networks of nodes that must achieve some desirable degree of synchrony.

5.3 Reward functions for time and power consumption

The reduced models are now annotated with rewards for which expected values over runs of the model can be obtained. For a network of WSN nodes the properties of interest are the time taken for the network to reach a state where some desirable degree of synchrony has been achieved, and the power consumption of the network.

5.3.1 Synchronisation time

The properties of interest here are the mean and maximum times taken for a network of WSN nodes to reach a state where some desirable degree of synchrony has been achieved. By accumulating the reward along a path until such a state is reached a measure of the time taken to synchronise can be obtained. We denote the reward function corresponding to synchronisation time by $\mathbf{R}^{\mathsf{t}}$.

Recall that deterministic transitions are omitted in the reduced model, and instead a transition is taken directly from a firing state to the next firing state. For each firing state s^{F} recall that

$$\overset{\frown}{\mathsf{Next}}(s^{\mathsf{F}}) = \{\overset{\frown}{\mathsf{succ}}(\overrightarrow{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f})) \mid \mathbf{f} \in \mathcal{F}_{s^{\mathsf{F}}}\},$$

is the set of all possible successor firing states of s^{F}. For each $s \in \overset{\curvearrowright}{\mathsf{Next}}(s^{\mathsf{F}})$, with corresponding failure vector $\mathbf{f} \in \mathcal{F}_{s^{\mathsf{F}}}$ for s^{F} and successor state $\overrightarrow{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f}) = \langle k_1, \ldots, k_T \rangle$ in the unreduced model, the highest phase of any oscillator in its successor state is given by

$$\delta = \max\{\Phi \mid k_\Phi > 0 \text{ and } 1 \leqslant \Phi \leqslant T\}.$$

From Definition 23 it is then clear that in the unreduced model $T - \delta$ deterministic transitions would have been taken from $\overrightarrow{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f})$ to the next firing state

$$\overset{\curvearrowright}{\mathsf{succ}}(\overrightarrow{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f})).$$

Therefore, the reward for taking a transition from s^{F} to s is set to

$$\mathbf{R}^{\mathsf{t}}(s^{\mathsf{F}}, s) = \frac{T - \delta}{T}.$$

The detailed calculation for the reward for taking a transition from the initial state to some firing state s^{F} is more involved, and is discussed in more detail in Section 5.4. By observing that each path of the reduced model, where s^{F} is the first firing state along that path, corresponds to several paths of the unreduced model, the reward is obtained by taking the sum of the total number of initial deterministic transitions taken to s^{F} across such paths, weighted by the probability of taking each path, and where the total is normalised by the probability of taking any of those paths.

5.3.2 Power consumption

The property of interest here is the total power consumed by a network of WSN nodes to reach a state where some desirable degree of synchrony has been achieved. We denote the

reward function corresponding to synchronisation time by $\mathbf{R}^{\mathsf{P}}$. Let I_{id}, I_{rx}, and I_{tx} be the current draw in amperes for the idle, receive, and transmit modes, V be the voltage, C be the length of the oscillation cycle in seconds, and M be the time taken to transmit a synchronisation message in seconds. The power consumption in Watt-hours of one node for one discrete step within its refractory period, that is, the node is in the idle mode, is

$$W_{id} = \frac{I_{id}VC}{3600T}.$$

Similarly, if the node is outside of the refractory period, that is, it is in the receive mode, the corresponding power consumption is defined by

$$W_{rx} = \frac{I_{rx}VC}{3600T}.$$

Finally, let

$$W_{tx} = \frac{I_{tx}VM}{3600}$$

be the power consumption in Watt-hours to transmit one synchronisation message. The power consumption of the network consists of the power necessary to transmit the synchronisation messages, and that of the node in the idle and receive modes.

We associate a reward with each state $s = \langle k_1, \ldots, k_T \rangle$ that corresponds to the total power consumption for nodes in the idle and receive modes in that state,

$$\mathbf{R}^{\mathsf{P}}(s) = \sum_{\Phi=1}^{R} k_\Phi W_{id} + \sum_{\Phi=R+1}^{T} k_\Phi W_{rx}.$$

For any firing state s^{F} and any failure vector $\mathbf{f}$ for that state, the power consumed by the network to transmit synchronisation messages and then reach the next firing state is

equivalent to the accumulation of the power consumption of the network in s^{F} and all non-firing successor states of s^{F} in the unreduced model – those that are skipped in the transition from s^{F} to $\overset{\curvearrowright}{\mathsf{succ}}(\overset{\rightarrow}{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f}))$ in the reduced model. Given

$$\overset{\rightarrow}{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f}) = \langle k_1, \ldots, k_T \rangle,$$

the successor of s^{F} with respect to $\mathbf{f}$ in the unreduced model, let $\delta = \max\{\Phi \mid k_\Phi > 0 \text{ and } 1 \leqslant \Phi \leqslant T\}$ be the maximal phase of any oscillator in that state. The reward for taking the transition from s^{F} to $\overset{\curvearrowright}{\mathsf{succ}}(\overset{\rightarrow}{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f}))$ is then given by

$$\mathbf{R}^{\mathsf{p}}(s^{\mathsf{F}}, \overset{\curvearrowright}{\mathsf{succ}}(\overset{\rightarrow}{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f}))) = k_1 W_{tx} + \sum_{j=0}^{(T-\delta)-1} \left(\sum_{\Phi=1}^{R-j} k_\Phi W_{id} + \sum_{\Phi=(R+1)-j}^{\delta} k_\Phi W_{rx} \right),$$

where $k_1 W_{tx}$ is the total power consumption for the transmission of k_1 synchronisation messages, and where the summand accumulates the power consumption over $\overset{\rightarrow}{\mathsf{succ}}(s^{\mathsf{F}}, \mathbf{f})$ and subsequent $(T - \delta) - 1$ non-firing states. The left and right and right summands inside the brackets accumulate the power consumption of nodes within, and outside of the refractory period, respectively.

Again the detailed calculation for the reward for taking a transition from the initial state to some firing state s^{F} is more involved, and is discussed in more detail in the next section.

5.4 Reward functions for reduced models

In the previous section, two reward functions were defined for the reduced population model that encoded rewards accumulated across the states and transitions removed as part of the model reduction process. The definitions are now generalised so that a reward function for

the reduced model can be derived from any reward function for the unreduced model.

Theorem 2. *For every population model $\mathcal{P}$ with corresponding PMC $\mathcal{D}_{\mathcal{X}} = (\mathcal{S}, s_0, \mathbf{P}, L, \mathcal{X})$ and reduction $\mathcal{D}'_{\mathcal{X}} = (\mathcal{S}', s_0, \mathbf{P}', L', \mathcal{X})$ of $\mathcal{D}_{\mathcal{X}}$, and for every reward function $\mathbf{R}$ for $\mathcal{D}_{\mathcal{X}}$, there is a reward function $\mathbf{R}'$ for $\mathcal{D}'_{\mathcal{X}}$ such that unbounded-time reachability reward properties with respect to synchronised firing states in $\mathcal{D}_{\mathcal{X}}$ are preserved in $\mathcal{D}'_{\mathcal{X}}$ for all total well-defined evaluations for $\mathcal{X}$.*

Given a reward function $\mathbf{R}$ for $\mathcal{D}_{\mathcal{X}}$ the corresponding reward function for $\mathcal{D}'_{\mathcal{X}}$ is constructed as follows:

- There is no reward for the initial state, since oscillators are not configured here, and $\mathbf{R}'(s_0) = 0$.
- For every firing state s^{F} in $\mathcal{S}$ with $\mathbf{R}(s^{\mathsf{F}}) = r$ also $\mathbf{R}'(s^{\mathsf{F}}) = r$.
- For every pair of distinct firing states $s_1^{\mathsf{F}}, s_2^{\mathsf{F}} \in \mathcal{S}'$, where there is a non-zero transition from s_1^{F} to s_2^{F} in $\mathcal{D}'_{\mathcal{X}}$, there is a (possibly empty) sequence $s_1^{\mathsf{NF}} \cdots s_k^{\mathsf{NF}}$ of k deterministic predecessors of s_2^{F} in $\mathcal{S}$ such that $k > 0$ implies $\mathbf{P}(s_1^{\mathsf{F}}, s_1^{\mathsf{NF}}) > 0$, $\mathbf{P}(s_k^{\mathsf{NF}}, s_2^{\mathsf{F}}) = 1$, and $\mathbf{P}(s_i^{\mathsf{NF}}, s_{i+1}^{\mathsf{NF}}) = 1$ for $1 \leqslant i < k$. The reward for taking the transition from s_1^{F} to s_2^{F} in $\mathcal{D}'_{\mathcal{X}}$ is set to be the sum of the rewards that would be accumulated across that sequence by a path in $\mathcal{D}_{\mathcal{X}}$, formally

$$\mathbf{R}'(s_1^{\mathsf{F}}, s_2^{\mathsf{F}}) = \mathsf{total}_{\mathbf{R}}(s_1^{\mathsf{F}} s_1^{\mathsf{NF}} \cdots s_k^{\mathsf{NF}} s_2^{\mathsf{F}}).$$

- For every firing state s^{F} in $\mathcal{S}'$ there is a non-zero transition from the initial state s_0 to s^{F} in $\mathbf{P}'$. Therefore, all paths of $\mathcal{D}'_{\mathcal{X}}$ where s^{F} is the first firing state along that path share the same prefix, namely s_0, s^{F}. For paths of $\mathcal{D}_{\mathcal{X}}$ this is not necessarily the case, since s^{F} is the first firing state not only along the path where the initial

transition is taken to s^{F} itself, but also along any path where the initial transition is taken to a non-firing state from which a sequence of deterministic transitions leads to s^{F} (that state is a deterministic predecessor of s^{F}). We therefore set the reward along a path $\omega' = s_0 s_1^{\mathsf{F}} s_2^{\mathsf{F}} \cdots$ for taking the initial transition to s^{F} in $\mathcal{D}'_{\mathcal{X}}$ to be the sum of the total rewards accumulated along all distinct path prefixes of the form $s_0 \omega^{\mathsf{NF}} s^{\mathsf{F}}$, normalised by the total probability of taking any of these paths, where ω^{NF} is a possibly empty sequence of deterministic predecessors of s^{F}, and where the total reward for each prefix is weighted by the probability of taking the transitions along that sequence,

$$\mathbf{R}'(s_0, s^{\mathsf{F}}) = \frac{\sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \mathsf{total}_{\mathbf{R}}(\omega_{\mathsf{pre}}) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}}}\{\omega_{\mathsf{pre}}\}}{\mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}}}\{s_0 s_1^{\mathsf{F}}\}}$$

Proof of Theorem 2. It is sufficient to show that for every reward function $\mathbf{R}$ for $\mathcal{D}_{\mathcal{X}}$ and corresponding reward function $\mathbf{R}'$ for $\mathcal{D}'_{\mathcal{X}}$, every $\bowtie \in \{<, \leqslant, \geqslant, >\}$ and every $r \in \mathbb{R}$, if $s_0 \models \mathsf{R}_{\bowtie r}[\mathsf{F}\,\mathsf{synch}]$ holds in $\mathcal{D}_{\mathcal{X}}$ then it also holds in $\mathcal{D}'_{\mathcal{X}}$. Assume that we have some total well-defined evaluation for $\mathcal{X}$, yielding the DTMCs

$$\mathcal{D}_{\mathcal{X}_\upsilon} = (\mathcal{S}, s_0, \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon], L, \mathcal{X}_\upsilon),$$
$$\mathcal{D}'_{\mathcal{X}_\upsilon} = (\mathcal{S}', s_0, \mathbf{P}[\mathsf{Dom}(\upsilon)/\upsilon]', L', \mathcal{X}_\upsilon).$$

Let $X_{\mathsf{Sat}(\mathsf{F}\,\mathsf{synch})}$ and $X'_{\mathsf{Sat}(\mathsf{F}\,\mathsf{synch})}$ respectively denote the random variables over $\mathsf{Paths}^{\mathcal{D}_{\mathcal{X}_\upsilon}(s_0)}$ and $\mathsf{Paths}^{\mathcal{D}'_{\mathcal{X}_\upsilon}}(s_0)$ whose expectations correspond to $\mathbf{R}$ and $\mathbf{R}'$. From the semantics of

PCTL over a DTMC,

$$\begin{aligned} s_0 \models \mathsf{R}_{\bowtie r}[\mathrm{F}\,\mathsf{synch}] \quad &\Leftrightarrow \mathrm{E}[X_{\mathsf{Sat(synch)}}] \bowtie r \\ &\Leftrightarrow \sum_{\omega \in \mathsf{Paths}} X_{\mathsf{Sat(synch)}} \mathsf{Pr} s_0\{\omega\} \bowtie r. \end{aligned}$$

Therefore, if is sufficient to show that

$$\sum_{\omega \in \mathsf{Paths}^{\mathcal{D}_{\mathcal{X}_v}}} X_{\mathsf{Sat(synch)}}(\omega) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega\} = \sum_{\omega' \in \mathsf{Paths}^{\mathcal{D}'_{\mathcal{X}_v}}} X'_{\mathsf{Sat(synch)}}(\omega') \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'\}, \tag{5.1}$$

where $\mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}$ and $\mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}$ denote the probability measures with respect to the sets of infinite paths from s_0 in $\mathcal{D}_{\mathcal{X}_v}$ and $\mathcal{D}'_{\mathcal{X}_v}$ respectively. There are two cases:

Firstly, if there exists some path of $\mathcal{D}_{\mathcal{X}_v}$ that does not synchronise then by definition $X_{\mathsf{Sat(synch)}} = \infty$. Also, from Lemma 7 there is a corresponding path of $\mathcal{D}'_{\mathcal{X}_v}$ that does not synchronise, and hence that $X'_{\mathsf{Sat(synch)}} = \infty$. By definition the probability measure of all paths of $\mathcal{D}_{\mathcal{X}_v}$ and $\mathcal{D}'_{\mathcal{X}_v}$ are strictly positive. Therefore, all summands of Equation 5.1 are defined, and the expectation of both $X_{\mathsf{Sat(synch)}}$ and $X'_{\mathsf{Sat(synch)}}$ is ∞.

Secondly, the case where all possible paths of $\mathcal{D}_{\mathcal{X}_v}$ and $\mathcal{D}'_{\mathcal{X}_v}$ synchronise. First let

$$\mathsf{reduce} : \mathsf{Paths}^{\mathcal{D}_{\mathcal{X}_v}} \to \mathsf{Paths}^{\mathcal{D}'_{\mathcal{X}_v}}$$

be the function mapping paths of $\mathcal{D}_{\mathcal{X}_v}$ to their corresponding path in the reduction $\mathcal{D}'_{\mathcal{X}_v}$,

$$\mathsf{reduce}(s_0 \omega_1^{\mathsf{NF}} s_1^{\mathsf{F}} \omega_2^{\mathsf{NF}} s_2^{\mathsf{F}} \cdots) = s_0 s_1^{\mathsf{F}} s_2^{\mathsf{F}} \cdots,$$

where ω_i^{NF} is the (possibly empty) sequence of deterministic predecessors of the firing state s_i^{F}. Let $\mathsf{reduce}^{-1}(\omega)$ denote the preimage of ω under reduce. Then the left side of (5.1) can

be rewritten to

$$\sum_{\omega' \in \mathsf{Paths}^{\mathcal{D}'_{\mathcal{X}_v}}} \sum_{\omega \in \mathsf{reduce}^{-1}(\omega')} X_{\mathsf{Sat(synch)}}(\omega) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega\}.$$

For any path ω of $\mathcal{D}_{\mathcal{X}_v}$ or $\mathcal{D}'_{\mathcal{X}_v}$ let $\mathsf{pref}_s(\omega)$ be the prefix of that path whose last state is the first firing state along that path that is in the set $\mathsf{Sat(synch)}$. It is now sufficient to show that the following holds for any path ω' of $\mathcal{D}'_{\mathcal{X}_v}$,

$$\sum_{\omega \in \mathsf{reduce}^{-1}(\omega')} X_{\mathsf{Sat(synch)}}(\omega) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega\} = X'_{\mathsf{Sat(synch)}}(\omega') \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'\}$$

$$\sum_{\omega \in \mathsf{reduce}^{-1}(\omega')} \mathsf{total}_{\mathbf{R}}(\mathsf{pref}_s(\omega)) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega\} = \sum_{\omega' \in \mathsf{Paths}^{\mathcal{D}'_{\mathcal{X}_v}}} \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')) \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'\}. \quad (5.2)$$

Given some path ω let $\omega[i:j]$ denote the sequence of states in ω from the i^{th} firing state to the j^{th} firing state along that path (inclusively). The notation $\omega[-:j]$ indicates that no states are removed from the start of the path i.e. the first state is s_0, and the notation $\omega[i:-]$ indicates that no states are removed from the end of the path. By recalling that

$$\mathsf{Pr}\{s_0 s_1 \cdots s_n\} = \prod_{i=1}^{n} \mathbf{P}(s_{i-1}, s_i)$$

we can see that

$$\mathsf{Pr}\{s_0 s_1 \cdots s_n\} = \mathsf{Pr}\{s_0 \cdots s_i\} \mathsf{Pr}\{s_i \cdots s_n\}$$

for any $0 < i < n$. Also from (2.1) it is clear that for any reward function $\mathbf{R}$,

$$\mathsf{total}_{\mathbf{R}}(s_0 \cdots s_n) = \mathsf{total}_{\mathbf{R}}(s_0 \cdots s_i) + \mathsf{total}_{\mathbf{R}}(s_i \cdots s_n)$$

holds for all $0 < i < n$. Now (5.2) can be rewritten to

$$\sum_{\omega \in \mathsf{reduce}^{-1}(\omega')} \left(\mathsf{total}_{\mathbf{R}}(\mathsf{pref}_s(\omega)[- : 1]) + \mathsf{total}_{\mathbf{R}}(\mathsf{pref}_s(\omega)[1 : -])\right) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega[- : 1]\} = \left(\mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[- : 1]) + \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1 : -])\right) \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'[- : 1]\}. \tag{5.3}$$

By the definition of $\mathbf{R}'$ the right hand side of (5.3) can be rewritten as

$$\left(\left(\frac{\sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \mathsf{total}_{\mathbf{R}}(\omega_{\mathsf{pre}}) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega_{\mathsf{pre}}\}}{\mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'[- : 1]\}}\right) + \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1 : -])\right) \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'[- : 1]\} = \sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \left(\mathsf{total}_{\mathbf{R}}(\omega_{\mathsf{pre}}) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega_{\mathsf{pre}}\}\right) + \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1 : -]) \mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'[- : 1]\}.$$

From Lemma 7 it is known that

$$\mathsf{Pr}^{\mathcal{D}'_{\mathcal{X}_v}}\{\omega'[- : 1]\} = \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}} \mathsf{Pref}(\omega') = \sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega_{\mathsf{pre}}\},$$

and hence

$$\sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \left(\mathsf{total}_{\mathbf{R}}(\omega_{\mathsf{pre}}) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega_{\mathsf{pre}}\}\right) + \sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1 : -]) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega_{\mathsf{pre}}\} = \sum_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \left(\mathsf{total}_{\mathbf{R}}(\omega_{\mathsf{pre}}) + \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1 : -])\right) \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_v}}\{\omega_{\mathsf{pre}}\}. \tag{5.4}$$

Since $\mathsf{Pref}(\omega')$ is the set of all possible finite prefixes from the initial state s_0 to the first firing state s_1^{F}, and since $\omega[- : 1] = \mathsf{pref}_s(\omega)[- : 1]$ clearly holds,

$$\bigcup_{\omega_{\mathsf{pre}} \in \mathsf{Pref}(\omega')} \{\omega_{\mathsf{pre}}\} = \bigcup_{\omega \in \mathsf{reduce}^{-1}(\omega')} \{\omega[- : 1]\}. = \bigcup_{\omega \in \mathsf{reduce}^{-1}(\omega')} \{\mathsf{pref}_s(\omega)[- : 1]\}.$$

Using this fact, and by observing that by definition

$$\mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1:-]) = \mathsf{total}_{\mathbf{R}}(\mathsf{pref}_s(\omega)[1:-]),$$

(5.4) can be written as

$$\sum_{\omega \in \mathsf{reduce}^{-1}(\omega')} \big(\mathsf{total}_{\mathbf{R}}(\mathsf{pref}_s(\omega)[-:1]) + \mathsf{total}_{\mathbf{R}'}(\mathsf{pref}_s(\omega')[1:-])\big)\, \mathsf{Pr}^{\mathcal{D}_{\mathcal{X}_\upsilon}}\{\omega[-:1]\}.$$

This is the same as the left hand side of (5.3) and the theorem is proved. □

5.5 Restabilisation

A network of oscillators is *restabilising* if it has reached a synchronised state, synchrony has been lost due to the occurrence of some external event, and the network must then again achieve synchrony. One could, for instance, imagine the introduction of additional nodes with arbitrary phases to an established and synchronised network. While such a change is not explicitly encoded within the population model, it can be represented by partitioning the set of oscillators into two subsets. Let the parameter A be the number of oscillators with arbitrary phase values that have been introduced into a network of $N - A$ synchronised oscillators, or to be the number of oscillators in a network of N oscillators whose clocks have reset to an arbitrary value, where $A \in \mathbb{N}$ and $1 \leqslant A < N$. Destabilising A oscillators in this way results in configurations where *at least* $N - A$ oscillators are synchronised, since the destabilised oscillators may coincidentally be assigned the phase of the synchronised group. the set of initial configurations can be restricted by identifying

the set

$$\mathcal{S}_A = \{\langle k_1, \ldots, k_T \rangle \in \mathcal{S} \mid k_i \geqslant N - A \text{ for some } 1 \leqslant i \leqslant T\},$$

where each $s \in \mathcal{S}_A$ is a configuration for the phases such that at least $N - A$ oscillators share some phase and the remaining oscillators have arbitrary phase values. As the value of A is decreased, the number of initial configurations for the phases of the oscillators (and hence, states of the PMC) also decreases. Since the model does not encode the loss or addition of oscillators it is clear that all global states where there are less than $N - A$ oscillators sharing the same phase are unreachable from any state in $\mathcal{S}_A$.

5.6 Evaluation

In this section, the results of model checking instantiations of the reduced population model with respect to the property of interest are given – the expected power consumption of the network to reach a state where some desirable degree of synchronisation has been achieved.

The models that are analysed in this section are again programmatically generated using an extended version of the script described in Section 4.4, that generates rewards for the models according to the reward functions defined in Section 5.4 and annotates model states with their phase coherence, as defined in Section 5.2.

The phase response function is instantiated to correspond to the Mirollo and Strogatz model of synchronisation presented in Chapter 4.5.1, where the perturbation induced by the firing of other oscillators is linear in the phase of the perturbed oscillator and the number of firing oscillators, and the coupling constant determines the slope of the linear dependency: $\Delta(\Phi, \alpha, \epsilon) = \lfloor \Phi\alpha\epsilon \rceil$.

For many experiments ϵ is set to 0.1 and μ is set to 0.2. Of course, other analyses

could have been conducted for different values for these parameters. For a real system, the probability μ of broadcast failure occurrence is highly dependent on the deployment environment. For deployments in benign environments a relatively low rate of failure would be expected, for instance a WSN within city limits under controlled conditions, whilst a comparably high rate of failure would be expected in harsh environments such as a network of off-shore sensors below sea level. The coupling constant ϵ is a parameter of the system itself. The results suggest that higher values for ϵ are always beneficial, however this is because analysis is restricted to fully connected networks. High values for ϵ may be detrimental when considering different topologies, since firing nodes may perturb subcomponents of a network that have achieved local synchrony. However, such an analysis is deferred to future work.

As an example, power consumption is analysed for values taken from the datasheet of the *MICAz* mote [115]. The transceiver of the MICAz mote has several modes. It can either transmit, receive, or remain idle. The idle and transmit modes are composed of several sub-modes. The transmit mode has three sub-modes for different transmission ranges, each of which influence the amount of current draw. The current draw of the idle mode depends on whether the voltage regulator is turned on or off. To account for the worst-case, only sub-modes with the maximal current draw are considered. The current draw of the mote is then assumed to be

$$\begin{aligned} I_{tx} &= 17.4\ mA \quad \text{in transmit mode,} \\ I_{rx} &= 19.7\ mA \quad \text{in receive mode,} \\ I_{id} &= 20\ \mu A \quad\ \ \text{when the transceiver is idling.} \end{aligned}$$

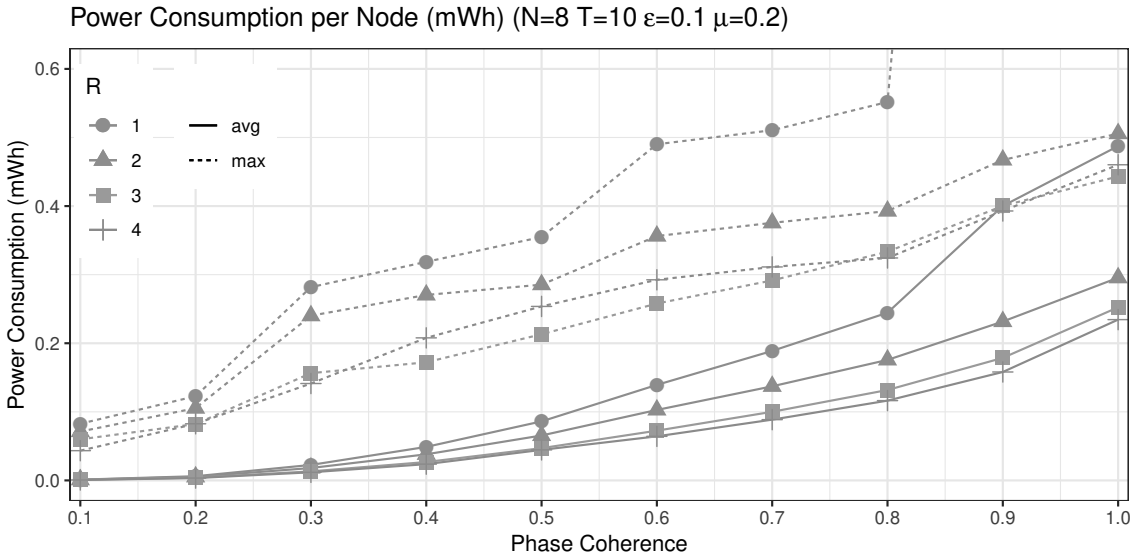
The MICAz is powered by two AA batteries or an external power supply with a voltage of $2.7 - 3.3\ V$. For consistency, it is assumed that the voltage of its power supply is $3.0\ V$.

5.6.1 Power consumption of a synchronising network

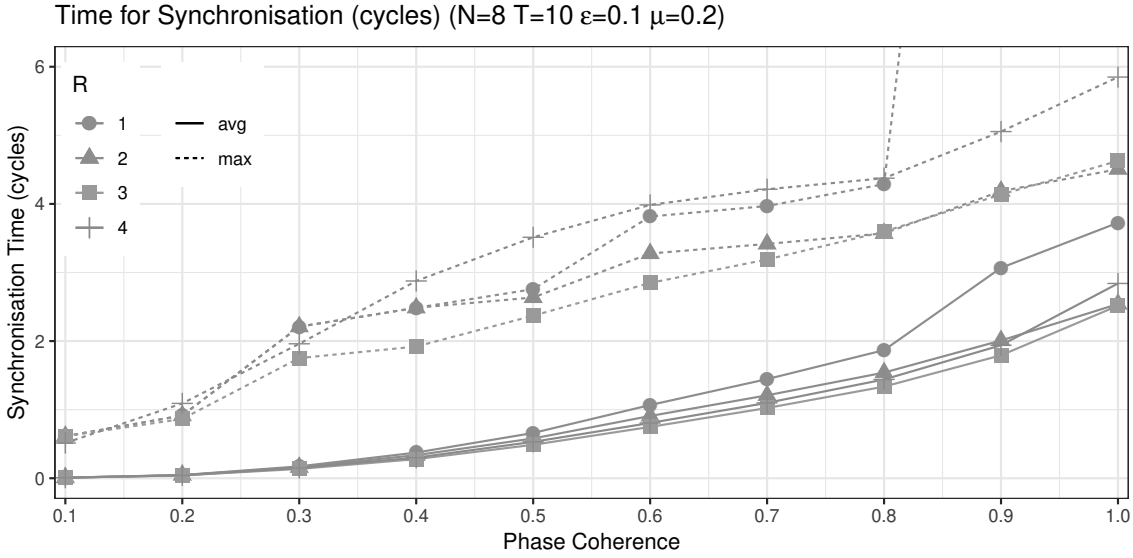
The power consumption and time taken for a fully connected network of oscillators to reach a state where some desirable degree of synchrony has been achieved is now analysed, where the granularity of the discrete oscillation cycle is given by $T = 10$. For this granularity it was possible to analyse larger networks than those considered in Chapter 4. Given the memory resources of the machine used for the analysis (16GB) networks of up to 10 oscillators could be analysed. However, since the cumulative model checking time over all model instantiations was very large (The results shown in Figure 5.4a already amount to 80 distinct runs.) a network of size $N = 8$ was chosen for the analysis.

Figures 5.4a and 5.4b show both the average and maximal power consumption per node (in mWh) and time (in oscillation cycles) needed to achieve some desirable degree of synchrony, with respect to different lengths of the refractory period, where $\epsilon = 0.1$ and $\mu = 0.2$. That is, they show the average, and maximal, power that is consumed (time that is needed, respectively) for a system in an arbitrary state to reach a state where some degree of phase coherence has been achieved. Let $\mathsf{coherent}_\lambda$ be a predicate that holds for any state s where $\mathsf{PCF}(s) \geqslant \lambda$. Then for the average case, the corresponding PCTL property checked is $\mathsf{R}[\mathsf{F}\,\mathsf{coherent}_\lambda]$, with respect to the reward function $\mathbf{R}^{\mathrm{p}}$ for power consumption and $\mathbf{R}^{\mathrm{t}}$ for synchronisation time. For the maximal case the *max* filter[1] of the PRISM model checker is used. This filter gives the maximum expected reward across all paths starting in states that satisfy some given predicate. The desired maximal values are therefore obtained by checking the property $\mathsf{R}[\mathsf{F}\,\mathsf{coherent}_\lambda]$ against all states of the model where oscillators are configured i.e. all $s \in \mathcal{S} \setminus s_0$, again with respect to the reward function $\mathbf{R}^{\mathrm{p}}$ for power consumption and $\mathbf{R}^{\mathrm{t}}$ for synchronisation time. The results show that for both the average and maximal cases the power consumption of the network increases monotonically with

[1] https://www.prismmodelchecker.org/manual/PropertySpecification/Filters



(a)



(b)

Figure 5.4: Expected rewards for power consumption (a) and synchronisation time (b) for a network of 8 nodes to achieve different degrees of phase coherence.

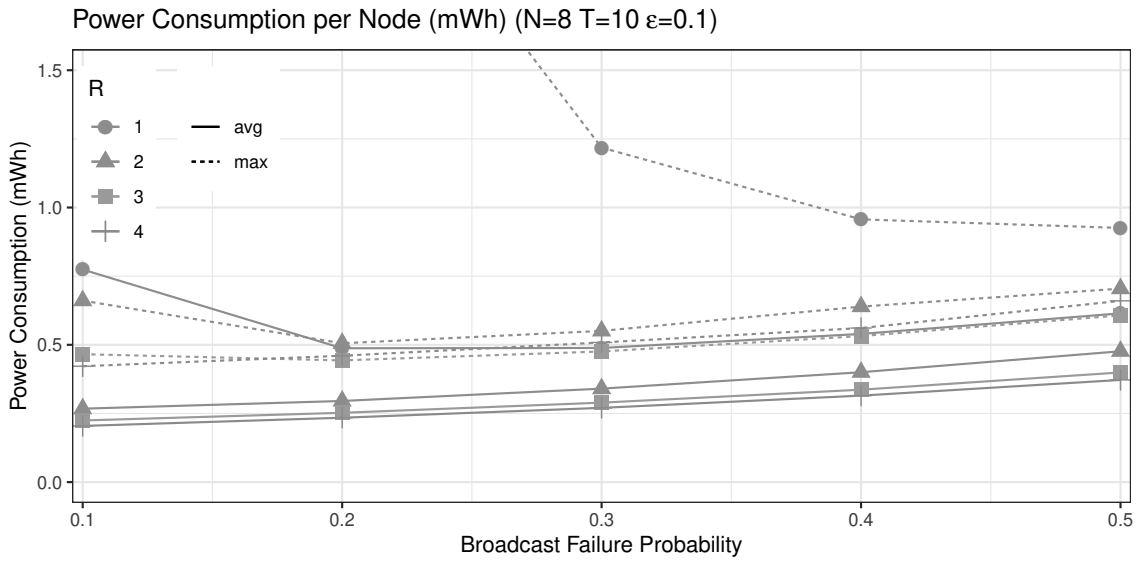


Figure 5.5: The relationship between broadcast failure probability and power consumption.

the required degree of phase coherence, as expected, which validates the metric defined in Section 5.2.

The much larger values obtained for $R = 1$ and phase coherence $\geqslant 0.9$ are not shown here, to avoid distortion of the figures. The energy consumption for these values is approximately $2.4mWh$, while the time needed is approximately 19 cycles. Observe that only values for the refractory period R with $R < \frac{T}{2}$ are shown. For larger values of R not all runs synchronise, as shown in Section 4.5, resulting in an infinitely large reward being accumulated for both the maximal and average cases. Results are not presented for the minimal power consumption (or time) as it is always zero, since all initial configurations (global states) for oscillator phases are considered. In particular, runs where the initial state is already synchronised are considered.

As expected when starting from an arbitrary state, the time and power consumption increases monotonically with the order of synchrony to be achieved. On average, networks with longer refractory periods (less than $\frac{T}{2}$) require less power for synchronisation, and

take less time to achieve it. The only exception is that the average time to achieve synchrony with a refractory period of four is higher than for two and three. However, if lower phase coherence is sufficient then this trend is stable. In contrast, the maximal power consumption of networks with $R = 4$ is consistently higher than of networks with $R = 3$. In addition, the maximal time needed to achieve synchrony for networks with $R = 4$ is higher than for lower refractory periods, except when the phase coherence is greater than or equal to 0.9. Networks with a refractory period of three will need the shortest times to synchronise, regardless of whether the maximal or average values are considered. Furthermore, the average power consumption for full synchronisation (phase coherence 1) differs only slightly between $R = 3$ and $R = 4$ (less than $0.3mWh$). Hence, for the given example, $R = 3$ gives the best results. These relationships are stable even for different broadcast failure probabilities μ, while the concrete values increase only slightly, as illustrated in Figure 5.5, which shows the power consumption for different μ when $\epsilon = 0.1$.

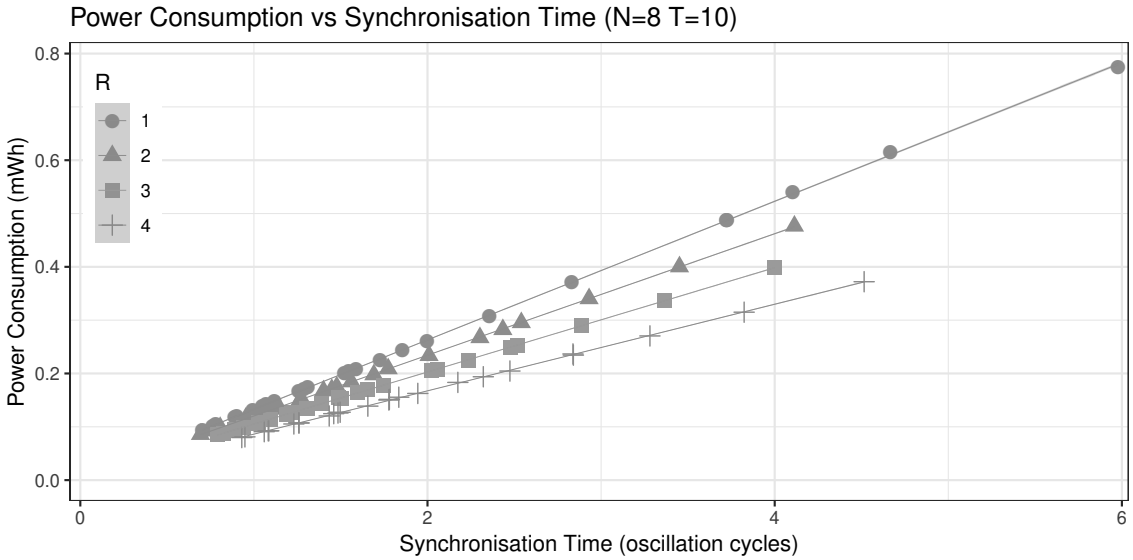
The general relationship between power consumption and synchronisation time is shown in Figures 5.6a and 5.6b. Within these figures, different coupling constant values and broadcast failure probabilities are not distinguished. The two values for $R = 1$, $\epsilon = 0.1$ and $\mu \in \{0.1, 0.2\}$ are omitted in Figure 5.6b to avoid distortion of the graph, since the low coupling strength and low probability of broadcast failure leads to longer synchronisation times and hence higher power consumption. While this might seem surprising it has been shown that uncertainty in discrete systems often aids convergence [56]. The relationship between power consumption and time to synchronise is linear, and the slope of the relation decreases for higher refractory periods. While the linearity is almost perfect for the average values, the maximal values have larger variation. The figures again suggest that $R = 3$ is a sensible and reliable choice, since it provides the best stability in terms of power consumption and synchronisation time. In particular, if the broadcast failure probability

changes, the variations are less severe for $R = 3$ than for other refractory period lengths.

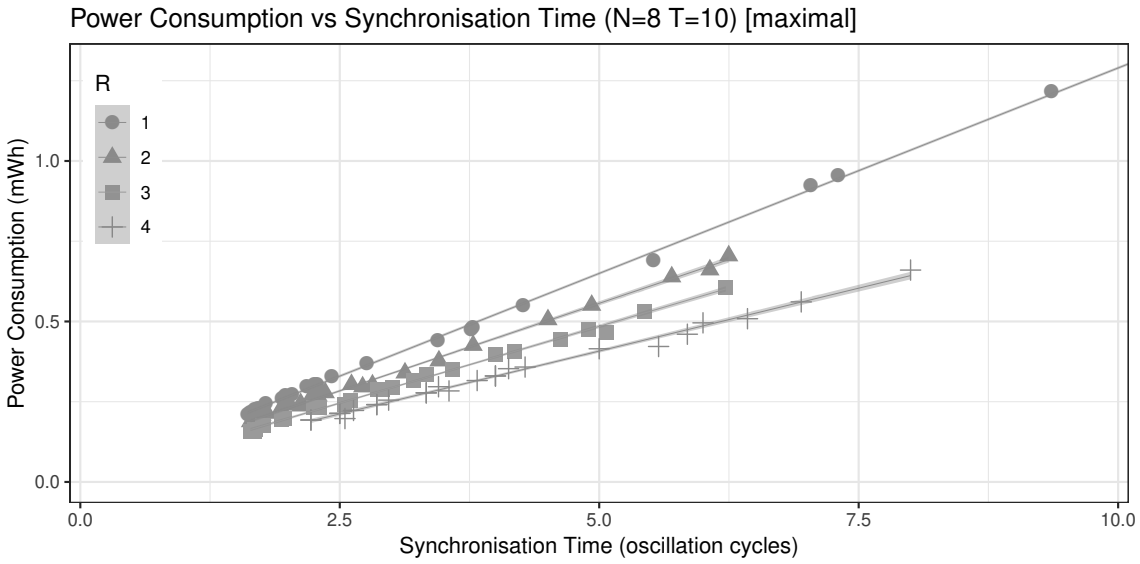
5.6.2 Power consumption for network restabilisation

The power consumption of a resynchronising network is now investigated. A network is resynchronising if it has already achieved a synchronised state but some event has caused synchrony to be temporarily lost. This event could be the clocks of one or more nodes changing due to clock drift or some internal error, an environmental factor causing some nodes to reset or temporarily lose connection to the network, or one or more new nodes being introduced to an existing network. The approach presented in Section 5.5 greatly decreases the state space of the models to be analysed, when compared to both the unreduced, and reduced, population models presented in the previous chapter, and hence allow the network size to be increased significantly. In particular, the smallest network analysed is already larger than that in the analysis above, while the largest is almost five times as large.

The average power consumption per node for networks of size $N \in \{10, 15, \ldots, 35\}$, where the oscillators are coupled with strength $\epsilon = 0.1$, and broadcast failure probability $\mu = 0.3$, is shown in Figure 5.7. Here the power consumption until the system is in fully synchronised is of interest, and the PCTL property checked is $\mathsf{R}[\mathsf{F}\,\mathsf{coherent}_1]$ with respect to the reward function $\mathbf{R}^{\mathrm{P}}$. The solid lines denote the results for a single redeployed node, while the dashed lines represent the results for the redeployment of two and three nodes, respectively. As expected, more nodes consume more energy to resynchronise. However, it can also be seen that for higher refractory periods, the amount of energy needed is more or less stable; in particular, for the case where $R = 4$, which is already invariant for more than ten nodes. For smaller refractory periods, increasing the network size decreases the average energy consumption. This behaviour can be explained as follows. The linear



(a)



(b)

Figure 5.6: Average (a) and maximal (b) expected power consumption for different synchronisation times.

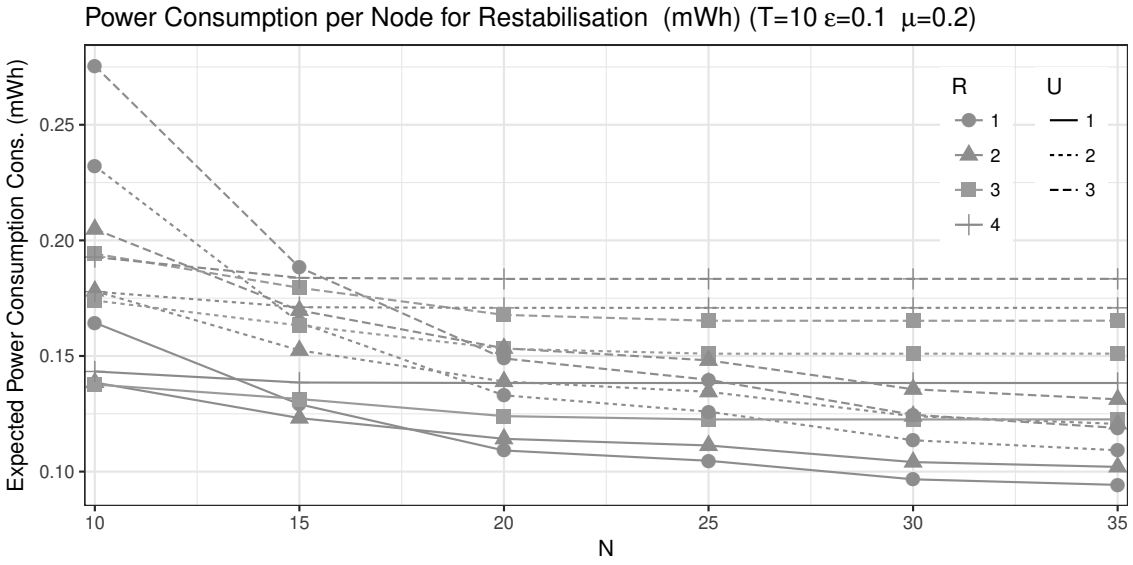


Figure 5.7: Power consumption for restabilisation for a range of network sizes.

synchronisation model implies that oscillators with higher phase values will incur higher perturbation to their phase, and thus are more likely to fire. Hence, in general a larger network will force the node to resynchronise faster. The refractory period determines how large the network has to be for this effect to stabilise.

5.7 Conclusion

A new metric for synchronisation was introduced that extended the binary notion of synchronisation used in the previous chapter. The metric facilitated reasoning about a network of oscillators reaching some state where it has achieved some desirable degree of synchrony; in particular, the property of interest investigated was the power consumed by such a network to achieve that state. A suitable reward function was then defined that would encode the rate of power consumption for a wireless sensor node, or member of a swarm, given some concrete values for the power consumption of such a device. It was also shown that

for any reward function for the unreduced model, there is a reward function for the reduced model that is equivalent with respect to unbounded reachability reward properties for states where one or more oscillators are firing.

Properties relating to the restabilisation of a small number of oscillators – when the network has synchronised, and synchrony has then been lost – in a network was then discussed, and it was shown that only a fraction of the state-space of the model needs to be explored, since only a small subset of the number of initial configurations for the oscillators needs to be considered.

The introduced techniques were then used to analyse the power consumption for the synchronisation and restabilisation of a network of MICAz motes, by instantiating the reduced population model with the model of synchronisation defined by Mirollo and Strogatz [117], and by instantiating the power consumption reward function using the values taken from the datasheet for the mote [115]. Using the reduced model it was possible to extend the size of the networks that could be analysed in the previous chapter, and trade-offs between the time and power needed to synchronise for different lengths of the refractory period (or duty-cycle of a node) were then discussed for these networks.

Results obtained using these techniques can be used by designers of WSNs to estimate the overall energy efficiency of a network during its design phase. Unnecessary energy consumption can be identified and rectified before network deployment. Also, the results provide guidance for estimating the battery life of a network depending on the anticipated frequency of restabilisations. Of course, these considerations only hold for the maintenance task of synchronisation. The energy consumption of the functional behaviour has to be examined separately.

It is clear that the approach is inhibited by the usual limitation of exact probabilistic model checking for large-scale systems. This could be overcome by using approximated

techniques, such as statistical model checking (see Section 2.6), or approaches based on fluid-flow approximation extended with rewards [20]. This would, of course, come at the expense of precision. An investigation of such a trade-off is deferred to future work. The current approach is restricted to fully connected networks of oscillators. While this is sufficient to analyse the behaviour of strongly connected components within a network, further investigation is needed to assess different network topologies. To that end, several interconnected population models could be used, thus modelling the interactions of the networks subcomponents. Furthermore, topologies that change over time are of particular interest. However, it is not obvious how the approach could be extended to consider such dynamic networks. The work of Lucarelli and Wang may serve as a starting point for further investigations [113]. Stochastic node failure, as well as more subtle models of energy consumption, present significant opportunities for future extensions. For example, in some cases, repeatedly powering nodes on and off over short periods of time might use considerably more power than leaving them on throughout.

Chapter 6

Incremental Verification of Parametric and Reconfigurable Markov Chains

The case studies presented in the Chapters 3, 4, and 5 explored the parameter space for families of parametric probabilistic models. The method used was the same that is often applied by practitioners: adjust the parameters, produce a model, and use a tool like ePMC [71], PRISM [101], or Storm [45] to analyse it.

Parameters investigated were typical of these considered when evaluating distributed systems, for instance the number of interacting entities, or the likelihood of interactions between entities exhibiting some external factor. The reason to explore the parameter space for models such as those investigated can be manifold. Depending on the application, the analyst might simply want to obtain a feeling of how the parameters impact on the behaviour. Another motivation is to see how the model behaves, compare it with observations, and adjust it when it does not match the observations. Regardless of the

reason to adjust the parameter, the changes often lead to structurally similar models. When analysing hundreds of similar models, it becomes paramount to re-use as much of the analysis as possible.

The effects of parameters that do not change the underlying graph of a model are well-studied, however parameters that induce structural change have received less interest. When analysing hundreds of parameterwise different, yet similar, models with such parameters, the time and resources taken can be reduced by re-using as much of the analysis as possible at each step. Some parameters result in dramatic changes to the structure of the underling model; others result in only minor changes where much of the information obtained from the analysis of the previous instance can be re-used for its analysis.

In this chapter novel algorithms built on the state elimination procedures of Hahn [70] discussed in Section 2.5 are introduced. The novelty is to heavily exploit the similarity between instances of parametrised systems. When the parameter grows, the system for the smaller value of the parameter is, broadly speaking, present in the larger system. This observation is used to guide the state-elimination method for parametric Markov chains in such a way, that the model transformations will start with those parts of the system that are stable under increasing the parameter. It is argued that this can lead to a very cheap iterative way to analyse parametric systems, and shown that the approach extends to reconfigurable systems. Finally, it is demonstrated on two benchmarks that the approach scales.

To recall from Section 2.5, state elimination is a technique that successively changes the model by removing states. A state is removed, and the new structure has all successors of this state as (potentially new) successors of the predecessors of this state, with the respectively adjusted probabilities (and, if applicable, expected rewards). If these models are changed in shape and size when adjusting with the parameters, then these changes

tend to be smooth: small changes of the parameters lead to small changes of the model. Moreover, the areas that change – and, consequentially, the areas that do *not* change— are usually quite easy to predict, be it by an automated analysis of the model or by the knowledge of the expert adjusting her model, who would know full well which parts do or do not change when increasing a parameter. These insights inform the order in which the states are eliminated.

When, say, the increase of a parameter allows for re-using all elimination steps but the last two or three, then repeating the analysis is quite cheap. Luckily, this is typically the case in structured models, for example those that can be inductively defined such as a chain-, ring-, or tree-like structure. The Zeroconf protocol [13] is used here as a running example of a structured model. Zeroconf is a protocol for the autonomous configuration of multiple hosts in a network with unique IP addresses (Figure 6.1). When a host joins the network, indicated by state i in the diagram, it selects an address uniformly at random from a available addresses. If the network consists of h hosts then the probability that the selected address is already in use is $q = \frac{h}{a}$.

The host then checks n times if the selected address is already in use by sending a request to the network. If the address is fresh (which happens with a probability of $1 - q$), none of these tests will fail and the address will be classed as new, indicated by state ok in the diagram. If the address is already in use (which happens with a probability of q), then each test is faulty. Collisions go undetected with some likelihood p due to message loss and time-outs. When a collision is detected (which happens with a likelihood of $1 - p$ in each attempt, provided that a collision exists), then the host restarts the process, denoted by a transition back to state i in the diagram. If a collision has gone undetected after n attempts then the host will incorrectly assume that its address is unique, indicated by state err in the diagram.

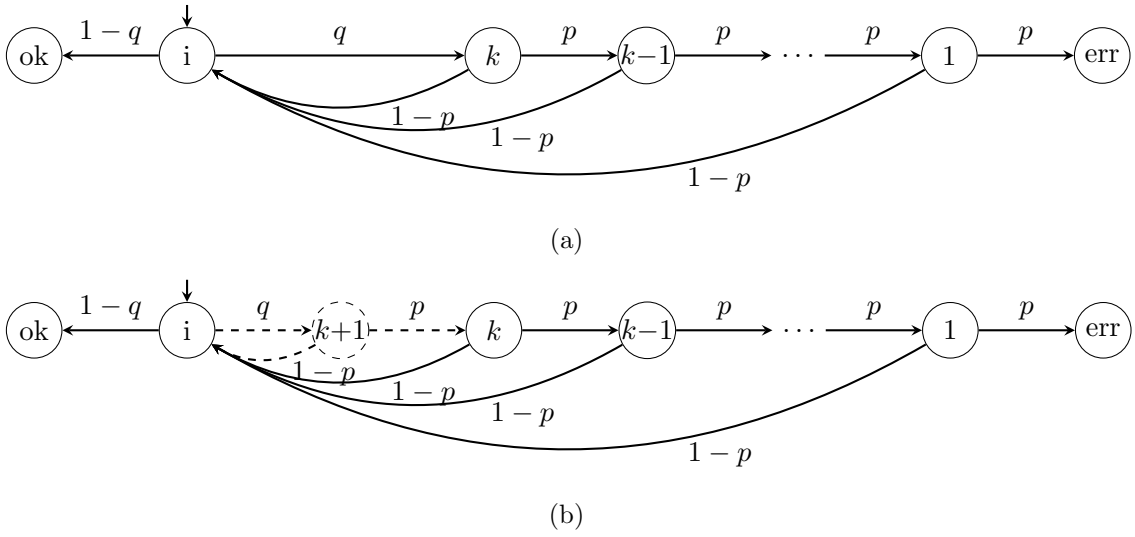


Figure 6.1: The zeroconf protocol for $n = k$ (a) and $n = k + 1$ (b), where n is the number of attempts to check if the selected IP address is already in use.

While the Zeroconf protocol is also parametrised in the transition probabilities, the primary interest is their parametrisation in the structure of the model. Figures 6.1a and 6.1b show the models for $n = k$ and $n = k + 1$ successive checks after each selection of an IP. These two models are quite similar: they structurally differ only in the introduction of the single state $k + 1$, the removal of the transition (i, k), and the introduction of the transitions $(i, k{+}1)$, $(k{+}1, k)$, and $(k{+}1, i)$. If we are interested in calculating the function that represents the probability of reaching the state err in both models, where this function is given in terms of individual rational functions that label the transitions, then the structural similarities allow us to re-use the intermediate terms obtained from the evaluation for $n = k$ when evaluating for $n = k + 1$.

The structure of the rest of this chapter is as follows. The work presented here is compared to similar approaches in the literature in Section 6.1. In Section 6.2, the novel algorithms for the analysis of reconfigured models are introduced. The approach is then

evaluated on two different types of parametric models and discussed in Section 6.3. Section 6.4 concludes the chapter and outlines future work.

6.1 Related work

The techniques introduced in this chapter build on previous results in the area of parametric Markov model checking and incremental runtime verification of stochastic systems.

Daws [41] considered Markov chains that are parametric in the transition probabilities, but not in their graph structure. He introduced an algorithm to calculate the function that represents the probability of reaching a set of target states for all well-defined evaluations for a parametric Markov chain. For this, he interpreted the Markov chain under consideration as a finite automaton, in which transitions are labelled with symbols that correspond to rational numbers or variables. He then used state elimination [84] to obtain a regular expression for the language of the automaton. Evaluating these regular expressions into rational functions yields the probability of reaching the target states. One limiting factor of this approach is that the complete regular expression has to be stored in memory.

Hahn et al. introduced [70] and implemented [69] a simplification and refinement of Daws' algorithm. Instead of using regular expressions, they stored rational functions directly. This has the advantage that possible simplifications of these functions, such as cancellation of common factors, can be applied on the fly. This allows memory to be saved. It also provides a more concise representation of the values computed to the user. The scope of the approach has also been extended from reachability, to additionally handle accumulated reward properties. Several works from RWTH Aachen have followed up on solution methods for parametric Markov chains [43, 88, 126]. This type of parametric model checking has been used in [7] to build a model-repair framework for stochastic systems and in [89, 90, 91] to reason about the robustness of robot controllers against sensor errors.

The technique presented here borrows some ideas from the work of Kwiatkowska et al. [102]. Their work considered MDPs that are labelled with parametric transition probabilities. The authors did not aim to compute a closed-form function that represents properties of a model, but rather to accelerate the computation of results for individual instantiations of parameter values. Rather than state elimination, they used value iteration and other methods to evaluate the model for certain parameter values. In doing so, they could for instance, re-use computations for different instantiations of parameters that only depend on the graph structure of the model that remains unchanged for different instantiations.

Inspiration is also taken from Forejt et al. [60], where the role of parameters is different. Forejt et al. described a policy iteration-based technique to evaluate parametric MDPs. While they also considered parameters in [60] that can influence the model structure, they would exploit similarities to inform the search for the policy when moving from one parameter value to the next. The repeatedly called model checking of Markov chains, on the other hand, is not parametric. The approach presented here is therefore completely orthogonal, as the focus is on the analysis of Markov chains. In more detail, Forejt et al. [60] used an incremental approach to find a good starting point for a policy iteration approach for MDPs. The intuition is that an optimal policy is likely to be good, if not optimal, on the shared part of an MDP that grows with a parameter. This approach has the potential to find the policy in less steps, as less noise disturbs the search in smaller MDPs, but its main promise is to provide an excellent oracle for a starting policy. Moreover, in the lucky instances where the policy is stable, it can also happen that there is a part of the Markov chain, obtained by using a policy that builds on a smaller parameter, that is outside of the cone of influence of the changes to the model. In this case, not only the policy, but also its evaluation is stable under the parameter change.

6.2 Algorithms

A novel algorithm that substantially reduces the cost of recomputation of the parametric reachability probability for a reconfigured PMC is now discussed. Recall that the goal is to re-use information when recalculating reachability for a reconfigured PMC. This can be achieved by choosing the order in which states are eliminated in the original PMC. The general idea is that if the set of states where structural changes might occur is known a priori, then state elimination can be applied to all other states first. States where structural changes might occur are called *volatile* states.

6.2.1 Definitions

Definition 28. *A* volatile *parametric Markov chain (VPMC),* $V = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{X}, \mathsf{Vol})$*, is a PMC augmented with a set* $\mathsf{Vol} \subseteq \mathcal{S}$ *of volatile states.*

An *elimination ordering* for a VPMC V is a bijection

$$\prec_V \colon \mathcal{S} \to \{1, \ldots, |\mathcal{S}|\}$$

that defines an ordering for the elimination of states in $\mathcal{S}$, such that $\prec_V(s) < \prec_V(s')$ holds for all $s \in \mathcal{S} \setminus \mathsf{Vol}, s' \in \mathsf{Vol}$, where $\prec_V(s) < \prec_V(s')$ indicates that s is eliminated before s'. Observe that a volatile state in V is only eliminated after all non-volatile states.

Definition 29. *A* reconfiguration *for a VPMC* $V = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{X}, \mathsf{Vol})$ *is a PMC* $V^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, \mathcal{X})$*, where* $\mathcal{S}^R$ *is a set of states with* $\{s_0\} \subset \mathcal{S}^R \cap \mathcal{S}$*,* s_0 *is the initial state, and* $\mathcal{X}$ *is the finite set of parameters for* V*. The reconfigured probability matrix* $\mathbf{P}^R$ *is a total function* $\mathbf{P}^R : \mathcal{S}^R \times \mathcal{S}^R \to \mathcal{Q}_\mathcal{X}$ *such that, for all* $s, s' \in \mathcal{S}^R$ *where* $\mathbf{P}(s, s')$ *is defined,* $\mathbf{P}(s, s') \neq \mathbf{P}^R(s, s')$ *implies* $s, s' \in \mathsf{Vol}$*.*

Given a VPMC V and a reconfiguration V^R for V, a state s in $\mathcal{S}$ is

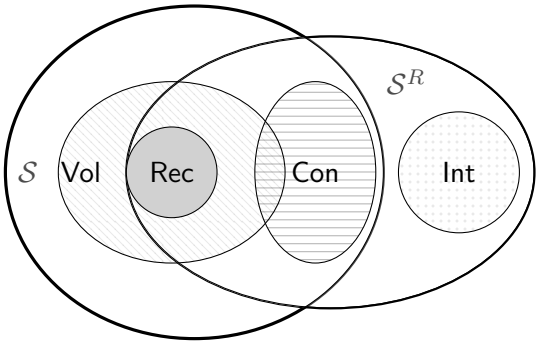


Figure 6.2: Venn diagram showing the consistent, reconfigured, and introduced states for a VPMC V and reconfiguration V^R.

- *consistent* in V^R if s is also in $\mathcal{S}^R$, and the set of all predecessors and successors of s remains unchanged in V^R (that is $\mathsf{pre}_V(s) = \mathsf{pre}_{V^R}(s)$, $\mathsf{post}_V(s) = \mathsf{post}_{V^R}(s)$, $\mathbf{P}(s_1, s) = \mathbf{P}^R(s_1, s)$ for every $s_1 \in \mathsf{pre}_V(s)$, and $\mathbf{P}(s, s_2) = \mathbf{P}^R(s, s_2)$ for every $s_2 \in \mathsf{post}_V(s)$),

- *reconfigured* in V^R if s is also in $\mathcal{S}^R$ and s is not consistent,

- *introduced* in V^R if s is neither consistent nor reconfigured.

The sets of all consistent, reconfigured, and introduced states are denoted by $\mathsf{Con}(V, V^R)$, $\mathsf{Rec}(V, V^R)$, and $\mathsf{Int}(V, V^R)$, respectively. Figure 6.2 shows the consistent, reconfigured, and introduced states for V and V^R.

Example 21. *Consider the Zeroconf models from Figures 6.1a and 6.1b. Let*

$$Z_k = (\mathcal{S}, i, \mathbf{P}, \mathcal{X}, \mathsf{Vol})$$

be a VPMC for $n = k$, such that

$$\begin{aligned}\mathcal{S} &= \{1, \ldots, k\} \cup \{s_0, \mathrm{i}, \mathrm{err}\},\\ \mathcal{X} &= \{p, q\},\\ \mathsf{Vol} &= \{\mathrm{i}, k\}.\end{aligned}$$

Now let

$$Z_{k+1} = (\mathcal{S}^R, s_0, \mathbf{P}^R, \mathcal{X})$$

be a reconfiguration for Z_k such that $\mathcal{S}^R = \mathcal{S} \cup \{k+1\}$. Then

$$\begin{aligned}\mathsf{Con}(Z_k, Z_{k+1}) &= \{1 \ldots k-1\} \cup \{s_0, \mathrm{err}\},\\ \mathsf{Rec}(Z_k, Z_{k+1}) &= \{k, i\},\\ \mathsf{Int}(Z_k, Z_{k+1}) &= \{k+1\}.\end{aligned}$$

6.2.2 Parametric reachability for VPMCs

Algorithm 6 was developed to compute the parametric reachability probability of some target state in a VPMC $V = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{X}, \mathsf{Vol})$ for a given elimination ordering for V, and is based on the state elimination algorithm described in Section 2.5. The procedure Preprocess is the same as that presented in Algorithm 1, where states from which Ω are unreachable are removed, and a new target state s_t is introduced to the model. Algorithm 6 computes a partial probability matrix $\mathbf{P}'$, initialised as a zero matrix, that stores the probability of reaching s_2 from s_1 via any eliminated non-volatile state, where s_1, s_2 are either volatile states, the initial state, or the target state. It also computes an *elimination*

Algorithm 6 Parametric reachability probability for VPMCs.

```
1: procedure VPMCParametricReachability(V, Ω, ≺_V)
2:     requires: a VPMC V = (S, s_0, P, X, Vol), a set of target states Ω ⊆ S, and an
   elimination ordering ≺_V for V.
3:     S, P ← Preprocess(V, Ω)
4:     Elim ← S \ {s_0, s_t}
5:     P' ← 0_{|S|,|S|}            // partial probability matrix
6:     m_V ← ∅                     // elimination map
7:     while Elim ≠ ∅ do
8:         s_e ← arg min ≺_V |_m
9:         for all (s_1, s_2) ∈ pre(s_e) × post(s_e) do
10:            p = P(s_1,s_e)P(s_e,s_2) / (1−P(s_e,s_e))
11:            if s_1 ∈ Vol ∪ {s_0, s_t} and s_2 ∈ Vol ∪ {s_0, s_t} then
12:                if s_e ∉ Vol then
13:                    P'(s_1, s_2) ← P'(s_1, s_2) + p
14:                else
15:                    m_V ← m_V ⊕ {(s_e, s_1, s_2) ↦ p}
16:                end if
17:            end if
18:            P(s_1, s_2) ← P(s_1, s_2) + p
19:        end for
20:        Eliminate(V, s_e)
21:        Elim ← Elim \ {s_e}
22:    end while
23:    return (P(s_0, s_t), P', m_V)
24: end procedure
```

map m_V, a function mapping tuples of the form (s_e, s_1, s_2), where s_e is an eliminated volatile state and s_1, s_2 are either volatile states, the initial state, or the target state, to the value computed during state elimination for the probability of reaching s_2 from s_1 via s_e. The only transitions of interest are those between volatile states, the initial state, or the target state, since all non-volatile states in any reconfiguration of V will be eliminated first. Computed values for transitions to or from these states therefore serve no purpose once they have been eliminated.

Algorithm 7 Parametric reachability probability for a reconfigured VMPC.

procedure ReconfiguredParametricReachability(V, V^R, Ω, $\prec_V$, $\mathbf{P}'$, m_V)
  requires: a VPMC $V = (\mathcal{S}, s_0, \mathbf{P}, \mathcal{X}, \mathsf{Vol})$, a reconfiguration $V^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, \mathcal{X})$ for V, a set of target states $\Omega \subseteq \mathcal{S}$, the elimination ordering $\prec_V$ for V, the partial probability matrix $\mathbf{P}'$, and the elimination map m_V.
  $\mathcal{S}, \mathbf{P} \leftarrow \mathsf{Preprocess}(V, \Omega)$
  $M \leftarrow (\mathsf{Vol} \cap \mathcal{S}^R) \cup \{s_0, s_t\}$
  $\mathsf{Elim} \leftarrow \mathsf{Con}(V, V^R) \setminus M$
  $\mathsf{Eliminate}(V^R, \mathsf{Elim})$
  $\mathsf{Elim} \leftarrow \mathsf{Vol} \cap \mathcal{S}^R$
  $\mathsf{Infected} \leftarrow \mathsf{Rec}(V, V^R)$
  $\mathbf{P}^R(s_0, s_t) = \mathbf{P}'(s_0, s_t)$
  while $\mathsf{Elim} \neq \emptyset$ **do**
    $s_e \leftarrow \arg\min \prec_V \restriction_{\mathsf{Elim}}$
    if $\mathsf{Infected} \cap \mathsf{Neigh}(s_e) = \emptyset$ **then**
      for all $(s'_e, s_1, s_2) \in \mathsf{Dom}(m_V) \restriction_{\{s_e\} \times M^2}$ **do**
        $\mathbf{P}'(s_1, s_2) \leftarrow \mathbf{P}'(s_1, s_2) + m_V(s'_e, s_1, s_2)$
      end for
      $\mathsf{Eliminate}(V^R, s_e)$
    else
      for all $\{(s_1, s_2) \in \mathcal{S}^R \times \mathcal{S}^R \mid s_1 = s_e \text{ or } s_2 = s_e\}$ **do**
        $\mathbf{P}^R(s_1, s_2) \leftarrow \mathbf{P}^R(s_1, s_2) + \mathbf{P}'(s_1, s_2)$
        $\mathbf{P}'(s_1, s_2) \leftarrow 0$
      end for
      $V^R \leftarrow \mathsf{StateElimination}(V^R, s_e)$
      $\mathsf{Infected} \leftarrow \mathsf{Infected} \cup \mathsf{Neigh}(s_e)$
    end if
    $\mathsf{Elim} \leftarrow \mathsf{Elim} \setminus \{s_e\}$
  end while
  for all $s_e \in \mathsf{Int}(V, V^R)$ **do** $V^R \leftarrow \mathsf{StateElimination}(V^R, s_e)$
  return $\mathbf{P}^R(s_0, s_t)$
end procedure

6.2.3 Parametric reachability for reconfigured VPMCs

Given a reconfiguration $V^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, \mathcal{X})$ for V, an elimination ordering for V, and the partial probability matrix and mapping computed using Algorithm 6, Algorithm 7 computes the parametric reachability probability for V^R as follows. Firstly the set of all non-volatile states of V and incident transitions are removed from V^R, though state elimination itself does not occur. A set of *infected* states is then initialised to be the set of all states that are reconfigured in V^R. Then, for every other remaining state that is not introduced in V^R, if that state or its neighbours are not infected this state is treated as a non-volatile state. That is, $\mathbf{P}'$ is updated with the corresponding values in m_V and the state and its incident transitions are removed without performing state elimination. If the state, or one of its neighbours, is infected then the probability matrix is updated such that all transitions to and from that state are augmented with the corresponding values in $\mathbf{P}'$. These entries are then removed from the mapping. Subsequently, state elimination (Algorithm 3 in Section 2.5) is applied, and the infected area is expanded to include the immediate neighbourhood of the eliminated state. Finally, state elimination is applied to the set of all remaining introduced states in V^R.

Example 22. *Consider again the Zeroconf models from Figures 6.1a and 6.1b, and Example 21. Recall that* $Z_k = (\mathcal{S}, i, \mathbf{P}, \mathcal{X}, \mathsf{Vol})$ *is a VPMC for* $n = k$*, with* $\mathcal{S} = \{1, \ldots, k\} \cup \{s_0, \mathrm{i}, \mathrm{err}\}$*,* $\mathcal{X} = \{p, q\}$*,* $\mathsf{Vol} = \{\mathrm{i}, k\}$*, and* $Z_{k+1} = (\mathcal{S}^R, s_0, \mathbf{P}^R, \mathcal{X})$ *is a reconfiguration for* Z_k *with* $\mathcal{S}^R = \mathcal{S} \cup \{k+1\}$*, and where*

$$\begin{aligned}
\mathsf{Con}(Z_k, Z_{k+1}) &= \{1 \ldots k-1\} \cup \{s_0, \mathrm{err}\},\\
\mathsf{Rec}(Z_k, Z_{k+1}) &= \{k, i\},\\
\mathsf{Int}(Z_k, Z_{k+1}) &= \{k+1\}.
\end{aligned}$$

The property of interest is the parametric reachability probability of the state err. *Note that preprocessing removes the state* ok *from* Z_k *since* $\mathsf{reach}(\mathrm{ok}, \mathrm{err})$ *does not hold. As there is only one target state, namely* err*, without loss of generality the introduction of a new target state* s_t *is simply omitted here, and the same result can be obtained by eliminating all states except the initial state* i *and the target state* err*. Now define an elimination ordering for* Z_k

$$\prec_{Z_k} = \{1 \mapsto 1, 2 \mapsto 2, \ldots, k \mapsto k, \mathrm{i} \mapsto k+1\}.$$

State elimination then proceeds according to $\prec_{Z_k}$*, and after the first* $k-1$ *states have been eliminated,*

$$\begin{aligned} \mathbf{P}'(k, \mathrm{err}) &= p^k, \\ \mathbf{P}'(k, \mathrm{i}) &= \sum_{j=1}^{k-1} (p^j - p^{j+1}). \end{aligned}$$

Eliminating the remaining volatile states k *and* i *then yields*

$$m_{Z_k} = \left\{ \begin{array}{rl} (k, \mathrm{i}, \mathrm{err}) \mapsto & qp^k, \\ (k, \mathrm{i}, \mathrm{i}) \mapsto & q(1-p^k), \\ (\mathrm{i}, s_0, \mathrm{err}) \mapsto & \frac{qp^k}{1-q(1-p^k)} \end{array} \right\}.$$

First, all states $1, \ldots, k-1$ *and their incident transitions are simply eliminated from* Z_{k+1}*, and the infected set is initialised to be* $\{k, i\}$*. Since* k *is already infected the probability*

matrix is updated as follows,

$$\begin{aligned}
\mathbf{P}^R(k, \mathrm{err}) \leftarrow\ & \mathbf{P}^R(k, \mathrm{err}) + \mathbf{P}'(k, \mathrm{err}) \\
&= 0 + p^k \\
&= p^k, \\
\mathbf{P}^R(k, \mathrm{i}) \leftarrow\ & \mathbf{P}^R(k, \mathrm{i}) + \mathbf{P}'(k, \mathrm{i}) \\
&= (1 - p) + \sum_{j=1}^{k-1} (p^j - p^{j+1}) \\
&= \sum_{j=0}^{k-1} (p^j - p^{j+1}) \\
&= 1 - p^k.
\end{aligned}$$

State elimination is then applied to state k and the corresponding entries in $\mathbf{P}'$ are set to zero. The state of the model after this step is shown in Figure 6.3a. State i *is also infected, but this time there are no corresponding non-zero values in $\mathbf{P}'$. State elimination is then applied to state* i *resulting in the model shown in Figure 6.3b. Finally, state elimination is applied to the single introduced state $k + 1$, resulting in the model shown in Figure 6.3c, and the algorithm terminates.*

6.2.4 Extension to parametric Markov reward models

The algorithms can also be extended to PMCs annotated with rewards. The notion of volatility is extended to PMRMs as follows. A state is volatile if structural changes might occur in that state *or* if the reward labelling that state or an adjacent transition might change. The constructions are straightforward. Algorithms 2 to 7 are extended to incorporate rewards. For Algorithm 2, in addition to updating the probability matrix for the

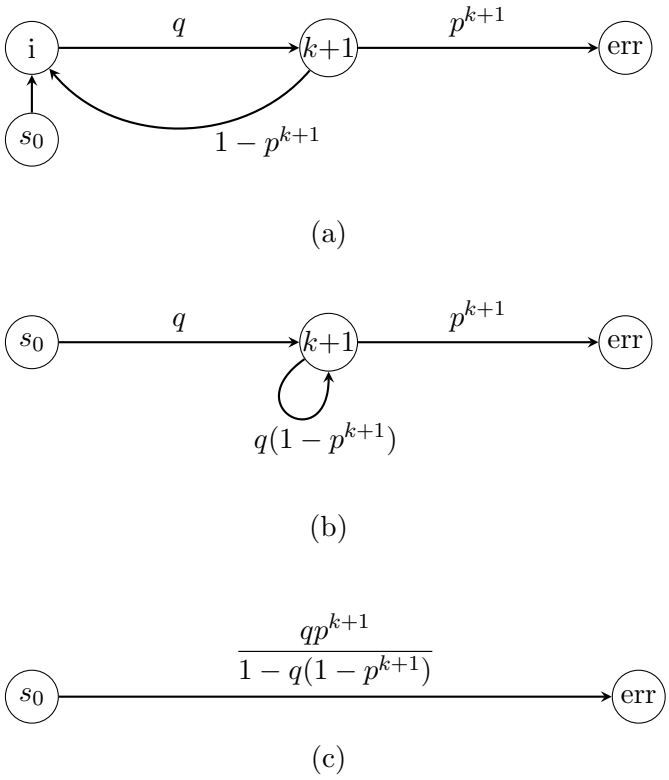


Figure 6.3: Z_{k+1} after the elimination of states k (a), i (b), and $k+1$ (c).

elimination of some state s_e, the state rewards are updated as follows,

$$\mathbf{R}(s_1) \leftarrow \mathbf{R}(s_1) + \mathbf{P}(s_1, s_e)\frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)}\mathbf{R}(s_e).$$

The updated value for $\mathbf{R}(s_1)$ reflects the reward that would be accumulated if a transition would be taken from s_1 to s_e, where the expected number of self transitions would be

$$\frac{\mathbf{P}(s_e, s_e)}{1 - \mathbf{P}(s_e, s_e)}.$$

Algorithm 6 then constructs additional mappings to record these computed expected reward values, which are then used for reconfiguration in Algorithm 7.

6.3 Case studies

A prototypical implementation of the technique was developed, and a metric that can be used for the evaluation of different models can be defined as the total number of arithmetic operations performed for the elimination of all states in a model. The implementation serves only to illustrate the potential of the method.

Two classes of models are analysed. Firstly, the family of Zeroconf protocols described in the introduction to this chapter are discussed, and secondly the family of synchronisation models presented in Chapters 4 and 5.

The prototype tool used for the analysis takes as input a set of model states and a set of model transitions. The underlying graphs of Zeroconf models have simple structures, and are programmatically generated by the tool itself. The underlying graphs of the synchronisation models have more complex structures. A modified version of the script described in Section 4.4 is used, where given a set of parameters defining the model the sets of all sets and transitions of the model are generated, and are written to files in a format that is widely accepted by probabilistic model checking tools, as illustrated in Figure 6.4.

6.3.1 Zeroconf

The property of interest is the reachability of the error state for the family of Zeroconf models, parametrised in the number n of attempts, after which the protocol will (potentially incorrectly) assume that it has selected an unused address. The initial model for $n = 1$ is defined, its volatile region is determined as in Example 22, and Algorithm 6 is applied. At each step we increment n and apply Algorithm 7 to the model. The set of volatile states

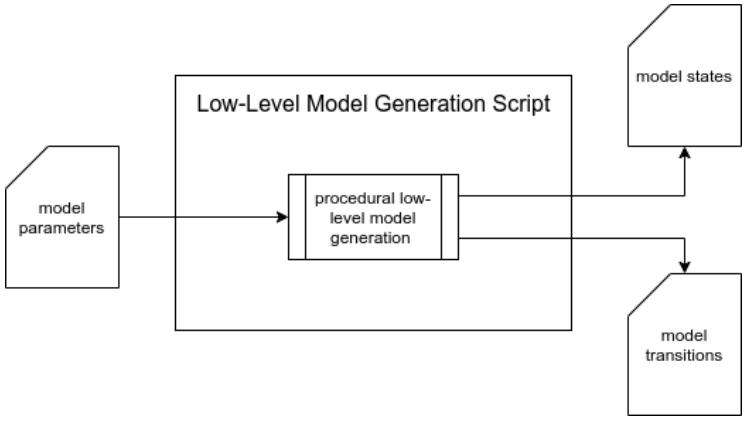


Figure 6.4: The automation of low-level model generation for synchronisation models, given a set of model parameters.

can always be identified at each step as the state i and the state introduced in the previous step.

Figures 6.5a and 6.5b show the total number of performed arithmetic operations accumulated during the incremental analysis of the models and the per instance and cumulative ratios of the number of arithmetic operations performed for regular state elimination, respectively. Each entry $n = x$ corresponds to the incremental step from $n = x$ to $n = x + 1$. This ratio shows the small share of the number of iterations required when the values are calculated for a range of parameters using reconfiguration (repeated applications of Algorithm 7), when compared to the naive approach to re-calculate all values from scratch (applying Algorithm 6).

Figure 6.5a shows that the total number of operations is approximately quadratic in the parameter when regular state elimination (applying Algorithm 6) is repeatedly applied from scratch. This is a consequence of the number of operations for *each* parameter being linear in the parameter value when naively applying Algorithm 6. This is in stark contrast to the number of operations needed when the parameter is stepwise incremented

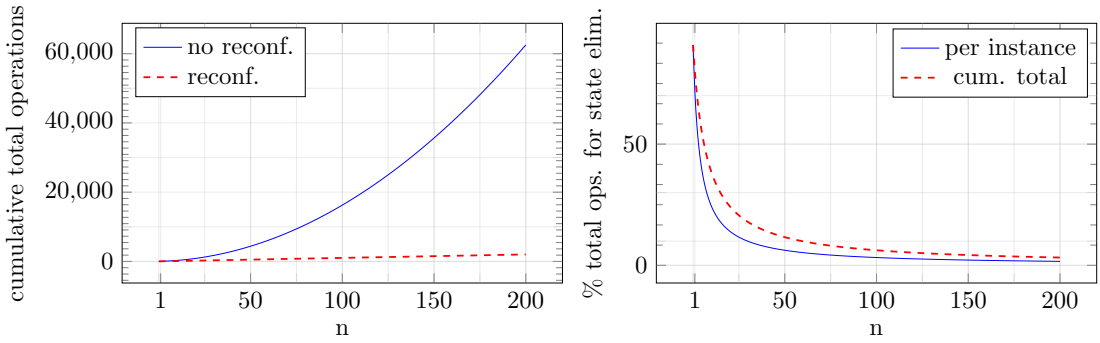


Figure 6.5: Cumulative total of arithmetic operations performed for iterative analysis of Zeroconf for $n = 1 \ldots 200$ (a), and the per instance/cumulative ratios of total operations for reconfiguration to total operations for regular state elimination, given as a percentage (b).

using Algorithm 7, in each case capitalising on the analysis of the respective predecessor model. Here the update cost is constant, since the extent of structural change at each step is constant. This leads to dramatic savings (quadratic vs. linear) when exploring the parameter space, as illustrated by Figure 6.5b.

6.3.2 Oscillator synchronisation

The models described in Chapters 4 and 5, for the clock synchronisation of nodes in a network, are now revisited. Recall that consensus on clock values emerges from interactions between the nodes. The underlying mathematical model is that of pulse-coupled oscillators. This family of models is parametric in the number N of nodes that form the network; the granularity T of the discretisation of the oscillation cycle; the length R of the refractory period, during which nodes ignores interactions with their neighbours; the strength ϵ of the coupling between the oscillators; and finally the likelihood μ of any individual interaction between two nodes not occurring due to some external factor. Each state of the model corresponds to some global configuration for the network—a vector encoding the size of

node clusters that share the same progress through their oscillation cycle. The target states of interest are those in which all nodes share the same progression through their cycle and are therefore *synchronised*.

Changing the parameters N and T redefines the encoding of a global network state. This results in drastic changes to the structure of the model and therefore makes it hard to identify volatile states. The prototypical implementation only considers low-level models defined explicitly as a set of states and a transition matrix, which trivialises the identification of volatile areas. Future implementation into a model checker, however, might allow volatile states to be clearly identified by analysing the guards present in high-level model description languages like the one described in Section 2.7.1.

Changing the parameter ϵ results in such significant changes in the structure of the model and it is not clear how the synergistic effects that have been observed can be ported to analysing its parameter space, while changing μ does not change the structure of the underlying graph and hence is not of interest in this chapter.

Here, the focus is on the incremental analysis for the parameter R. Arbitrarily, N is fixed to 5 and ϵ is fixed to 0.1, and the incremental analysis is repeated for four different values for T. The parameter R varies from 1 to T (for each of the different values of T considered).

Figures 6.6 and 6.7 show the cumulative total of performed arithmetic operations and the per instance and cumulative ratios of the totals for regular state elimination to the totals for reconfiguration given as a percentage, respectively. Each entry $R = x$ corresponds to the incremental step from $R = x$ to $R = x + 1$.

The effectiveness of the approach increases slightly as T increases. Incrementing R for models with a higher value for T results in greater structural changes than for a model with a lower value for T. This is because an increment to R of 1 corresponds to an increase

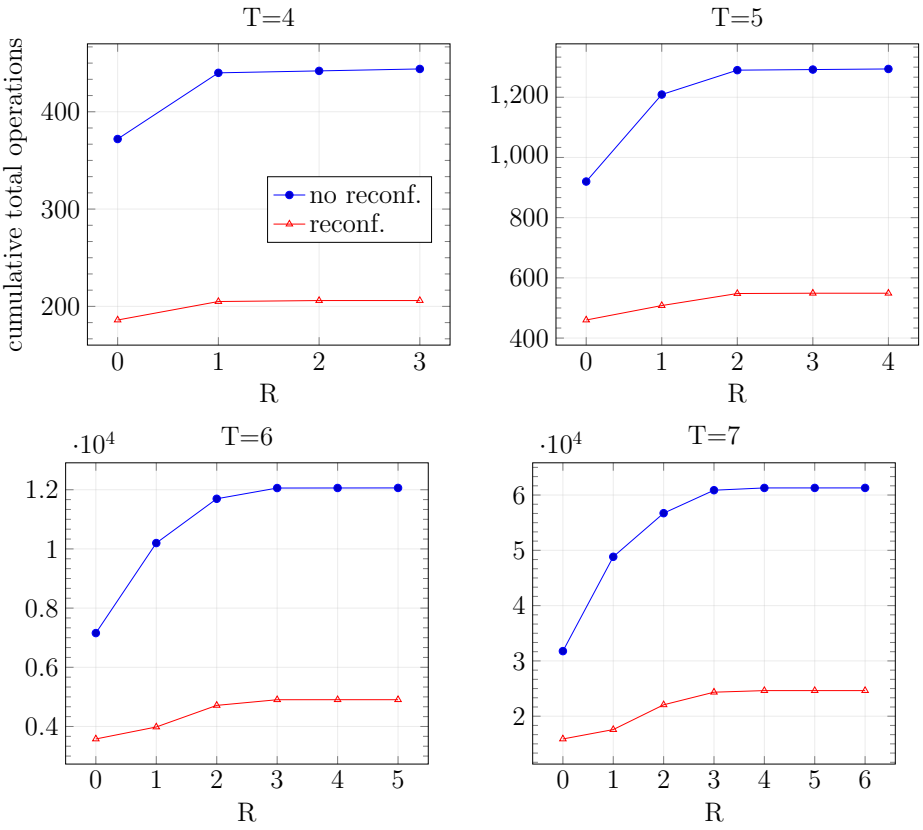


Figure 6.6: Cumulative total of arithmetic operations performed for iterative analysis of synchronisation models with respect to R.

of $\frac{1}{4}$ in the length of the refractory period (in oscillation cycles), and therefore a larger number of states and transitions are affected than when such an increment corresponds to an increase of only $\frac{1}{7}$ when $T = 7$. Increasing R up to half of the length of a cycle increases the amount of structural change, and consequentially a decrease in the amount of information that can be reused from the previous analysis at each step, while increasing R past half the length of a cycle decreases the structural changes. The increase is because the low value for the coupling strength ($\epsilon = 0.1$) results in very small perturbations being lost due to rounding, and hence the refractory period has a very minor effect when it is

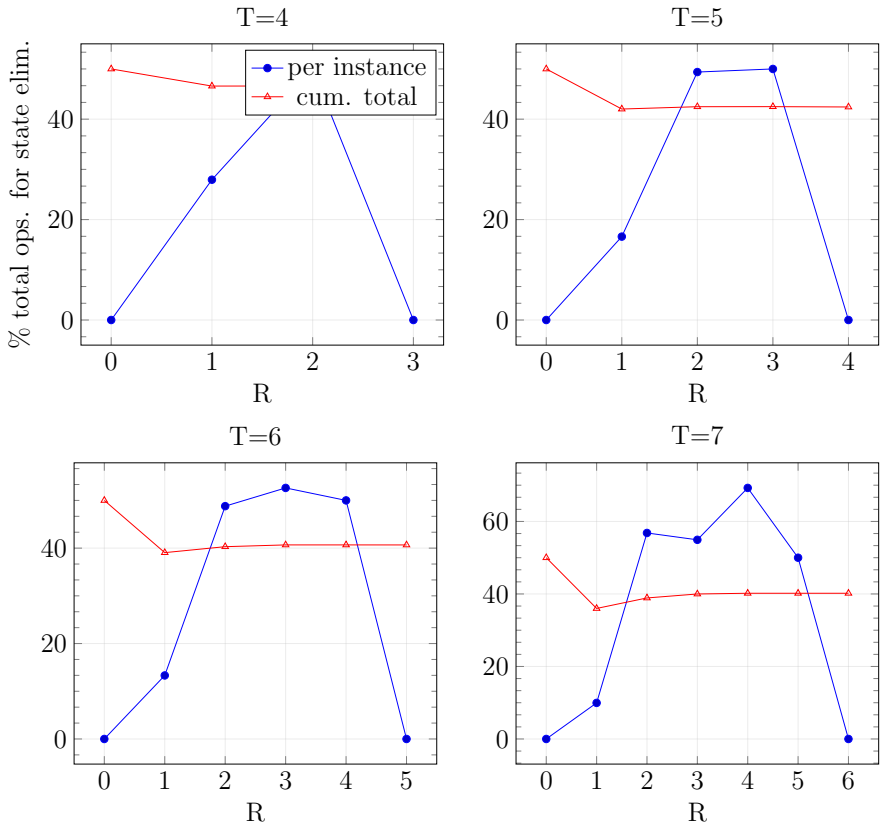


Figure 6.7: Per instance and cumulative ratios of total operations for reconfiguration to totals for regular state elimination given as a percentage.

short. In the extreme case no structural changes occur at all in the step from $R = 0$ to $R = 1$ for all instances of T, and all of the information from the analysis of the initial instance $R = 0$ can be reused. The decrease is because as the length of the refractory period increases past half of a cycle nearly all interactions between oscillators are lost, as was shown in Figure 4.5. Eventually increasing the parameter has no effect and again all of the information from the analysis can be reused, namely for the incremental step from $R = T - 1$ to $R = T$. While for many individual instances the cost of incremental analysis is low, the initial cost of constructing the first instance of the model cannot be avoided, as

illustrated by the cumulative ratios.

Overall it is clear that, while still substantial, the gains here are not as pronounced as those seen for the analysis of the Zeroconf protocol. This is to be expected, since the structural changes induced by changing the parameter R are not constant for each iteration.

6.4 Conclusion and future work

It is clear that the approach presented here works well for structured Markov chains, such as chain-, ring-, or tree-like structures. The Zeroconf experiments evidence this by showing that where the cost of model-checking an individual model grows linearly with a parameter, model checking up to a parameter becomes linear in the maximal parameter considered, whereas the overall costs grew quadratically if all models were considered individually. Therefore, significant gains would be expected whenever changes can be localised and isolated. Moreover, it is likely that this would be the norm, rather than the exception, since chains, rings, and trees are common structures in models. For an analyst constructing a specialised structure corresponding to some system of interest, it may be the case that a closed form solution exists for the analysis of that structure. However, if the structure of the model is slightly different, or if the analyst either fails to identify the existing solution, or, knows it exists yet does not want to invest the time to research it, then a naive analysis of the model needs to be conducted. The technique presented here provides a sweet spot between these extremes: the speed is close to that for evaluating closed form solutions, but applying the method does not put any burden on the analyst who creates the parametrised model.

The limitation of this approach is that much of the advantage is lost when a change in a parameter induces severe structural changes in the model. For the synchronisation

protocols presented in Chapters 4 and 5, some model parameters changed the structure of the model. For the parameters N, the number of nodes in the network, and T, the granularity of the discretisation of the oscillation cycle, it is unclear how any similarity at all can be identified between the two structures obtained when moving from one parameter instantiation to the next by incrementing (or decrementing) one of the parameters, since the encoding of a global state of the network is redefined for each instance. Incrementing the parameter ϵ results in new models where the encoding of a state remains the same, but where changes to the transitions in the underlying graph of the model appear sporadically throughout the structure. Here the density of the graphs (observe for example, the initial fan-out from the initial state to an initial configuration for the oscillators, as described in Section 4.4) results in such structural changes having a huge cone of influence. In the worst case, for example a fully-connected graph, a cubic overhead would be incurred [69].

Chapter 7

Accelerated Model Checking of Parametric Markov Chains

As discussed in Chapter 1, parametric Markov chains occur quite naturally in various applications: they can be used for conservative analyses of probabilistic systems, to find optimal settings for parameters, to visualise the influence of parameters, or to facilitate the adjustment of a system in case parameters change. Unfortunately, these advancements come at a cost: parametric model checking is very slow. When an analysis of a large family of parametric probabilistic models is conducted, as was the case in Chapters 3 through 5, it is often the case that either the parametric analysis takes a substantial amount of time, or is not feasible at all, when the models are quite large. The analysis of families of models is then reduced to instantiating the models for some set of parameter values, obtaining results, and then re-instantiating the model when the parameters change.

A wide variety of existing techniques and tools can be used for the analysis of parametric Markov chains. However, when analysing thousands of parameter instantiations their analysis is often slow. The previous chapter investigated how the parametric analysis of

such families of models could be accelerated, when the parameters changing the model structure are changed. In this chapter, new ideas are introduced for accelerating the analysis when changes to model parameters do not result in changes in the underlying graph; for instance when investigating the effect of environmental effects on communication (broadcast failure) in Chapters 4 and 5.

As described in Section 2.5, the application of parametric model checking yields a rational function corresponding to some simple temporal property or expected reward, that can be evaluated for different parameter instances. Traditionally, co-prime polynomial nominator-denominator representations have been used to represent rational functions. During the analysis, it is often the case that there are few large common factors that could be cancelled to simplify the function. This can result in the representation growing in size and slowing down the analysis. This work proposes the use of arithmetic circuits (ACs) to represent these functions. The arithmetic circuits are essentially directed acyclic graphs (DAGs). Using ACs ensures that expensive methods are not required to cancel out common factors, and common sub-expressions can be shared by different formulae, greatly reducing the overall memory footprint.

The AC representation introduced here was integrated into the probabilistic model checker ePMC. As was the case in the previous chapter, the backbone of the verification technique used here was the state elimination algorithm of Hahn [70]. The extended tool was then used to analyse a range of case studies, including the synchronisation models introduced in Chapters 4 and 5. A speed-up of a hefty factor of 20 to 120 when compared to storing functions in terms of co-prime numerator and denominator polynomials.

In section 7.2 it is shown how ACs can be used for the representation of rational functions, and are constructed during the successive elimination of states from the model. In Section 7.3, the approach is evaluated on a range of benchmarks, and the chapter is

concluded in Section 7.4.

7.1 Related work

Daws [41] devised a language-theoretic approach for the parametric model checking of discrete-time Markov chains. In this approach, the transition probabilities are considered as letters of an alphabet, and the model can be viewed as a finite automaton. Then, based on the state elimination method [84], a regular expression that describes the language of such an automaton is calculated. In a post-processing step, this regular expression is recursively evaluated, resulting in a rational function over the parameters of the model. This approach was extended and tuned in [70] so as to operate with rational functions, which are stored as co-prime numerator and denominator polynomials, rather than with regular expressions.

The process of computing a function that describes properties like reachability probabilities or long-run average rewards that depend on model parameters is often costly. However, once the function has been obtained, it can very efficiently be evaluated for given parameter instantiations. Because of this, parametric model checking of Markov models has also attracted attention in the area of runtime verification, where the acceptable time to obtain values is limited. Calinescu et al.[28] showed how probabilistic verification can extend its operation to runtime, where system parameters change and new verification results are needed immediately. The transition of probabilistic verification to efficiently address runtime scenarios was left as an open problem. In [59] Filieri et al. worked towards addressing this problem by proposing a new method to compute closed form solutions for reachability properties of DTMCs. They obtained speed ups that were equivalent to the results presented in [70], however suggestions for how this could be further improved were given.

In [126], Quatmann et al. proposed a new technique for the analysis of PMCs, where the model is transformed into a parameter-free Markov decision process that can be solved efficiently using existing techniques. While their solution is scalable and fast, it does not give a closed form solution. Other works in the area are centred around deciding the validity of boolean formulas depending on the parameter range using SMT solvers or extending these techniques to models that involve nondeterminism [44, 68, 40].

7.2 Representing formulas as arithmetic circuits

In existing tools for parametric model checking of Markov models, rational functions have traditionally been represented in the form presented in Section 2.2:

$$q(x_1, \ldots, x_n) = \frac{\rho_1(x_1, \ldots, x_n)}{\rho_2(x_1, \ldots, x_n)},$$

where $\rho_1(x_1, \ldots, x_n)$ and $\rho_2(x_1, \ldots, x_n)$ are co-prime polynomials, where each polynomial is represented as a list of monomials [69, 45, 101]. As a result, for some cases the representations of such functions are very short. Often, during the state elimination phase, large common factors can be cancelled out, such that one can operate with relatively small functions throughout the whole algorithm. There are, however, many cases without, or with very few, large common factors. The nominator-denominator representations then become larger and larger during the analysis. In this case, the analysis is slowed down severely, mostly by the time taken for the cancellation of common factors. Cancelling out such factors is non-trivial, and indeed a research area in itself. In addition, if formulas become large, this can also lead to out-of-memory problems. To overcome this issue, the representation of rational functions by arithmetic circuits (ACs) is proposed. These ACs are directed acyclic graphs (DAGs).

7.2.1 Arithmetic circuit representation

Definition 30. *An* arithmetic circuit (AC) *is a tuple* $\mathcal{A} = (N, \mathcal{X}, L)$ *where*

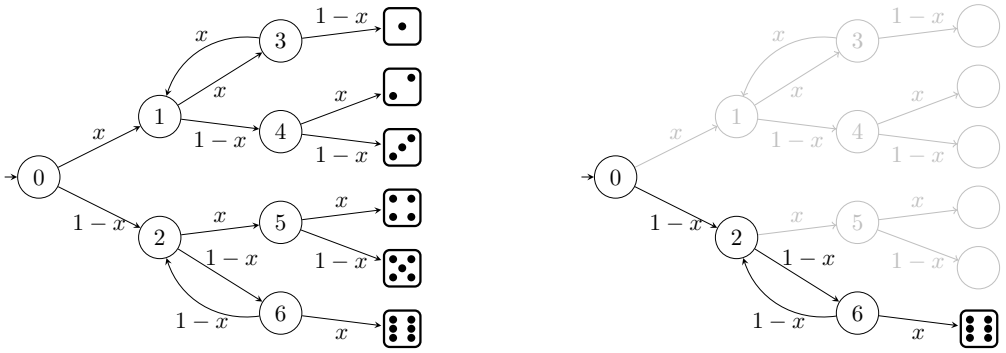
- $N \in \mathbb{N}$ *is the* size *of the AC,*
- $\mathcal{X}$ *is the finite set of* parameters*,*
- $\mathcal{L}_{\mathcal{A}} = \mathbb{Q} \cup \mathcal{X} \cup (\{+, \cdot\} \times \mathbb{N} \times \mathbb{N}) \cup (\{-, /\} \times \mathbb{N})$ *is the set of labels for the AC,*
- $L : \{0, \ldots, N-1\} \to \mathcal{L}_{\mathcal{A}}$ *is the labelling function.*

It is required that for any $k \in \{0, \ldots, N-1\}$ *if* $L(k) = (\bowtie, n, m)$ *then* $n < k$ *and* $m < k$*, and if* $L(k) = (\bowtie, n)$ *then* $n < k$*, where* $\bowtie \in \{+, \cdot, -, /\}$*. An* $n \in \{0, \ldots, N-1\}$ *is called a* node index *and* $L(n)$ *is called a* node*. Any node* $L(n) \in \mathbb{Q} \cup \mathcal{X}$ *is called a* terminal node*. Accordingly, a node* $L(n) \in (\{+, \cdot\} \times \mathbb{N} \times \mathbb{N}) \cup (\{-, /\} \times \mathbb{N})$ *is called a* non-terminal *node. An AC is* reduced *if* $L(n) \neq L(m)$ *for all* n, m *with* $0 \leqslant n < m < N$*.*

Terminal nodes are either numbers or parameters, with non-terminal nodes describing operators to be applied on nodes. The restrictions of the non-terminal nodes are used to ensure that the AC forms a DAG by enforcing acyclicity. Terminal nodes that are numbers are restricted to $\mathbb{Q}$ rather than $\mathbb{R}$, because the AC should be represented by a computer. If an AC is reduced it contains no duplicate nodes. This is usually a desirable property, because duplicate nodes are equivalent and lead to an unnecessarily large DAG.

Definition 31. *The function represented by a node index* k *of an AC* $\mathcal{A} = (N, \mathcal{X}, L)$ *is defined as* $f_{\mathcal{A},k} \colon \mathbb{Q}^{\mathcal{X}} \to \mathbb{Q}$*, where* $\mathbb{Q}^{\mathcal{X}}$ *is the set of all functions from* $\mathcal{X}$ *into* $\mathbb{Q}$*, and for* $\upsilon \in \mathbb{Q}^{\mathcal{X}}$

- *if* $L(k) \in \mathbb{Q}$ *then* $f_{\mathcal{A},k}(\cdot) = L(k)$*,*
- *if* $L(k) \in \mathcal{X}$ *then* $f_{\mathcal{A},k}(\upsilon) = \upsilon(L(k))$*,*



(a) (b)

Figure 7.1: Parametric model of a dice simulated using a biased coin (a), and the model after preprocessing with respect to the parametric reachability probability of ⚅ (b).

- *if* $L(k) = (+, n, m)$ *then* $f_{\mathcal{A},k}(\upsilon) = f_{\mathcal{A},n}(\upsilon) + f_{\mathcal{A},m}(\upsilon)$,
- *if* $L(k) = (\cdot, n, m)$ *then* $f_{\mathcal{A},k}(\upsilon) = f_{\mathcal{A},n}(\upsilon) \cdot f_{\mathcal{A},m}(\upsilon)$,
- *if* $L(k) = (-, n)$ *then* $f_{\mathcal{A},k}(\upsilon) = -f_{\mathcal{A},n}(\upsilon)$,
- *if* $L(k) = (/, n)$ *then* $f_{\mathcal{A},k}(\upsilon) = \frac{1}{f_{\mathcal{A},n}(\upsilon)}$.

Example 23. *Figure 7.1a shows the Knuth and Yao [94] model of a six-sided dice, simulated by repeatedly tossing a coin. Here the model has been extended to simulate the tossing of a biased coin, where with probability* x *the coin toss results in heads, and with probability* $1 - x$ *the coin toss results in tails. The model is a PMC* $\mathcal{D}_\mathcal{X}$ *with states* $\mathcal{S} = \{0 \dots 6\} \cup \{⚀ \dots ⚅\}$ *and parameters* $\mathcal{X} = \{x\}$*. The property of interest is* $\mathsf{Reach}(\mathcal{D}_\mathcal{X}, \{⚅\})$*, the probability that the final result is* ⚅*. Recall from Section 2.5 that as a preprocessing step for state elimination, all states from which the target state is unreachable are removed from the model. Without loss of generality, the introduction of a new target state is simply omitted here, and the same result can be obtained by eliminating all states except the initial state* 0 *and the target state* ⚅*. The preprocessed model is shown in Figure 7.1b. Figure 7.2*

then shows the stepwise elimination of states from the model (left) and corresponding entries in the AC (right). Initially (Figure 7.2a), a terminal node is introduced to the AC for the single parameter x. Next, new nodes are introduced to the AC to represent the expression $1-x$ (Figure 7.2b). A new terminal node is introduced for the constant 1, and the expression $1-x$ is then represented by the AC node annotated with I*, as the sum of the constant 1 and the additive inverse of x. Figure 7.2c then shows the model and AC after the elimination of state 6. The introduced AC nodes annotated with* I *and* II *correspond to the new expressions $(1-x)^2$ and $(1-x)x$, respectively. Finally, state 2 is eliminated in Figure 7.2d, and the introduced AC nodes annotated with* I-IV *correspond to the following expressions*

$$\text{I: } 1-(1-x)^2$$
$$\text{II: } \frac{1}{1-(1-x)^2}$$
$$\text{III: } \frac{(1-x)x}{1-(1-x)^2}$$
$$\text{IV: } \frac{(1-x)(1-x)x}{1-(1-x)^2}$$

and $\mathsf{Reach}(\mathcal{D}_{\mathcal{X}}, \{\boxed{8}\})$ *is given by the expression represented by AC node* IV.

7.2.2 Simplifications

When operating with arithmetic circuits, there are a number of ways to reduce their memory footprint at the expense of higher running times. The simplest one is that, while creating a new node to represent a function, it might be the case that a node already exists with exactly the same operator and operand. In this case, it is better to drop the newly created node and use a reference to the existing node to counter the growth of the AC. Another optimisation is to use simple algebraic equivalences. This includes computing the

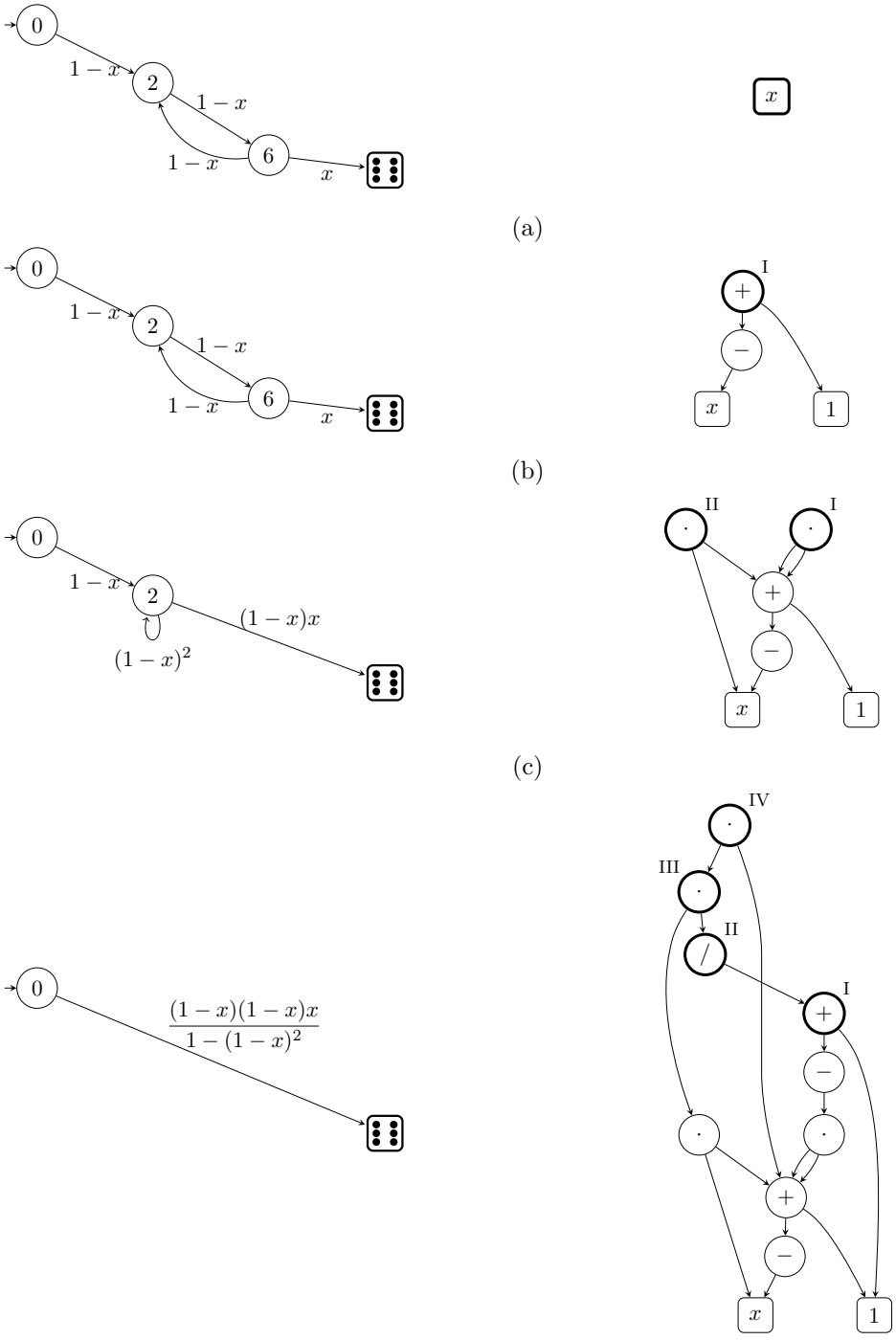


Figure 7.2: Construction of the AC for the parametric reachability probability of ⚄.

values of constant functions. For example, instead of creating a node representing $2 + 3$ a new terminal node can be introduced with the label 5, and if a new node needs to be introduced for the expression $y + x$, but a node already exists for $x + y$, then this node can be reused instead. In addition, additive and multiplicative neutral elements can be identified (rather than creating a new node for $0 + x$, the node for x is reused, and so forth). A further optimisation is to define a set of sample points for the parameters, and then evaluate these points for each function (internal node) represented in the AC. If the result of this evaluation is the same for two nodes then these nodes might represent the same function, and the AC can be simplified accordingly. The probability that two functions are mistakenly identified as being equivalent can be bound by using the Schwartz-Zippel Lemma [133, 152, 46].

7.2.3 Applications

Arithmetic circuits sometimes become very large, consisting of millions of nodes. Therefore, they cannot serve as a concise, human-readable description of the analysis result. However, compared to performing a non-parametric analysis, it is still beneficial to obtain a function representation in this form. Even for large ACs, evaluating parameter instantiations is very fast, and linear in the number of AC nodes. This is particularly useful if a large number of points need to be sampled, for instance when plotting a graph. In this case, results can be obtained much faster than using non-parametric model checking, as demonstrated in Section 7.3. In particular, for any instantiation values can be obtained in the same predictable time. This is quite in contrast to value iteration (see Section 2.4.2), where the number of iterations required to obtain a certain precision varies with the concrete values of parameters.

For this reason, parametric model checking is particularly useful for online model check-

ing or runtime verification [28]. Here, one can precompute the AC before running the actual system, while concrete values can be instantiated at runtime, with a running time that can be precisely calculated offline. Using arithmetic circuits expands the range of systems for which this method is applicable.

7.3 Experiments

Four case studies now illustrate the efficiency and scalability of the approach. Three models [79, 86, 127] are taken from the PRISM benchmark suite[1], and the last considers the synchronisation models of Chapters 4 and 5. All experiments were conducted on a PC with an Intel Core i7-2600 (tm) processor at 3.4GHz, equipped with 16GB of RAM, and running Ubuntu 16.04. For each case study a comparison is made between the performance times obtained for model analysis when using the parametric engine of the model checker ePMC [71], using either polynomial fractions or ACs to represent the functions corresponding to transition probabilities and state rewards. Basically, the AC is implemented as an array of 64-bit integers. Functions are represented as indices to this array. 4 bits describe the type of the node. For terminal nodes, the remaining bits denote the parameter or number used. For non-terminal nodes, 2×30 bits are used to refer to the operands within the AC. The results are also compared to those obtained using the parametric engine of PRISM, and the parametric and sampling engines of Storm.

Given a parametric model, and a set of valuations for its parameters, we are interested in the total time taken to check some property of interest for every valuation for the parameters. Since the primary concern is the efficiency of multiple evaluations of an existing model, model construction times are omitted, and the analysis is restricted to the total time taken for the evaluation of all parameter valuations. For the parametric engines of ePMC,

[1] `http://www.prismmodelchecker.org/benchmarks/`

PRISM, and Storm, the total time taken for both state elimination and the evaluation of the resulting function for all parameter valuations is recorded. For the sampling engine of Storm, we record the total time taken for value iteration. To determine convergence the precision is set to 10^{-10} rather than the default of 10^{-6}. This had a very minor influence on the runtime, and allowed a better comparison to ePMC, the results of which have a precision of $< 10^{-13}$.

In the following tables the following terminology is used: ePMC and PRISM denote the parametric engines of ePMC and PRISM, respectively. The parametric and sparse engines of Storm are denoted by Storm(P) and Storm(S), respectively. The parametric engine of ePMC using ACs is denoted by ePMC(D), and the engine using simplified ACs is denoted by ePMC(DS). All performance time listed in the tables are given in seconds.

7.3.1 Crowds protocol

The Crowds protocol [127] provides anonymity for a crowd consisting of N Internet users, of whom M are dishonest, by hiding their communication via random routing, where there are R different path reformulates. The model is a PMC parametrised by $B = \frac{M}{M+N}$, the probability that a member of the crowd is untrustworthy, and P, the probability that a member sends a package to a randomly selected receiver. With probability $1-P$ the packet is directly delivered to the receiver. The property of interest is the probability that the untrustworthy members observe the sender more than they observe others.

Table 7.1 shows the performance statistics for different values of N and R, where each entry shows the total time taken to check all pairwise combinations of values for B, P taken from $\{0.002, 0.004, \ldots, 0.998\}$. There is a substantial increase in the performance of ePMC when using non-simplified ACs (ePMC(D)), and using ACs (ePMC(DS)) that have been simplified by evaluating random points (see Section 7.2), instead of polynomial fractions

N	R	States	Trans.	PRISM	ePMC	ePMC(D)	ePMC(DS)	Storm(P)	Storm(S)
5	3	1198	2038	722	737	13	13	681	26
5	5	8653	14953	745	806	15	15	723	64
5	7	37291	65011	818	900	19	17	735	153
10	3	6563	15143	732	771	15	14	690	26
10	5	111294	261444	1146	910	23	16	712	63
10	7	990601	2351961	–T–	–T–	103	42	737	159
15	3	19228	55948	761	825	16	16	703	26
15	5	592060	1754860	–T–	–M–	42	28	709	64
15	7	8968096	26875216	–M–	–M–	–M–	–M–	777	174
20	3	42318	148578	814	805	15	14	709	26
20	5	2061951	7374951	–M–	–M–	108	90	720	67

Table 7.1: Performance statistics for the crowds protocol.

(ePMC) to represent functions. Here, ePMC with AC representations clearly outperforms the parametric engines of both PRISM and Storm. In some instances, ePMC turns out to be the fastest choice, while the sampling engine of Storm proves to be faster for other instances. Processes that exceeded the time limit of one hour are indicated by –T–, and processes that ran out of memory are indicated by –M–. In Figure. 7.3 the results for $N = 5$ and $R = 7$ are plotted.

7.3.2 Bounded retransmission protocol

The bounded retransmission protocol [79] divides a file, which is to be transmitted, into N chunks. For each chunk, there are at most MAX retransmissions over two lossy channels K and L that send data and acknowledgements, respectively. The model is a PMC parametrised by pK and pL, the reliability of the channels. The property of interest is the probability that the sender reports an unsuccessful transmission after more than 8 chunks have been sent successfully.

The performance statistics for different values of N and MAX are shown in Table 7.2,

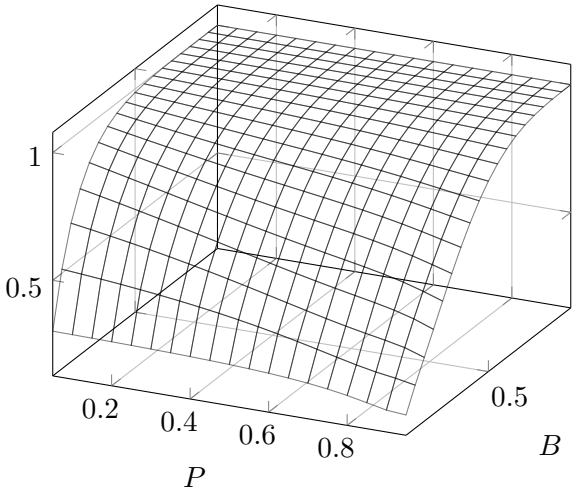


Figure 7.3: Parametric analysis of the crowds protocol for $N = 5$ internet users and $R = 7$ path reformulates. P is the probability that a user is untrustworthy, and B is the probability that a user sends a packet to a randomly selected receiver. The property of interest is the probability that untrustworthy users observe the sender more than they observe others.

where each entry shows the total time taken to check all pairwise combinations of values for pK, pL taken from $\{0.002, 0.004, \ldots, 0.998\}$. Here, ePMC with ACs again has the best performance: the running time remains approximately constant when using this data structure, even for much larger problem instances. In contrast, the running time for both engines of Storm scale linearly. Both the parametric engines of PRISM and ePMC, with polynomial fraction representation, run out of memory for all larger problem instances. Figure 7.4 plots the results obtained for $N = 256$ and $MAX = 4$. The probability of interest first increases with increasing channel reliability, but then decreases again. The reason is that, on the one hand, if the channel reliability is low, then not many chunks are sent successfully. On the other hand, if the channel reliability is high, transmission will fail.

N	MAX	States	Trans.	PRISM	ePMC	ePMC(D)	ePMC(DS)	Storm(P)	Storm(S)
64	4	4139	5543	1029	1016	36	38	991	160
64	5	4972	6695	1145	1118	36	33	1021	188
256	4	16427	22055	–M–	–M–	48	40	3332	403
256	5	19756	26663	–M–	–M–	35	15	–T–	318
512	4	32811	44071	–M–	–M–	29	19	–T–	491
512	5	39468	53287	–M–	–M–	28	23	–T–	596

Table 7.2: Performance statistics for the bounded retransmission protocol.

7.3.3 Cyclic polling server

This cyclic server polling model [86] is a model of a network, described as a continuous-time Markov chain. There are two parameters, μ and γ. The model consists of one server and N clients. When a client is idle, then a new job arrives at this client with a rate of μ/N. The server polls the clients in a cyclic manner. At each point of time, it observes a single client. If there is a job waiting for a given client, the server servers its job (provided there is one) with a rate of μ. When the client it observes is idle, then the server moves on to observe the next client with a rate of γ. Even though the AC technique targets discrete-time models, this model can be handled by computing the embedded DTMC [100].

In this case study, the property of interest is the probability that, in the long run, Station 1 is idle. That is, the expected limit average of the time that Station 1, or, due to symmetry, any other station, is idle. Probabilities are displayed as a function of the parameters in Figure 7.5, and Table 7.3 shows how the various tools perform on this benchmark. With increasing γ the likelihood that Station 1 is idle increases: if γ is increased, then the server will find stations to be served more quickly. As the long-run average idle time only depends on the rate between μ and γ, the likelihood that Station 1 is idle falls with increasing μ.

Here classic parametric model checking does not seem to be advantageous. Using the

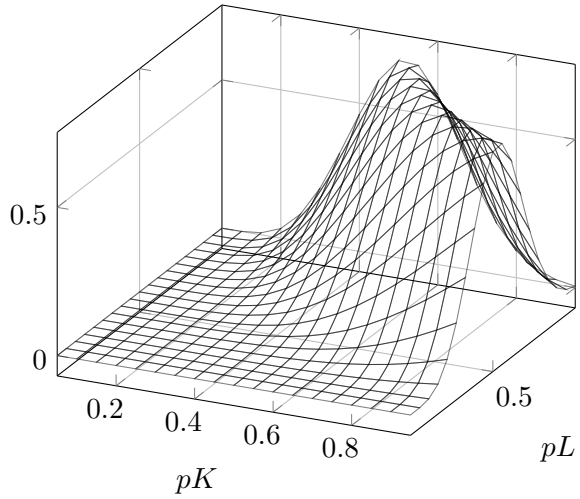


Figure 7.4: Parametric analysis of the bounded retransmission protocol for $N = 256$ file chunks and $MAX = 2$ maximum retransmissions over two lossy channels. The parameters pK and pL are the reliability of the data and acknowledgement channels, respectively. The property of interest is the probability that the sender reports an unsuccessful transmission after more than 8 chunks have been sent successfully.

AC-based implementation, however, is much more efficient than classic parametric model checking, but it is space consuming. With the chosen number of parameter instantiations, the method does not quite compete with non-parametric model checking.

7.3.4 Oscillator synchronisation

Finally, the synchronisation models presented in Chapters 4 and 5 are considered. To recall, the models encode the behaviour of a population of N coupled nodes in a network. Each node has a clock that progresses, cyclically, through a range of discrete values $1 \dots T$. At the end of each clock cycle a node transmits a message to other nodes in the network. Nodes that receive this message adjust their clocks to more closely match those of the firing node. All nodes have a refractory period during which they ignore synchronisation messages from other modes. The length of this period is given by the parameter R, that

N	States	Trans.	PRISM	ePMC	ePMC(D)	ePMC(DS)	Storm(P)	Storm(S)
4	96	272	1166	888	14	14	953	50
5	240	800	–T–	–T–	28	25	–T–	121
6	576	2208	3550	–T–	108	102	–T–	305
7	1344	5824	1399	–T–	759	736	–T–	801
8	3072	14848	1052	–T–	–T–	–T–	–T–	1991
9	6912	36864	–T–	–T–	–M–	–M–	–T–	–T–

Table 7.3: Performance statistics for the cyclic polling server model.

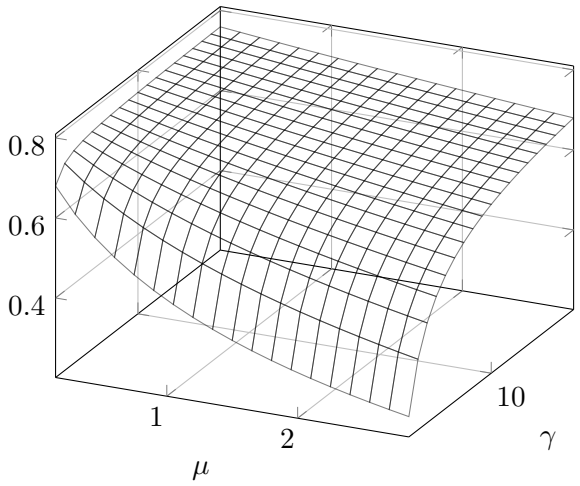


Figure 7.5: Parametric analysis of the cyclic polling server model for $N = 4$ clients. μ is the service rate and γ is the rate at which the token is moved. The property of interest is the probability that, in the long run, a station is idle.

is fixed to 1 for all instances investigated here. The model is a PMC, parametrised by the likelihood μ that a firing message is lost in the communication medium. The property of interest is the expected power consumption of the network (in Watt-hours) to reach a state where the clocks of all nodes are synchronised.

Table 7.4 shows the results for different values of N and T, where each entry shows the total time taken to check all values of μ taken from $\{10^{-5}, 2\cdot10^{-5}, \ldots, 1-10^{-5}\}$. Figure 7.6 plots the results obtained for $N = 6$ and $T = 8$. For extremal values of μ, the network is

N	T	States	Trans.	PRISM	ePMC	ePMC(D)	ePMC(DS)	Storm(P)	Storm(S)
4	6	218	508	280	304	6	4	6	20
4	7	351	822	302	399	7	4	6	28
4	8	535	1257	542	1520	9	5	13	37
5	6	449	1179	354	499	10	5	11	39
5	7	799	2094	1694	–T–	11	6	34	60
5	8	1333	3533	–T–	–T–	17	8	137	90
6	6	841	2491	1070	–T–	12	7	48	74
6	7	1639	4820	–T–	–T–	19	8	239	130
6	8	2971	8871	–T–	–T–	33	10	2311	211

Table 7.4: Performance statistics for the oscillator synchronisation model.

expected to use much more energy to synchronise, because the expected time required for this to occur increases. Very high values of μ result in nearly all firing messages being lost, and hence nodes cannot communicate well enough to coordinate, while very low values of μ lead to perpetually asynchronous states for the network. This is an artefact of the discreteness of the clock values, and was discussed in Section 4.5.

In this case study, the AC-based method, in particular with random points evaluation, performs best, followed by the sampling-based method of Storm. The time required for each value iteration is relatively high, while the cost of evaluating a point for the AC-based method is quite low. Therefore, the advantage of the approach would have been even more pronounced if more parameter instantiations had been evaluated in the experiments above. The choice of which technique to use therefore depends on whether such a high number of instantiations is required.

7.3.5 Heuristics

An important consideration when performing state elimination is the order in which different states are eliminated from the graph. Using different elimination orders to evaluate the same model can result in different functions, whose representations (nominator-

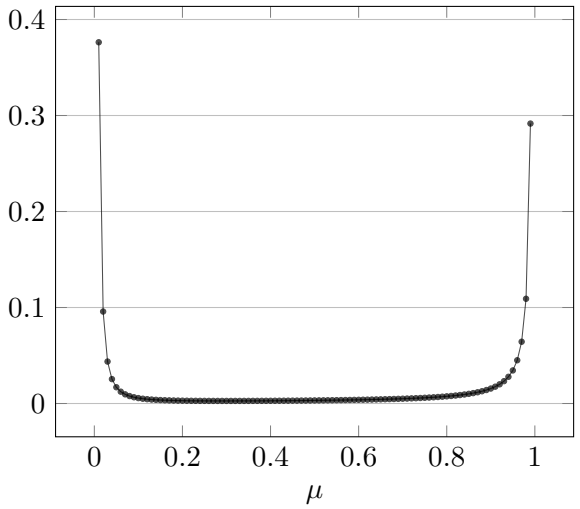


Figure 7.6: Parametric analysis of the oscillator synchronisation model for $N = 6$ nodes and $T = 8$ clock values. The parameter μ is the probability of synchronisation messages being lost in the communication medium. The property of interest is the expected power consumption of the network (in Watt-hours) to reach a state, where the clocks of all nodes are synchronised.

denominator or ACs) vary greatly in size, and hence also in the corresponding memory footprint and analysis time. Heuristics for efficient state elimination have been studied in automata theory, to obtain shorter regular expressions from finite-state automata [74, 73], and in graph theory, for efficient peeling of a probabilistic network [76]. The following heuristics were implemented into the model checker ePMC, consisting of both existing schemes taken from the literature, and novel schemes that prove to be effective for some models.

- NumNew: each state is weighted by the number of new transitions that are introduced to the model when that state is eliminated. That is, for each predecessor-successor pair for that state, add one to the weight if the transition from the predecessor to the successor was not already defined in the underlying graph before state elimination. States with the lowest weight are eliminated first. The aim here is to minimise the

Model		Elimination Heuristic					
		NumNew	MinProd	TargetBFS	Random	BFS	ReverseBFS
Crowds	(N=10,R=5)	19	103	5	14	22	6
BRP	(N=512,MAX=5)	4	11	4	–M–	5	4
Cyclic	(N=7)	7	9	8	8	8	8
Synch	(N=6,T=8)	18	18	17	19	18	17

Table 7.5: Performance statistics for different heuristics.

total number of transitions as elimination progresses.

- MinProd: similarly to NumNew, each predecessor-successor pair is considered. However, one is added to the weight irrespective of whether that transition already existed in the underlying graph. Again states with the lowest weight are considered first.

- TargetBFS: states are eliminated in the order in which they are discovered when conducting a breadth-first search backwards from the target states.

- Random: a state is selected uniformly at random for elimination from the set of remaining states.

- BFS: states are eliminated in the order in which they are discovered when conducting a breadth-first search from the initial state(s) of the model.

- ReverseBFS: similar to BFS, except states are eliminated in reverse order.

Table 7.5 shows the times taken for state elimination of a medium-sized instance of each of the case studies, when employing each of the heuristics described above. In general, TargetBFS appears to be a good choice. In one case, however, NumNew turns out to be faster.

7.4 Conclusion and future work

An approach has been presented for the evaluation of parametric Markov chains that exploits the synergies of using ACs in a state-elimination based analysis and using ACs in an encoding of rational functions as arithmetic circuits. The experimental evaluation suggests that these two approaches integrated so seamlessly that they often provide a notable speed up. The nicest observation is that this seems so natural in hindsight, and it is surprising that to the best of our knowledge this has not been attempted before.

Of particular interest here is the analysis of the synchronisation model introduced in Chapters 4 and 5. Using the existing parametric engine of the model checker PRISM failed to give a closed form solution for nearly all of the model instances explored in Chapter 4. The new technique allowed the influence of the broadcast failure parameter to be easily visualised, and accordingly analysed. A similar analysis could now be conducted for the models presented in Chapter 3, where the influence of the parameter determining the likelihood of choosing unexplored paths over explored paths could be visualised, and more thoroughly analysed, even for large model instances.

Chapter 8

Summary and Discussion

In this chapter the main contributions of this thesis are summarised. Directions for future research for the formal analysis of artificial collectives are given, focusing on the development of new techniques for the analysis of parametric Markov chain models of these systems. Finally, a discussion of how the work presented in this thesis addresses the research questions presented in Section 1.3 is given.

In the case study presented in Chapter 3 a formal, parametric population model of a swarm of flying vehicles is developed. The swarm imitates the foraging behaviour of a colony of ants, with the aim of locating a target at some unknown location in a potentially hostile environment. The formal model is validated by comparing the results of an initial analysis to those obtained through simulation. The parametric model is instantiated and by checking probabilistic reachability and reachability reward properties it is demonstrated that results could be obtained that could be used to plan the deployment of a real swarm. Battery life greatly impacts the flight duration of swarm members, and the a priori calculation of the total expected flight time, or total expected distance travelled, would ensure that sufficient resources could be made available to ensure that the swarm achieves its

objectives.

A natural extension of this work would be to perform a parametric analysis of the developed model, to investigate the relationship between the parameter defining the rate of exploration and the likelihood of the swarm finding the target. Such an analysis could determine the optimal values for this parameter to maximise or minimise probabilistic reachability or reachability reward properties of interest. In addition, new reward functions could be developed for a more detailed analysis of the power consumption of the swarm, similar to those presented in Chapter 5, since the rate of power consumption for each micro air vehicle in the swarm would differ according to its current mode of operation, and the rate at which it communicates with the rest of the swarm.

In Chapter 4 a further case study investigates the phenomenon of emergent synchronisation in a network of nodes, where nodes reach consensus on their local clocks by imitating the behaviour of fireflies. This behaviour is achieved through the dynamics of interacting pulse-coupled oscillators, whose cyclic behaviour and discrete interactions lead to global synchronisation. Again, a formal, parametric population model is developed, where oscillation cycles for each node were defined as sequences of discrete states. Models are generated for two models of synchronisation that have been used for the coordination of swarm robotic systems or wireless sensor nets. Each formal model is instantiated for a range of parameter values, and the parametric influence on both the rate at which synchronisation occurs, and the time taken for it to occur, are investigated, and trade-offs with respect to properties of interest are discussed. The population model is then further refined, and the state-space explosion typically encountered during the analysis of large models is mitigated by collapsing deterministic paths in the model. This results in models that are equivalent to the original model with respect to the likelihood of reaching a synchronised state.

Formal population models are appropriate when nodes in the network are indistinguishable with respect to their behaviour, and when all nodes in that network can communicate with each other. To analyse more complex network topologies, each component of the system could be encoded as a separate population model, and then the product of all such components could then be analysed. However, interactions between components would need to be refined, to offset the inevitable explosion in the size of the model. Furthermore, topologies that change over time are of particular interest. However, it is not obvious how the approach could be extended to consider such dynamic networks. The work of Lucarelli and Wang may serve as a starting point for further investigations [113].

The work in Chapter 5 extends the work presented in Chapter 4. A new metric for the synchronisation of a network of nodes is introduced, derived from existing work in the literature. This metric is then used to investigate the power consumption of the synchronising networks described in Section 4. Suitable reward functions are defined that would encode the rate of power consumption for nodes in a wireless sensor network, or member of a swarm, given real values obtainable from hardware specifications for those devices. It is also demonstrated how these reward functions could be automatically applied to models resulting from the refinement presented in Section 4.6. Properties relating to the restabilisation of a small number of oscillators, when the network has synchronised, and synchrony has then been lost, in a network are investigated, and it is demonstrated that only a fraction of the state-space of a model needs to be explored for their analysis. The introduced techniques are used to analyse the power consumption of a network of MICAz motes, and a reduced population model is generated for a widely-used model of synchronisation taken from the literature, and then instantiated for a range of parameter values. Using the reduced model it is possible to extend the size of the networks that could be analysed in Chapter 4. The results obtained using these techniques could be used

during the design phase of wireless sensor networks, or swarm robotic systems, to identify and rectify unnecessary energy consumption.

While the development of more refined population models allow larger networks to be analysed, it is clear that the approach is still inhibited by the limitation of exact probabilistic model checking for large-scale systems. While this could be overcome by using approximated techniques, such as statistical model checking (see Section 2.6), or approaches based on fluid-flow approximation extended with rewards [20], this would, of course, come at the expense of precision. An investigation of such a trade-off is deferred to future work.

In Chapter 6, new algorithms are introduced that exploit the structural similarity between instances of parametrised models. Small changes to the value of a parameter sometimes result in comparably small changes in the structure of the underlying graph. It is shown that this observation can be used to reuse much of the analysis between different instances of parametric models, by maintaining information about subformulae of rational functions between model instances. This technique is shown to be highly effective for models where the underlying graph was composed of replicated sub-structures. The limitation of this approach are that much of the advantage was lost when a change in a parameter induced severe structural changes in the model. For the synchronisation protocols presented in Chapters 4 and 5, some model parameters change the structure of the model. Changing some parameters results in a large amount of structural change in the underlying graph of the model. However, the cone of influence for other parameters lead to relatively minor changes in the graph, and the technique is shown to be effective for their analysis.

A natural extension of this work would be to tap the full potential of the approach by integrating it into existing probabilistic model checking software. Here the symbolic descriptions of the system would expose those volatile areas of the underling graph of the

model where structural changes might occur, and more importantly, those areas where the structure appears to be stable under successive increments of a parameter value. For example, it is likely that volatile areas of a model formalised using the state-based language described in Section 2.7.1 could be identified by analysing the guards of commands. Synergies might also be obtained by combining this method with the approach of [60], by extending the approach to models with non-determinism, such as Markov decision processes.

In Chapter 7 an approach is presented for the analysis of parametric Markov chains, where arithmetic circuits are used to encode the rational functions labelling the transitions of the model. Empirical results show the effectiveness of the technique when applied to a range of distributed network protocols taken from the literature. In particular, the technique is applied to the synchronisation model introduced in Chapters 4 and 5. Using the existing parametric engine of the model checker PRISM fails to give a closed form solution for nearly all of the model instances explored in Chapters 4. The new technique allows the influence of the parameter defining the likelihood of message loss to be analysed. A similar analysis could now be extended to the the models presented in Chapter 3, to investigate the influence of the parameter determining the rate of exploration.

A natural extension of this work would be to combine the approach with the incremental techniques introduced in Chapter 6. Instead of maintaining information about subformulae of rational functions between model instances, substructures of the arithmetic circuit could be maintained. A possible next step in exploiting the approach could be its integration in the context of parameter extraction, which would be expected to work similar to model extraction, for online model checking. The growing knowledge of the model could be used to refine or adjust the parameters in the application. The technique could help to provide the speed required to make the approach scale, and to keep the analysis and, if required,

the visualisation of the effect of the learnt parameters efficient.

To address the first research question posed in this thesis, that asks if parametric Markov models could be used to efficiently reason about artificial collectives, case studies are conducted in Chapters 3, 4, and 5. For each of these case studies, macroscopic models are constructed that alleviate state-space explosion. The formal models of the ant-inspired behaviour of a swarm of vehicles developed in Chapter 3 are validated against results obtained through simulation, and then properties relating to the logistics of the deployment of such a swarm are checked in the models. While model checking is possible to analyse swarms of up to eight individuals, statistical methods have to be used for the analysis of larger swarms. The formal analysis of the models of synchronising network nodes developed in Chapters 4 and 5 yields results that match those obtained from both simulation and empirical evaluation. However, the analysis is similarly impeded by the state-space explosion encountered when checking desirable properties for networks of more than eight nodes. A further refinement of the macroscopic model helps to mitigate state-space explosion to some extent, however the reduction achieved becomes increasingly insignificant as the size of the formal models increases.

The macroscopic models developed in the case studies help to delay state space explosion, and allow larger models to be analysed. However, it is clear that the analysis of such models is still restricted by both state explosion, and the time taken to analyse large number of model instances. To combat this, and to address the second research question that asks whether new techniques could be developed to facilitate the parametric analysis of large families of parametric Markov models, novel algorithms are proposed in Chapter 6 for their analysis. These algorithms exploited structural similarities between parameterwise-different instances of a model to reduce the time and resources required for their analysis, and are shown to be effective for the analysis of the family of synchronisation

models developed in Chapters 4 and 5, and for a model taken the literature.

To address the third research question, that asks how the efficiency of existing techniques for the parametric analysis of families of parametric Markov chains could be improved, the representation of the rational functions labelling model transitions as arithmetic circuits is proposed. It is demonstrated that this new representation integrates with existing techniques for the analysis of such models, and its implementation into an existing probabilistic model checker yields results that demonstrated the effectiveness of the technique in facilitating the analysis of much larger models.

Bibliography

[1] Ian F Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[2] Susanne Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

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

[4] Rajeev Alur and Thomas A Henzinger. Reactive modules. *Formal Methods in System Design*, 15(1):7–48, 1999.

[5] Dana Angluin, James Aspnes, Zoë Diamadi, Michael J Fischer, and René Peralta. Computation in networks of passively mobile finite-state sensors. *Distributed computing*, 18(4):235–253, 2006.

[6] Ezio Bartocci, Flavio Corradini, Emanuela Merelli, and Luca Tesei. Detecting synchronisation of biological oscillators by model checking. *Theoretical Computer Science*, 411(20):1999–2018, 2010.

[7] Ezio Bartocci, Radu Grosu, Panagiotis Katsaros, C. R. Ramakrishnan, and Scott A. Smolka. Model repair for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 326–340, 2011.

[8] Ralph Beckers, Jean-Louis Deneubourg, and Simon Goss. Modulation of trail laying in the antlasius niger (hymenoptera: Formicidae) and its role in the collective selection of a food source. *Journal of Insect Behavior*, 6(6):751–759, 1993.

[9] Abdelkader Behdenna, Clare Dixon, and Michael Fisher. Deductive verification of simple foraging robotic behaviours. *International Journal of Intelligent Computing and Cybernetics*, 2(4):604–643, 2009.

[10] Gerd Behrmann, Alexandre David, Kim Guldstrand Larsen, John Håkansson, Paul Pettersson, Wang Yi, and Martijn Hendriks. Uppaal 4.0. In *Proc. QEST 2006*, pages 125–126. IEEE Computer Society, 2006.

[11] Gerardo Beni. From swarm intelligence to swarm robotics. In *SAB 2004 Revised Selected Papers*, volume 3342 of *LNCS*, pages 1–9. Springer, 2005.

[12] Francesco Bernardini and Marian Gheorghe. Population p systems. *J. UCS*, 10(5):509–539, 2004.

[13] Henrik Bohnenkamp, Peter van der Stok, Holger Hermanns, and Frits Vaandrager. *Cost-optimization of the IPv4 zeroconf protocol*, pages 531–540. IEEE Computer Society Press, 2003.

[14] Eerke Boiten. Modeling in event-b-. *Journal of Functional Programming*, 22(2):217–219, 2012.

[15] Iva Bojic, Tomislav Lipic, and Mario Kusek. Scalability issues of firefly-based self-synchronization in collective adaptive systems. In *International Conference on Self-Adaptive and Self-Organizing Systems Workshops*, pages 68–73. IEEE, 2014.

[16] Iva Bojic, Vedran Podobnik, Igor Ljubi, Gordan Jezic, and Mario Kusek. A self-optimizing mobile network: Auto-tuning the network with firefly-synchronized agents. *Information Science*, 182(1):77–92, 2012.

[17] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, 1999.

[18] Eric Bonabeau, Andrej Sobkowski, Guy Theraulaz, and Jean-Louis Deneubourg. Adaptive task allocation inspired by a model of division of labor in social insects. In *Biocomputing and Emergent Computation*, pages 36–45. World Scientific, 1997.

[19] Eric Bonabeau, Guy Theraulaz, Jean-Louls Deneubourg, Serge Aron, and Scott Camazine. Self-organization in social insects. *Trends in ecology & evolution*, 12(5):188–193, 1997.

[20] Luca Bortolussi and Jane Hillston. Efficient checking of individual rewards properties in markov population models. In *Workshop on Quantitative Aspects of Programming Languages*, volume 194 of *EPTCS*, pages 32–47. Open Publishing Association, 2015.

[21] Luca Bortolussi, Jane Hillston, Diego Latella, and Mieke Massink. Continuous approximation of collective system behaviour: A tutorial. *Performance Evaluation*, 70(5):317–349, 2013.

[22] Andrew FG Bourke, Nigel R Franks, and Nigel R Franks. *Social evolution in ants*. Princeton University Press, 1995.

[23] Manuele Brambilla, Arne Brutschy, Marco Dorigo, and Mauro Birattari. Property-driven design for robot swarms: A design method based on prescriptive modeling and model checking. *ACM Transactions on Autonomous and Adaptive Systems*, 9(4):17, 2015.

[24] Manuele Brambilla, Marco Dorigo, and Mauro Birattari. *Formal methods for the design and analysis of robot swarms*. PhD thesis, Ph. D. thesis, Universite Libre de Bruxelles, Berlin, Heidelberg, 2014.

[25] Manuele Brambilla, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Property-driven design for swarm robotics. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 139–146. International Foundation for Autonomous Agents and Multiagent Systems, 2012.

[26] Toma Brazdil, Stefan Kiefer, Antonin Kucera, and Petr Novotnỳ. Long-run average behaviour of probabilistic vector addition systems. In *ACM/IEEE Symposium on Logic in Computer Science*, pages 44–55. IEEE, 2015.

[27] Michael Breza. *Bio-inspired tools for a distributed wireless Sensor network operating system*. PhD thesis, Imperial College London, 2013.

[28] Radu Calinescu, Carlo Ghezzi, Marta Kwiatkowska, and Raffaela Mirandola. Self-adaptive software needs quantitative verification at runtime. *Communications of the ACM*, 55(9):69–77, 2012.

[29] Alexandre Campo and Marco Dorigo. Efficient multi-foraging in swarm robotics. In *European Conference on Artificial Life*, volume 4648 of *LNCS*, pages 696–705. Springer, 2007.

[30] Matteo Casadei and Mirko Viroli. Using probabilistic model checking and simulation for designing self-organizing systems. In *ACM symposium on Applied Computing*, pages 2103–2104. ACM, 2009.

[31] Chih-Chun Chen, Sylvia B Nagl, and Christopher D Clack. Identifying multi-level emergent behaviours in agent-based simulations using complex event type specifications. *Simulation Journal special issue: Recent Advances in Unified Modeling and Simulation Approaches*, 2009, 2008.

[32] Zhe Chen, Daqiang Zhang, Rongbo Zhu, Yinxue Ma, Ping Yin, and Feng Xie. A review of automated formal verification of ad hoc routing protocols for wireless sensor networks. *Sensor Letters*, 11(5):752–764, 2013.

[33] A. L. Christensen, R. O. Grady, and M. Dorigo. From fireflies to fault-tolerant swarms of robots. *IEEE Transations on Evolutionary Computation*, 13(4):754–766, 2009.

[34] Pamela E Clark, Steven A Curtis, and Michael L Rilee. Ants: applying a new paradigm for lunar and planetary exploration. In *Astrophysics and Space Science Library*, volume 278, page 15, 2002.

[35] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[36] Edmund M Clarke, E Allen Emerson, and Joseph Sifakis. Model checking: algorithmic verification and debugging. *Communications of the ACM*, 52(11):74–84, 2009.

[37] Edmund M. Clarke, E Allen Emerson, and A Prasad Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[38] Edmund M. Clarke, Reinhard Enders, Thomas Filkorn, and Somesh Jha. Exploiting symmetry in temporal logic model checking. *Formal methods in system design*, 9(1-2):77–104, 1996.

[39] Simon Coakley, Rod Smallwood, and Mike Holcombe. Using x-machines as a formal basis for describing agents in agent-based modelling. *Simulation Series*, 38(2):33, 2006.

[40] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, Ivan Papusha, Hasan A. Poonawala, and Ufuk Topcu. Sequential convex programming for the efficient verification of parametric mdps. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 133–150, 2017.

[41] Conrado Daws. Symbolic and parametric model checking of discrete-time markov chains. In *International Colloquium on Theoretical Aspects of Computing*, pages 280–294. Springer, 2004.

[42] Julius Degesys, Prithwish Basu, and Jason Redi. Synchronization of strongly pulse-coupled oscillators with refractory periods and random medium access. In *ACM Symposium on Applied Computing*, pages 1976–1980. ACM, 2008.

[43] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám. PROPhESY: A probabilistic parameter synthesis tool. In *International Conference on Computer Aided Verification*, pages 214–231, 2015.

[44] Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius, Matthias Volk, Joost-Pieter Katoen, Erika Ábrahám, and Harold Bruintjes. Parameter synthesis for probabilistic systems. In *Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen*, pages 72–74, 2016.

[45] Christian Dehnert, Sebastian Junges, Joost-Pieter Katoen, and Matthias Volk. A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer, 2017.

[46] Richard A. DeMillo and Richard J. Lipton. A probabilistic remark on algebraic program testing. *Information Processing Letters*, 7:193–195, 1978.

[47] Jean-Louis Deneubourg, Simon Goss, Nigel Franks, and J. M. Pasteels. The blind leading the blind: modeling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, 2(5):719–725, 1989.

[48] Clare Dixon, Alan FT Winfield, Michael Fisher, and Chengxiu Zeng. Towards temporal verification of swarm robotic systems. *Robotics and Autonomous Systems*, 60(11):1429–1441, 2012.

[49] Alastair F Donaldson and Alice Miller. Symmetry reduction for probabilistic model checking using generic representatives. In *International Symposium on Automated Technology for Verification and Analysis*, volume 4218 of *LNCS*, pages 9–23. Springer, 2006.

[50] Alastair F Donaldson, Alice Miller, and David Parker. Grip: Generic representatives in prism. In *International Conference on Quantitative Evaluation of Systems*, pages 115–116. IEEE, 2007.

[51] Alastair F Donaldson, Alice Miller, and David Parker. Language-level symmetry reduction for probabilistic model checking. In *International Conference on the Quantitative Evaluation of Systems*, pages 289–298. IEEE, 2009.

[52] Marco Dorigo and Mauro Birattari. *Ant colony optimization*. Springer, 2010.

[53] Samuel Eilenberg. *Automata, languages, and machines*. Academic press, 1974.

[54] George Eleftherakis. *Formal verification of X-machine models: Towards formal development of computer-based systems*. PhD thesis, University of Sheffield, 2003.

[55] E Allen Emerson and Richard J Trefler. From asymmetry to full symmetry: New techniques for symmetry reduction in model checking. In *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*, volume 1703 of *LNCS*, pages 142–156. Springer, 1999.

[56] Nazim Fatès. Remarks on the cellular automaton global synchronisation problem. In *AUTOMATA*, volume 9099 of *LNCS*, pages 113–126. Springer, 2015.

[57] Ansgar Fehnker and Peng Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In *International Conference on Ad-Hoc, Mobile, and Wireless Networks*, volume 4104 of *LNCS*, pages 128–141. Springer, 2006.

[58] William Feller. *An introduction to probability theory and its applications: volume I*, volume 3. John Wiley & Sons New York, 1968.

[59] Antonio Filieri, Carlo Ghezzi, and Giordano Tamburrelli. Run-time efficient probabilistic model checking. In *International Conference on Software Engineering*, pages 341–350. ACM, 2011.

[60] Vojtěch Forejt, Marta Kwiatkowska, David Parker, Hongyang Qu, and Mateusz Ujma. Incremental runtime verification of probabilistic systems. In *International Conference on Runtime Verification*, pages 314–319. Springer, 2012.

[61] Paul Gainer, Clare Dixon, and Ullrich Hustadt. Probabilistic model checking of ant-based positionless swarming. In *Conference Towards Autonomous Robotic Systems*, volume 9716 of *LNCS*, pages 127–138. Springer, 2016.

[62] Paul Gainer, Ernst Moritz Hahn, and Sven Schewe. Accelerated model checking of parametric markov chains. In *International Symposium on Automated Technology for Verification and Analysis*, pages 300–316. Springer, 2018.

[63] Paul Gainer, Ernst Moritz Hahn, and Sven Schewe. Incremental verification of parametric and reconfigurable markov chains. In *International Conference on Quantitative Evaluation of Systems*, pages 140–156. Springer, 2018.

[64] Paul Gainer, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. Multi-scale verification of distributed synchronisation. *Formal Methods in System Design*. Submitted for publication.

[65] Paul Gainer, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. Investigating parametric influence on discrete synchronisation protocols using quantitative model checking. In *International Conference on Quantitative Evaluation of Systems*, volume 10503 of *LNCS*, pages 224–239. Springer, 2017.

[66] Paul Gainer, Sven Linker, Clare Dixon, Ullrich Hustadt, and Michael Fisher. The power of synchronisation: formal analysis of power consumption in networks of pulse-coupled oscillators. In *International Conference on Formal Engineering Methods*, pages 160–176. Springer, 2018.

[67] Marian Gheorghe, Mike Holcombe, and Petros Kefalas. Computational models of collective foraging. *BioSystems*, 61(2-3):133–141, 2001.

[68] Ernst Moritz Hahn, Tingting Han, and Lijun Zhang. Synthesis for PCTL in parametric markov decision processes. In *NASA Formal Methods Symposium*, pages 146–161, 2011.

[69] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Param: A model checker for parametric markov models. In *International Conference on Computer Aided Verification*, pages 660–664. Springer, 2010.

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

[71] Ernst Moritz Hahn, Yi Li, Sven Schewe, Andrea Turrini, and Lijun Zhang. iscas m c: a web-based probabilistic model checker. In *Formal Methods*, pages 312–317. Springer, 2014.

[72] Stefan Hallerstede and Thai Son Hoang. Qualitative probabilistic modelling in event-b. In *International Conference on Integrated Formal Methods*, pages 293–312. Springer, 2007.

[73] Yo-Sub Han. State elimination heuristics for short regular expressions. *Fundamenta Informaticae*, 128(4):445–462, 2013.

[74] Yo-Sub Han and Derick Wood. Obtaining shorter regular expressions from finite-state automata. *Theoretical Computer Science*, 370(1-3):110–120, 2007.

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

[76] Chris Harbron. Heuristic algorithms for finding inexpensive elimination schemes. *Statistics and Computing*, 5(4):275–287, 1995.

[77] Sabine Hauert, Laurent Winkler, Jean-Christophe Zufferey, and Dario Floreano. Ant-based swarming with positionless micro air vehicles for communication relay. *Swarm Intelligence*, 2(2-4):167–188, 2008.

[78] Faranak Heidarian, Julien Schmaltz, and Frits Vaandrager. Analysis of a clock synchronization protocol for wireless sensor networks. *Theoretical Computer Science*, 413(1):87–105, 2012.

[79] Leen Helmink, Martin Paul Alexander Sellink, and Frits W Vaandrager. Proof-checking a data link protocol. In *International Workshop on Types for Proofs and Programs*, pages 127–165. Springer, 1993.

[80] Benjamin Herd, Simon Miles, Peter McBurney, and Michael Luck. Approximate verification of swarm-based systems: a vision and preliminary results. In *Safety-critical Systems Symposium*, pages 361–378. SCSC, SCSC, 2015.

[81] Jane Hillston, Mirco Tribastone, and Stephen Gilmore. Stochastic process algebras: From individuals to populations. *The Computer Journal*, 55(7):866–881, 2011.

[82] Peter Höfner and Maryam Kamali. Quantitative analysis of aodv and its variants on dynamic topologies using statistical model checking. In *International Conference on Formal Modeling and Analysis of Timed Systems*, volume 8053 of *LNCS*, pages 121–136. Springer, 2013.

[83] John Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8(2):135–159, 1979.

[84] John E Hopcroft. *Introduction to automata theory, languages, and computation*. Pearson Education India, 2008.

[85] Bret Hull, Kyle Jamieson, and Hari Balakrishnan. Mitigating congestion in wireless sensor networks. In *International Conference on Embedded Networked Sensor Systems*, pages 134–147. ACM, 2004.

[86] Oliver C. Ibe and Kishor S. Trivedi. Stochastic petri net models of polling systems. *IEEE Journal on Selected Areas in Communications*, 8(9):1649–1657, 1990.

[87] Sandy Irani and Kirk R Pruhs. Algorithmic problems in power management. *ACM Sigact News*, 36(2):63–76, 2005.

[88] Nils Jansen, Florian Corzilius, Matthias Volk, Ralf Wimmer, Erika Ábrahám, Joost-Pieter Katoen, and Bernd Becker. Accelerating parametric probabilistic verification. In *International Conference on Quantitative Evaluation of Systems*, pages 404–420, 2014.

[89] Benjamin Johnson and Hadas Kress-Gazit. Probabilistic analysis of correctness of high-level robot behavior with sensor error. In *Robotics: Science and Systems*, 2011.

[90] Benjamin Johnson and Hadas Kress-Gazit. Probabilistic guarantees for high-level robot behavior in the presence of sensor error. *Autonomous Robots*, 33(3):309–321, 2012.

[91] Benjamin Lee Johnson. *Synthesis, analysis, and revision of correct-by-construction controllers for robots with sensing and actuation errors*. PhD thesis, Cornell University, 2015.

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

[93] John G Kemeny, J Laurie Snell, and Anthony W Knapp. *Denumerable Markov chains: with a chapter of Markov random fields by David Griffeath*, volume 40. Springer Science & Business Media, 2012.

[94] Donald Knuth and Andrew Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[95] Keiji Konishi and Hideki Kokame. Synchronization of pulse-coupled oscillators with a refractory period and frequency distribution for a wireless sensor network. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 18(3), 2008.

[96] Savas Konur, Clare Dixon, and Michael Fisher. Analysing robot swarm behaviour via probabilistic model checking. *Robotics and Autonomous Systems*, 60(2):199–213, 2012.

[97] Panagiotis Kouvaros and Alessio Lomuscio. Verifying emergent properties of swarms. In *Proc. AAAI 2015*, pages 1083–1089. AAAI Press, 2015.

[98] Yoshiki Kuramoto. Self-entrainment of a population of coupled non-linear oscillators. In *International Symposium on Mathematical Problems in Theoretical Physics*, volume 39 of *LNP*, pages 420–422. Springer, 1975.

[99] Yoshiki Kuramoto. Collective synchronization of pulse-coupled oscillators and excitable units. *Physica D: Nonlinear Phenomena*, 50(1):15–30, 1991.

[100] Marta Kwiatkowska, Gethin Norman, and David Parker. Stochastic model checking. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 220–270. Springer, 2007.

[101] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism 4.0: Verification of probabilistic real-time systems. In *International Conference on Computer Aided Verification*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[102] Marta Kwiatkowska, David Parker, and Hongyang Qu. Incremental quantitative verification for markov decision processes. In *International Conference on Dependable Systems & Networks*, pages 359–370. IEEE, 2011.

[103] Linas Laibinis, Elena Troubitsyna, Zeineb Graja, Frédéric Migeon, and Ahmed Hadj Kacem. Formal modelling and verification of cooperative ant behaviour in event-b. In *International Conference on Software Engineering and Formal Methods*, pages 363–377. Springer, 2014.

[104] Ruggero Lanotte, Andrea Maggiolo-Schettini, and Angelo Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Aspects of Computing*, 19(1):93–109, 2007.

[105] Diego Latella, Michele Loreti, and Mieke Massink. On-the-fly pctl fast mean-field approximated model-checking for self-organising coordination. *Science of Computer Programming*, 110:23–50, 2015.

[106] Diego Latella, Michele Loreti, and Mieke Massink. Flyfast: A scalable approach to probabilistic model-checking based on mean-field approximation. In *ModelEd, TestEd, TrustEd*, pages 254–275. Springer, 2017.

[107] Gilbert Thomas Laycock. *The theory and practice of specification based software testing*. PhD thesis, Citeseer, 1993.

[108] Axel Legay, Benoît Delahaye, and Saddek Bensalem. Statistical model checking: An overview. In *International conference on runtime verification*, pages 122–135. Springer, 2010.

[109] Kristina Lerman, Alcherio Martinoli, and Aram Galstyan. A review of probabilistic macroscopic models for swarm robotic systems. In *International Workshop on Swarm Robotics*, pages 143–152. Springer, 2004.

[110] Wenguo Liu, Alan Winfield, Jin Sa, Jie Chen, and Lihua Dou. Strategies for energy optimisation in a swarm of foraging robots. In *SAB 2007 Revised Selected Papers*, volume 4433 of *LNCS*, pages 14–26. Springer, 2007.

[111] Alessio Lomuscio and Edoardo Pirovano. Verifying emergence of bounded time properties in probabilistic swarm systems. In *International Joint Conferences on Artificial Intelligence*, pages 403–409, 2018.

[112] Alessio Lomuscio and Edoardo Pirovano. A counter abstraction technique for the verification of probabilistic swarm systems. In *International Conference on Autonomous Agents and MultiAgent Systems*, pages 161–169. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[113] Dennis Lucarelli and I-Jeng Wang. Decentralized synchronization protocols with nearest neighbor communication. In *International Conference on Embedded Networked Sensor Systems*, pages 62–68. ACM, 2004.

[114] Mieke Massink and Diego Latella. Fluid analysis of foraging ants. In *International Conference on Coordination Languages and Models*, pages 152–165. Springer, 2012.

[115] MEMSIC Inc. MICAz datasheet. `www.memsic.com/userfiles/files/Datasheets/WSN/micaz_datasheet-t.pdf`, last accessed 15th Jan, 2018.

[116] Mercedes G Merayo and Manuel Núnez. Testing conformance on stochastic stream x-machines. In *International Conference on Software Engineering and Formal Methods*, pages 227–236. IEEE, 2007.

[117] Renato E. Mirollo and Steven H. Strogatz. Synchronization of pulse-coupled biological oscillators. *SIAM Journal on Applied Mathematics*, 50(6):1645–1662, 1990.

[118] Michael K. Molloy. Discrete time stochastic petri nets. *IEEE Transactions on Software Engineering*, 4:417–423, 1985.

[119] Joaquim Oller, Ilker Demirkol, Jordi Casademont, Josep Paradells, Gerd Ulrich Gamm, and Leonhard Reindl. Has time come to switch from duty-cycled mac protocols to wake-up radio for wireless sensor networks? *IEEE/ACM Transactions on Networking*, 24(2):674–687, 2016.

[120] David Anthony Parker. *Implementation of symbolic model checking for probabilistic systems.* PhD thesis, University of Birmingham, 2002.

[121] Fernando Perez-Diaz, Stefan M. Trenkwalder, Ruediger Zillmer, and Roderich Groß. Emergence and inhibition of synchronization in robot swarms. In *International Symposium on Distributed Autonomous Robotic Systems*, Springer Tracts in Advanced Robotics. Springer, in press.

[122] Fernando Perez-Diaz, Ruediger Zillmer, and Roderich Groß. Firefly-inspired synchronization in swarms of mobile agents. In *International Conference on Autonomous Agents and Multiagent Systems*, pages 279–286. International Foundation for Autonomous Agents and Multiagent Systems, 2015.

[123] Charles S Peskin. *Mathematical aspects of heart physiology.* Courant Lecture Notes. Courant Institute of Mathematical Sciences, New York University, 1975.

[124] M Praveen and Kamal Lodaya. Analyzing reachability for some petri nets with fast growing markings. *Electronic Notes in Theoretical Computer Science*, 223:215–237, 2008.

[125] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[126] Tim Quatmann, Christian Dehnert, Nils Jansen, Sebastian Junges, and Joost-Pieter Katoen. Parameter synthesis for markov models: Faster than ever. In *International Symposium on Automated Technology for Verification and Analysis*, pages 50–67, 2016.

[127] Michael K Reiter and Aviel D Rubin. Crowds: Anonymity for web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[128] Ill-Keun Rhee, Jaehan Lee, Jangsub Kim, Erchin Serpedin, and Yik-Chung Wu. Clock synchronization in wireless sensor networks: An overview. *Sensors*, 9(1):56–85, 2009.

[129] Sokwoo Rhee, Deva Seetharam, and Sheng Liu. Techniques for minimizing power consumption in low data-rate wireless sensor networks. In *Wireless Communications and Networking Conference*, pages 1727–1731. IEEE, 2004.

[130] Kay Romer and Friedemann Mattern. The design space of wireless sensor networks. *IEEE wireless communications*, 11(6):54–61, 2004.

[131] Christopher A Rouff, Walter Truszkowski, James Rash, and Mike Hinchey. A survey of formal methods for intelligent swarms. *Greenbelt, MD: NASA Goddard Space Flight Center*, 2005.

[132] Erol Şahin. Swarm robotics: From sources of inspiration to domains of application. In *International workshop on swarm robotics*, pages 10–20. Springer, 2004.

[133] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM*, 27(4):701–717, October 1980.

[134] Koushik Sen, Mahesh Viswanathan, and Gul Agha. Statistical model checking of black-box probabilistic systems. In *International Conference on Computer Aided Verification*, pages 202–215. Springer, 2004.

[135] Kazem Sohraby, Daniel Minoli, and Taieb Znati. *Wireless sensor networks: technology, protocols, and applications*. John Wiley and Sons, Ltd, 2007.

[136] Ridha Soua and Pascale Minet. A survey on energy efficient techniques in wireless sensor networks. In *Wireless and Mobile Networking Conference*, pages 1–9. IEEE, 2011.

[137] Ioanna Stamatopoulou, Ilias Sakellariou, Petros Kefalas, and George Eleftherakis. OPERAS for social insects: formal modelling and prototype simulation. *Science and Technology*, 11(3):267–280, 2008.

[138] Kasper Støy. Using situated communication in distributed autonomous mobile robotics. In *Seventh Scandinavian Conference on Artificial Intelligence*, pages 44–52. IOS Press, 2001.

[139] David JT Sumpter, Guy B Blanchard, and David S Broomhead. Ants and agents: a process algebra approach to modelling ant colony behaviour. *Bulletin of Mathematical Biology*, 63(5):951–980, 2001.

[140] Yoshiaki Taniguchi, Naoki Wakamiya, and Masayuki Murata. A distributed and self-organizing data gathering scheme in wireless sensor networks. In *Information and Telecommunication Technologies*, pages 299–304. IEEE, 2005.

[141] Guy Theraulaz and Eric Bonabeau. Coordination in distributed building. *Science*, 269:686–686, 1995.

[142] Chris Tofts. Describing social insect behaviour using process algebra. *Transactions of the Society for Computer Simulation*, 9:227–227, 1992.

[143] Chris Tofts. Algorithms for task allocation in ants.(a study of temporal polyethism: theory). *Bulletin of Mathematical Biology*, 55(5):891–918, 1993.

[144] Alexander Tyrrell, Gunther Auer, and Christian Bettstetter. Fireflies as role models for synchronization in ad hoc networks. In *International Conference on Bio inspired Models of Network, Information and Computing systems*, page 4. ACM, 2006.

[145] Moshe Y Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE, 1985.

[146] Dargie Waltenegus and Poellabauer Christian. *Fundamentals of wireless sensor networks: theory and practice*. John Wiley and Sons, Ltd, 2011.

[147] Yongqiang Wang, Felipe Nuñez, and Francis J Doyle. Energy-efficient pulse-coupled synchronization strategy design for wireless sensor networks through reduced idle listening. *IEEE Transactions on Signal Processing*, 60(10):5293–5306, 2012.

[148] Geoffrey. Werner-Allen, Geetika Tewari, Ankit Patel, Matt Welsh, and Radhika Nagpal. Firefly-inspired sensor network synchronicity with realistic radio effects. In

International Conference on Embedded Networked Sensor Systems, pages 142–153. ACM, 2005.

[149] Jennifer Yick, Biswanath Mukherjee, and Dipak Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292 – 2330, 2008.

[150] Hakan L Younes. Verification and planning for stochastic processes with asynchronous events. Technical report, Carnegie-Mellon University Pittsburgh PA School of Computer Science, 2005.

[151] Haidi Yue and Joost-Pieter Katoen. Leader election in anonymous radio networks: Model checking energy consumption. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, volume 6148 of *LNCS*, pages 247–261. Springer, 2010.

[152] Richard Zippel. Probabilistic algorithms for sparse polynomials. In *International Symposiumon on Symbolic and Algebraic Computation*, pages 216–226, London, UK, UK, 1979. Springer-Verlag.

Appendices

A Synchronisation PRISM Model

Listing 1: Generated PRISM code for the Mirollo and Strogatz synchronisation model with parameters $N = 4$, $T = 5$, $\epsilon = 0.1$, and $R = 1$.

```
//=============================================================================
// model name: mirollo-strogatz
//=============================================================================
dtmc

//-----------------------------------------------------------------------------
// parameters
//
// N       = 4
// T       = 5
// EPSILON = 0.1
// R       = 1
//-----------------------------------------------------------------------------

//=============================================================================
// rewards
//=============================================================================
//-----------------------------------------------------------------------------
// records power consumption in watt hours
//-----------------------------------------------------------------------------
```

```
rewards "power_consumption"
    assigned :
        (k_1 * 0.0000002778)
        + ((k_1) * 0.0000000333)
        + ((k_4 + k_3 + k_2) * 0.0000328333);
endrewards

//=============================================================================
// order parameter
//=============================================================================
const double unit_vector_x_1 = 1.000000000000;
const double unit_vector_y_1 = 0.000000000000;
const double unit_vector_x_2 = 0.309016994375;
const double unit_vector_y_2 = 0.951056516295;
const double unit_vector_x_3 = -0.809016994375;
const double unit_vector_y_3 = 0.587785252292;
const double unit_vector_x_4 = -0.809016994375;
const double unit_vector_y_4 = -0.587785252292;
const double unit_vector_x_5 = 0.309016994375;
const double unit_vector_y_5 = -0.951056516295;

formula unit_vector_x_avg_squared = pow((((unit_vector_x_1 * k_1) +
    (unit_vector_x_2 * k_2) + (unit_vector_x_3 * k_3) + (unit_vector_x_4 *
    k_4) + (unit_vector_x_5 * k_5)) / 4), 2);
formula unit_vector_y_avg_squared = pow((((unit_vector_y_1 * k_1) +
    (unit_vector_y_2 * k_2) + (unit_vector_y_3 * k_3) + (unit_vector_y_4 *
    k_4) + (unit_vector_y_5 * k_5)) / 4), 2);
formula order_parameter = pow(unit_vector_x_avg_squared +
    unit_vector_y_avg_squared, 0.5);

//=============================================================================
// constants
//=============================================================================
//-----------------------------------------------------------------------------
// infinity
//-----------------------------------------------------------------------------
```

```
52 const int INFINITY = 5;
53
54 //-------------------------------------------------------------------------
55 // broadcast failure probability
56 //-------------------------------------------------------------------------
57 const double mu;
58
59 //=========================================================================
60 // formulae
61 //=========================================================================
62 //-------------------------------------------------------------------------
63 // the total number of oscillators at any given time
64 //-------------------------------------------------------------------------
65 formula num_oscillators = k_1 + k_2 + k_3 + k_4 + k_5;
66
67 //-------------------------------------------------------------------------
68 // true after all oscillators have been assigned phases
69 //-------------------------------------------------------------------------
70 formula assigned = num_oscillators = 4;
71
72 //-------------------------------------------------------------------------
73 // true if all oscillators have the same phase
74 //-------------------------------------------------------------------------
75 formula synchronised = k_1 = 4 | k_2 = 4 | k_3 = 4 | k_4 = 4 | k_5 = 4;
76
77 //-------------------------------------------------------------------------
78 // message loss probabilities
79 //-------------------------------------------------------------------------
80 formula pr_fail_0_0 = 1;
81 formula pr_fail_1_0 = (pow(mu, 0) * pow(1 - mu, 1) * 1.0);
82 formula pr_fail_1_1 = (pow(mu, 1) * pow(1 - mu, 0) * 1.0);
83 formula pr_fail_2_0 = (pow(mu, 0) * pow(1 - mu, 2) * 1.0);
84 formula pr_fail_2_1 = (pow(mu, 1) * pow(1 - mu, 1) * 2.0);
85 formula pr_fail_2_2 = (pow(mu, 2) * pow(1 - mu, 0) * 1.0);
86 formula pr_fail_3_0 = (pow(mu, 0) * pow(1 - mu, 3) * 1.0);
87 formula pr_fail_3_1 = (pow(mu, 1) * pow(1 - mu, 2) * 3.0);
```

```
formula pr_fail_3_2 = (pow(mu, 2) * pow(1 - mu, 1) * 3.0);
formula pr_fail_3_3 = (pow(mu, 3) * pow(1 - mu, 0) * 1.0);
formula pr_fail_4_0 = (pow(mu, 0) * pow(1 - mu, 4) * 1.0);
formula pr_fail_4_1 = (pow(mu, 1) * pow(1 - mu, 3) * 4.0);
formula pr_fail_4_2 = (pow(mu, 2) * pow(1 - mu, 2) * 6.0);
formula pr_fail_4_3 = (pow(mu, 3) * pow(1 - mu, 1) * 4.0);
formula pr_fail_4_4 = (pow(mu, 4) * pow(1 - mu, 0) * 1.0);

//=============================================================================
// modules
//=============================================================================
//-----------------------------------------------------------------------------
// initial state indicator module
//-----------------------------------------------------------------------------
module initial_state_indicator
    is_initial_state : bool init false;
    is_non_initial_state : bool init false;

    [step] (!assigned) ->
        (is_initial_state' = true) &
        (is_non_initial_state' = false);
    [step] (assigned) ->
        (is_initial_state' = false) &
        (is_non_initial_state' = is_initial_state | is_non_initial_state);
endmodule

//-----------------------------------------------------------------------------
// oscillator population module
//-----------------------------------------------------------------------------
module oscillator_population
    //-------------------------------------------------------------------------
    // counters for oscillators in each state
    //-------------------------------------------------------------------------
    k_1 : [0..4] init 0;
    k_2 : [0..4] init 0;
    k_3 : [0..4] init 0;
```

```
124     k_4 : [0..4] init 0;
125     k_5 : [0..4] init 0;
126
127     //-------------------------------------------------------------------------------
128     // transitions from starting state
129     //-------------------------------------------------------------------------------
130     [step] (!assigned) ->
131         (1 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 4) +
132         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 3) +
133         (6 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 0) & (k_4' = 2) &
                (k_5' = 2) +
134         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 0) & (k_4' = 3) &
                (k_5' = 1) +
135         (1 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 0) & (k_4' = 4) &
                (k_5' = 0) +
136         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 3) +
137         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 1) & (k_4' = 1) &
                (k_5' = 2) +
138         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 1) & (k_4' = 2) &
                (k_5' = 1) +
139         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 1) & (k_4' = 3) &
                (k_5' = 0) +
140         (6 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 2) & (k_4' = 0) &
                (k_5' = 2) +
141         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 2) & (k_4' = 1) &
                (k_5' = 1) +
142         (6 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 2) & (k_4' = 2) &
                (k_5' = 0) +
143         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 3) & (k_4' = 0) &
                (k_5' = 1) +
144         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 3) & (k_4' = 1) &
                (k_5' = 0) +
```

```
145         (1 / pow(5, 4)): (k_1' = 0) & (k_2' = 0) & (k_3' = 4) & (k_4' = 0) &
                (k_5' = 0) +
146         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 3) +
147         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 2) +
148         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 0) & (k_4' = 2) &
                (k_5' = 1) +
149         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 0) & (k_4' = 3) &
                (k_5' = 0) +
150         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 2) +
151         (24 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 1) & (k_4' = 1) &
                (k_5' = 1) +
152         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 1) & (k_4' = 2) &
                (k_5' = 0) +
153         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 2) & (k_4' = 0) &
                (k_5' = 1) +
154         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 2) & (k_4' = 1) &
                (k_5' = 0) +
155         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 1) & (k_3' = 3) & (k_4' = 0) &
                (k_5' = 0) +
156         (6 / pow(5, 4)): (k_1' = 0) & (k_2' = 2) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 2) +
157         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 2) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 1) +
158         (6 / pow(5, 4)): (k_1' = 0) & (k_2' = 2) & (k_3' = 0) & (k_4' = 2) &
                (k_5' = 0) +
159         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 2) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 1) +
160         (12 / pow(5, 4)): (k_1' = 0) & (k_2' = 2) & (k_3' = 1) & (k_4' = 1) &
                (k_5' = 0) +
161         (6 / pow(5, 4)): (k_1' = 0) & (k_2' = 2) & (k_3' = 2) & (k_4' = 0) &
                (k_5' = 0) +
162         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 3) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 1) +
```

```
163         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 3) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 0) +
164         (4 / pow(5, 4)): (k_1' = 0) & (k_2' = 3) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 0) +
165         (1 / pow(5, 4)): (k_1' = 0) & (k_2' = 4) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 0) +
166         (4 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 3) +
167         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 2) +
168         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 2) &
                (k_5' = 1) +
169         (4 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 3) &
                (k_5' = 0) +
170         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 2) +
171         (24 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 1) & (k_4' = 1) &
                (k_5' = 1) +
172         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 1) & (k_4' = 2) &
                (k_5' = 0) +
173         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 2) & (k_4' = 0) &
                (k_5' = 1) +
174         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 2) & (k_4' = 1) &
                (k_5' = 0) +
175         (4 / pow(5, 4)): (k_1' = 1) & (k_2' = 0) & (k_3' = 3) & (k_4' = 0) &
                (k_5' = 0) +
176         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 2) +
177         (24 / pow(5, 4)): (k_1' = 1) & (k_2' = 1) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 1) +
178         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 1) & (k_3' = 0) & (k_4' = 2) &
                (k_5' = 0) +
179         (24 / pow(5, 4)): (k_1' = 1) & (k_2' = 1) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 1) +
180         (24 / pow(5, 4)): (k_1' = 1) & (k_2' = 1) & (k_3' = 1) & (k_4' = 1) &
                (k_5' = 0) +
```

```
181         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 1) & (k_3' = 2) & (k_4' = 0) &
                (k_5' = 0) +
182         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 2) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 1) +
183         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 2) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 0) +
184         (12 / pow(5, 4)): (k_1' = 1) & (k_2' = 2) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 0) +
185         (4 / pow(5, 4)): (k_1' = 1) & (k_2' = 3) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 0) +
186         (6 / pow(5, 4)): (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 2) +
187         (12 / pow(5, 4)): (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 1) +
188         (6 / pow(5, 4)): (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 2) &
                (k_5' = 0) +
189         (12 / pow(5, 4)): (k_1' = 2) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 1) +
190         (12 / pow(5, 4)): (k_1' = 2) & (k_2' = 0) & (k_3' = 1) & (k_4' = 1) &
                (k_5' = 0) +
191         (6 / pow(5, 4)): (k_1' = 2) & (k_2' = 0) & (k_3' = 2) & (k_4' = 0) &
                (k_5' = 0) +
192         (12 / pow(5, 4)): (k_1' = 2) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 1) +
193         (12 / pow(5, 4)): (k_1' = 2) & (k_2' = 1) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 0) +
194         (12 / pow(5, 4)): (k_1' = 2) & (k_2' = 1) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 0) +
195         (6 / pow(5, 4)): (k_1' = 2) & (k_2' = 2) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 0) +
196         (4 / pow(5, 4)): (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 1) +
197         (4 / pow(5, 4)): (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) &
                (k_5' = 0) +
198         (4 / pow(5, 4)): (k_1' = 3) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) &
                (k_5' = 0) +
```

```
199         (4 / pow(5, 4)): (k_1' = 3) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 0) +
200         (1 / pow(5, 4)): (k_1' = 4) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) &
                (k_5' = 0);
201
202     //-------------------------------------------------------------------------
203     // update oscillator phases, no flashes
204     //-------------------------------------------------------------------------
205     [step] (assigned & k_5 = 0) -> (k_2' = k_1) & (k_3' = k_2) & (k_4' = k_3)
            & (k_5' = k_4) & (k_1' = k_5);
206
207     //-------------------------------------------------------------------------
208     // update oscillator phases, flashes
209     //-------------------------------------------------------------------------
210     [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 0 & k_4 = 0 & k_5 = 4) ->
211             (k_1' = 4) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0);
212     [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 0 & k_4 = 1 & k_5 = 3) ->
213         ((pr_fail_1_0 * pr_fail_3_0) + (pr_fail_1_1 * pr_fail_3_0) +
                (pr_fail_1_0 * pr_fail_3_1) + (pr_fail_1_1 * pr_fail_3_1)):
214             (k_1' = 4) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0) +
215         ((pr_fail_3_2) + (pr_fail_3_3)):
216             (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 1);
217     [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 0 & k_4 = 2 & k_5 = 2) ->
218         ((pr_fail_2_0 * pr_fail_2_0) + (pr_fail_2_1 * pr_fail_2_0) +
                (pr_fail_2_2 * pr_fail_2_0)):
219             (k_1' = 4) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0) +
220         ((pr_fail_2_1) + (pr_fail_2_2)):
221             (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 2);
222     [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 0 & k_4 = 3 & k_5 = 1) ->
223             (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 3);
224     [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 1 & k_4 = 0 & k_5 = 3) ->
225         ((pr_fail_0_0 * pr_fail_3_0) + (pr_fail_0_0 * pr_fail_3_1)):
226             (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 1) +
227         ((pr_fail_3_2) + (pr_fail_3_3)):
228             (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) & (k_5' = 0);
229     [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 1 & k_4 = 1 & k_5 = 2) ->
```

```
        ((pr_fail_1_0 * pr_fail_2_0) + (pr_fail_1_1 * pr_fail_2_0)):
            (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 1) +
        ((pr_fail_2_1) + (pr_fail_2_2)):
            (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) & (k_5' = 1);
    [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 1 & k_4 = 2 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) & (k_5' = 2);
    [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 2 & k_4 = 0 & k_5 = 2) ->
        ((pr_fail_0_0 * pr_fail_2_0)):
            (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 0) & (k_5' = 2) +
        ((pr_fail_2_1) + (pr_fail_2_2)):
            (k_1' = 2) & (k_2' = 0) & (k_3' = 0) & (k_4' = 2) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 2 & k_4 = 1 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 2) & (k_5' = 1);
    [step] (assigned & k_1 = 0 & k_2 = 0 & k_3 = 3 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 0) & (k_4' = 3) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 1 & k_3 = 0 & k_4 = 0 & k_5 = 3) ->
        ((pr_fail_0_0 * pr_fail_3_0)):
            (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) & (k_5' = 0) +
        ((pr_fail_0_0 * pr_fail_3_1) + (pr_fail_3_2) + (pr_fail_3_3)):
            (k_1' = 3) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 1 & k_3 = 0 & k_4 = 1 & k_5 = 2) ->
        ((pr_fail_1_0 * pr_fail_2_0)):
            (k_1' = 3) & (k_2' = 0) & (k_3' = 0) & (k_4' = 1) & (k_5' = 0) +
        ((pr_fail_1_1 * pr_fail_2_0)):
            (k_1' = 3) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) & (k_5' = 0) +
        ((pr_fail_2_1) + (pr_fail_2_2)):
            (k_1' = 2) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) & (k_5' = 1);
    [step] (assigned & k_1 = 0 & k_2 = 1 & k_3 = 0 & k_4 = 2 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) & (k_5' = 2);
    [step] (assigned & k_1 = 0 & k_2 = 1 & k_3 = 1 & k_4 = 0 & k_5 = 2) ->
        ((pr_fail_0_0 * pr_fail_2_0)):
            (k_1' = 2) & (k_2' = 0) & (k_3' = 1) & (k_4' = 0) & (k_5' = 1) +
        ((pr_fail_2_1) + (pr_fail_2_2)):
            (k_1' = 2) & (k_2' = 0) & (k_3' = 1) & (k_4' = 1) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 1 & k_3 = 1 & k_4 = 1 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 1) & (k_4' = 1) & (k_5' = 1);
```

```
    [step] (assigned & k_1 = 0 & k_2 = 1 & k_3 = 2 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 1) & (k_4' = 2) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 2 & k_3 = 0 & k_4 = 0 & k_5 = 2) ->
            (k_1' = 2) & (k_2' = 0) & (k_3' = 2) & (k_4' = 0) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 2 & k_3 = 0 & k_4 = 1 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 2) & (k_4' = 0) & (k_5' = 1);
    [step] (assigned & k_1 = 0 & k_2 = 2 & k_3 = 1 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 2) & (k_4' = 1) & (k_5' = 0);
    [step] (assigned & k_1 = 0 & k_2 = 3 & k_3 = 0 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 0) & (k_3' = 3) & (k_4' = 0) & (k_5' = 0);
    [step] (assigned & k_1 = 1 & k_2 = 0 & k_3 = 0 & k_4 = 0 & k_5 = 3) ->
            (k_1' = 3) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0);
    [step] (assigned & k_1 = 1 & k_2 = 0 & k_3 = 0 & k_4 = 1 & k_5 = 2) ->
        ((pr_fail_1_0 * pr_fail_2_0) + (pr_fail_1_1 * pr_fail_2_0)):
            (k_1' = 3) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0) +
        ((pr_fail_2_1) + (pr_fail_2_2)):
            (k_1' = 2) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) & (k_5' = 1);
    [step] (assigned & k_1 = 1 & k_2 = 0 & k_3 = 0 & k_4 = 2 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) & (k_5' = 2);
    [step] (assigned & k_1 = 1 & k_2 = 0 & k_3 = 1 & k_4 = 0 & k_5 = 2) ->
        ((pr_fail_0_0 * pr_fail_2_0)):
            (k_1' = 2) & (k_2' = 1) & (k_3' = 0) & (k_4' = 0) & (k_5' = 1) +
        ((pr_fail_2_1) + (pr_fail_2_2)):
            (k_1' = 2) & (k_2' = 1) & (k_3' = 0) & (k_4' = 1) & (k_5' = 0);
    [step] (assigned & k_1 = 1 & k_2 = 0 & k_3 = 1 & k_4 = 1 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 1) & (k_3' = 0) & (k_4' = 1) & (k_5' = 1);
    [step] (assigned & k_1 = 1 & k_2 = 0 & k_3 = 2 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 1) & (k_3' = 0) & (k_4' = 2) & (k_5' = 0);
    [step] (assigned & k_1 = 1 & k_2 = 1 & k_3 = 0 & k_4 = 0 & k_5 = 2) ->
            (k_1' = 2) & (k_2' = 1) & (k_3' = 1) & (k_4' = 0) & (k_5' = 0);
    [step] (assigned & k_1 = 1 & k_2 = 1 & k_3 = 0 & k_4 = 1 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 1) & (k_3' = 1) & (k_4' = 0) & (k_5' = 1);
    [step] (assigned & k_1 = 1 & k_2 = 1 & k_3 = 1 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 1) & (k_3' = 1) & (k_4' = 1) & (k_5' = 0);
    [step] (assigned & k_1 = 1 & k_2 = 2 & k_3 = 0 & k_4 = 0 & k_5 = 1) ->
            (k_1' = 1) & (k_2' = 1) & (k_3' = 2) & (k_4' = 0) & (k_5' = 0);
```

```
302     [step] (assigned & k_1 = 2 & k_2 = 0 & k_3 = 0 & k_4 = 0 & k_5 = 2) ->
303             (k_1' = 2) & (k_2' = 2) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0);
304     [step] (assigned & k_1 = 2 & k_2 = 0 & k_3 = 0 & k_4 = 1 & k_5 = 1) ->
305             (k_1' = 1) & (k_2' = 2) & (k_3' = 0) & (k_4' = 0) & (k_5' = 1);
306     [step] (assigned & k_1 = 2 & k_2 = 0 & k_3 = 1 & k_4 = 0 & k_5 = 1) ->
307             (k_1' = 1) & (k_2' = 2) & (k_3' = 0) & (k_4' = 1) & (k_5' = 0);
308     [step] (assigned & k_1 = 2 & k_2 = 1 & k_3 = 0 & k_4 = 0 & k_5 = 1) ->
309             (k_1' = 1) & (k_2' = 2) & (k_3' = 1) & (k_4' = 0) & (k_5' = 0);
310     [step] (assigned & k_1 = 3 & k_2 = 0 & k_3 = 0 & k_4 = 0 & k_5 = 1) ->
311             (k_1' = 1) & (k_2' = 3) & (k_3' = 0) & (k_4' = 0) & (k_5' = 0);
312 endmodule
```