Incremental Verification of Parametric and Reconfigurable Markov Chains

Paul Gainer, Ernst Moritz Hahn, and Sven Schewe

University of Liverpool, UK
{P.Gainer,E.M.Hahn,Sven.Schewe}@liverpool.ac.uk

Abstract. The analysis of parametrised systems is a growing field in verification, but the analysis of parametrised probabilistic systems is still in its infancy. This is partly because it is much harder: while there are beautiful cut-off results for non-stochastic systems that allow to focus only on small instances, there is little hope that such approaches extend to the quantitative analysis of probabilistic systems, as the probabilities depend on the size of a system. The unicorn would be an automatic transformation of a parametrised system into a formula, which allows to plot, say, the likelihood to reach a goal or the expected costs to do so, against the parameters of a system. While such analysis exists for narrow classes of systems, such as waiting queues, we aim both lower—stepwise exploring the parameter space—and higher—considering general systems.
The novelty is to heavily exploit the similarity between instances of parametrised systems. When the parameter grows, the system for the smaller parameter is, broadly speaking, present in the larger system. We use this observation to guide the elegant state-elimination method for parametric Markov chains in such a way, that the model transformations will start with those parts of the system that are stable under increasing the parameter. We argue that this can lead to a very cheap iterative way to analyse parametric systems, show how this approach extends to reconfigurable systems, and demonstrate on two benchmarks that this approach scales.

1 Introduction

Probabilistic systems are everywhere, and their analysis can be quite challenging. Challenges, however, come in many flavours. They range from theoretical questions, such as decidability and complexity, through algorithms design and tool development, to the application of parametric systems. This paper is motivated by the latter, but melds the different flavours together.

We take our motivation from the first author's work on biologically inspired synchronisation protocols [8,9]. This application leaning work faced the problem of exploring a parameter space for a family of network coordination protocols, where interacting nodes achieve consensus on their local clocks by imitating the behaviour of fireflies [22]. Global clock synchronisation emerges from local interactions between the nodes, whose behaviour is that of coupled limit-cycle

oscillators. The method used was the same that we have seen applied by several practitioners from engineering and biology: adjust the parameters, produce a model, and use a tool like ePMC/IscasMC [12], PRISM [20], or Storm [6] to analyse it.

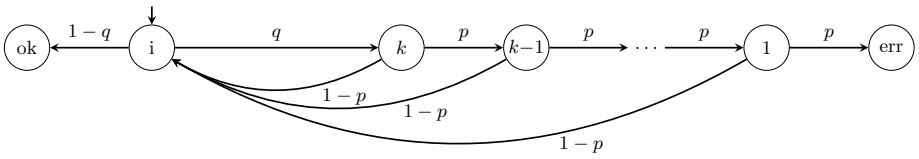
In the case of the synchronisation protocols, the parameters investigated were typical of those considered when evaluating the emergence of synchronisation in a network of connected nodes: the number of nodes forming the network, the granularity of the model (discrete length of an oscillation cycle), the strength of coupling between the oscillators, the likelihood of interactions between nodes being inhibited by some external factor, for instance message loss in a communication medium, and the length of the refractory period, an initial period in the oscillation cycle of a node where interactions with other nodes are ignored.

The reason to explore the parameter space can be manifold. Depending on the application, one might simply want to obtain a feeling of how the parameters impact on the behaviour. Another motivation is to see how the model behaves, compare it with observations, and adjust it when it fails to comply. Regardless of the reason to adjust the parameter, the changes often lead to very similar models.

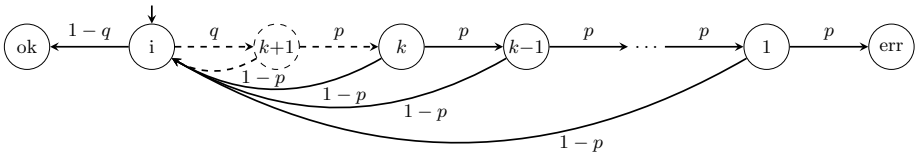
Now, if we want to analyse hundreds of similar models, then it becomes paramount to re-use as much of the analysis as possible. With this in mind, we have selected model checking techniques for safety and reachability properties of Markov chains that build on repeated state elimination [11] as the backbone of our verification technique. State elimination is a technique that successively changes the model by removing states. It works like the transformation from finite automata to regular expressions: a state is removed, and the new structure has all successors of this state as (potentially new) successors of the predecessors of this state, with the respectively adjusted probabilities (and, if applicable, expected rewards).

If these models are changed in shape and size when playing with the parameters, then these changes tend to be smooth: small changes of the parameters lead to small changes of the model. Moreover, the areas that change—and, 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 playing with her model, who would know full well which parts do or do not change when increasing a parameter. These insights inform the order in which the states are eliminated.

When, say, the increase of a parameter allows for re-using all elimination steps but the last two or three, then repeating the analysis is quite cheap. Luckily, this is typically the case in structured models, e.g. those who take a chain-, ring-, or tree-like shape that can be inductively defined. As a running example of a structured model we consider the Zeroconf protocol [3] for the autonomous configuration of multiple hosts in a network with unique IP addresses (Figure 1). A host that joins the network selects an address uniformly at random from a available addresses. If the network consists of h hosts, then the probability that the selected address is already in use is $q = \frac{h}{a}$.



(a)



(b)

Fig. 1: The Zeroconf Protocol for $n = k$ (a) and $n = k + 1$ (b).

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

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

The structure of the paper is as follows. We begin by comparing our work to similar approaches in Section 2. In Section 3, we introduce the novel algorithms for the analysis of reconfigured models. We then evaluate our approach on two different types of parametric models which are discussed in Section 4. Section 5 concludes the paper and outlines future work.

2 Related Work

Our work builds on previous results in the area of parametric Markov model checking and incremental runtime verification of stochastic systems.

Daws [4] considered Markov chains, which are parametric in the transition probabilities, but not in their graph structure. He introduced an algorithm to calculate the function that represents the probability of reaching a set of target states for all well-defined evaluations for a parametric Markov chain. For this, he interprets the Markov chain under consideration as a finite automaton, in which transitions are labelled with symbols that correspond to rational numbers or variables. He then uses state elimination [13] to obtain a regular expression for the language of the automaton. Evaluating these regular expressions into rational functions yields the probability of reaching the target states. One limiting factor of this approach is that the complete regular expression has to be stored in memory.

Hahn et al. introduced [11] and implemented [10] a simplification and refinement of Daws' algorithm. Instead of using regular expressions, they store rational functions directly. This has the advantage that possible simplifications of these functions, such as cancellation of common factors, can be applied on the fly. This allows memory to be saved. It also provides a more concise representation of the values computed to the user. They have also extended the scope of the approach from reachability, to additionally handle accumulated reward properties. Several works from RWTH Aachen have followed up on solution methods for parametric Markov chains [5,14,23]. This type of parametric model checking has been used in [2] to build a model-repair framework for stochastic systems and in [15,16,17] to reason about the robustness of robot controllers against sensor errors.

Our paper borrows some ideas from the work of Kwiatkowska et al. [21]. Their work considers MDPs that are labelled with parametric transition probabilities. The authors do not aim to compute a closed-form function that represents properties of a model, but rather at accelerating the computation of results for individual instantiations of parameter values. Rather than state elimination, they use value iteration and other methods to evaluate the model for certain parameter values. In doing so, they can for instance, re-use computations for different instantiations of parameters that only depend on the graph structure of the model that remains unchanged for different instantiations.

We also take inspiration from Forejt et al. [7], where the role of parameters is different. Forejt et al. describe a policy iteration-based technique to evaluate parametric MDPs. While they also considered parameters in [7] that can influence the model structure, they would exploit similarities to inform the search for the policy when moving from one parameter value to the next. The repeatedly called model checking of Markov chains, on the other hand, is not parametric. Our approach is therefore completely orthogonal, as we focus on the analysis of Markov chains. In more detail, Forejt et al. [7] would use an incremental approach to find a good starting point for a policy iteration approach for MDPs. The intuition there is that an optimal policy is likely to be good—if not optimal—on the shared part of an MDP that grows with a parameter. This approach has

the potential to find the policy in less steps, as less noise disturbs the search in smaller MDPs, but its main promise is to provide an excellent oracle for a starting policy. Moreover, in the lucky instances where the policy is stable, it can also happen that there is a part of the Markov chain, obtained by using a policy that builds on a smaller parameter, that is outside of the cone of influence of the changes to the model. In this case, not only the policy, but also its evaluation is stable under the parameter change.

3 Algorithms

We first describe the state elimination method of Hahn [11] for parametric Markov Chains (PMCs), and then introduce an algorithm that substantially reduces the cost of recomputation of the parametric reachability probability for a reconfigured PMC. First we give some general definitions. Given a function f we denote the domain of f by $\mathsf{Dom}(f)$. We use the notation $f \oplus f' = f \restriction_{\mathsf{Dom}(f) \setminus \mathsf{Dom}(f')} \cup f'$ to denote the overriding union of f and f'. Let $V = \{v_1, \ldots, v_n\}$ denote a set of variables over $\mathbb{R}$. A *polynomial* g over V is a sum of monomials

$$g(v_1, \ldots, v_n) = \sum_{i_1, \ldots, i_n} a_{i_1, \ldots, i_n} v_1^{i_1} \ldots v_n^{i_n},$$

where each $i_j \in \mathbb{N}$ and each $a_{i_1, \ldots, i_n} \in \mathbb{R}$. A *rational function* f over a set of variables V is a fraction $f(v_1, \ldots, v_n) = \frac{f_1(v_1, \ldots, v_n)}{f_2(v_1, \ldots, v_n)}$ of two polynomials f_1, f_2 over V. We denote the set of rational functions from V to $\mathbb{R}$ by $\mathcal{F}_V$.

Definition 1. *A* parametric Markov chain *(PMC) is a tuple $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$, where $\mathcal{S}$ is a finite set of states, s_0 is the initial state, $V = \{v_1, \ldots, v_n\}$ is a finite set of parameters, and $\mathbf{P}$ is the probability matrix $\mathbf{P} : \mathcal{S} \times \mathcal{S} \to \mathcal{F}_V$.*

A *path* ω of a PMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$ is a non-empty finite, or infinite, sequence $s_0, s_1, s_2, \ldots$ where $s_i \in \mathcal{S}$ and $\mathbf{P}(s_i, s_{i+1}) > 0$ for $i \geqslant 0$. We denote the i^{th} state of ω by $\omega[i]$, the set of all paths starting at state s by $\mathsf{Paths}(s)$, and the set of all finite paths starting in s by $\mathsf{Paths}_f(s)$. For a finite path $\omega_f \in \mathsf{Paths}_f(s)$ the *cylinder set* of ω_f is the set of all infinite paths in $\mathsf{Paths}(s)$ that share the prefix ω_f. The probability of taking a finite path $s_0, s_1, \ldots, s_n \in \mathsf{Paths}_f(s_0)$ is given by $\prod_{i=1}^{n} \mathbf{P}(s_{i-1}, s_i)$. This measure over finite paths can be extended to a probability measure Pr_s over the set of infinite paths $\mathsf{Paths}(s)$, where the smallest σ-algebra over $\mathsf{Paths}(s)$ is the smallest set containing all cylinder sets for paths in $\mathsf{Paths}_f(s)$. For a detailed description of the construction of the probability measure we refer the reader to [18].

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

Given a state s, we denote the set of all immediate predecessors and successors of s in the underlying graph of $\mathcal{D}$ by $\mathsf{pre}_\mathcal{D}(s)$ and $\mathsf{post}_\mathcal{D}(s)$, respectively, and we define the *neighbourhood* of s as $\mathsf{Neigh}(s) = s \cup \mathsf{pre}_\mathcal{D}(s) \cup \mathsf{post}_\mathcal{D}(s)$. We write $\mathsf{reach}_D(s, s')$ if s' is reachable from s in the underlying graph of D.

Algorithm 1 State Elimination

1: **procedure** StateElimination($\mathcal{D}, s_e$)
2: **requires:** A PMC $\mathcal{D}$ and s_e, a state to eliminate in $\mathcal{D}$.
3: **for all** $(s_1, s_2) \in \mathsf{pre}_{\mathcal{D}}(s_e) \times \mathsf{post}_{\mathcal{D}}(s_e)$ **do**
4: $\mathbf{P}(s_1, s_2) \leftarrow \mathbf{P}(s_1, s_2) + \mathbf{P}(s_1, s_e)\frac{1}{1-\mathbf{P}(s_e, s_e)}\mathbf{P}(s_e, s_2)$
5: **end for**
6: Eliminate($\mathcal{D}, s_e$) // *remove* s_e *and incident transitions from* $\mathcal{D}$
7: **return** $\mathcal{D}$
8: **end procedure**

3.1 State Elimination

The algorithm of Hahn [11] proceeds as follows, where the input is a PMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$ and a set of target states $T \subset \mathcal{S}$. Initially, preprocessing is applied and without loss of generality all outgoing transitions from states in T are removed and a new state s_t is introduced such that $\mathbf{P}(t, s_t) = 1$ for all $t \in T$. All states s, where s is unreachable from the initial state or T is unreachable from s, are then removed along with all incident transitions. A state s_e in $\mathcal{S} \setminus \{s_0, s_t\}$ is then chosen for elimination and Algorithm 1 is applied. Firstly, for every pair $(s_1, s_2) \in \mathsf{pre}_{\mathcal{D}}(s_e) \times \mathsf{post}_{\mathcal{D}}(s_e)$, the existing probability $\mathbf{P}(s_1, s_2)$ is incremented by the probability of reaching s_2 from s_1 via s_e. The state and any incident transitions are then eliminated from D. This procedure is repeated until only s_0 and s_t remain, and the probability of reaching T from s_0 is then given by $\mathbf{P}(s_0, s_t)$.

3.2 Reconfiguration

Recall that we are interested in the re-use of information when recalculating reachability for a reconfigured PMC. We can do this by choosing the order in which we eliminate states in the original PMC. The general idea is that, if the set of states where structural changes might occur is known a priori, then we can apply state elimination to all other states first. We say that states where structural changes might occur are *volatile* states.

Definition 3. *A* volatile *parametric Markov chain (VPMC) is a tuple* $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \mathsf{Vol})$ *where* $(\mathcal{S}, s_0, \mathbf{P}, V)$ *is a PMC and* $\mathsf{Vol} \subseteq \mathcal{S}$ *is a set of volatile states for* $\mathcal{D}$*.*

Given a VPMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \mathsf{Vol})$, we can define an *elimination ordering* for $\mathcal{D}$ as a bijection $\prec_{\mathcal{D}}\colon \mathcal{S} \to \{1, \ldots, |\mathcal{S}|\}$ that defines an ordering for the elimination of states in S, such that $\prec_{\mathcal{D}}(s) < \prec_{\mathcal{D}}(s')$ holds for all $s \in \mathcal{S} \setminus \mathsf{Vol}, s' \in \mathsf{Vol}$, where $\prec_{\mathcal{D}}(s) < \prec_{\mathcal{D}}(s')$ indicates that s is eliminated before s'. Observe that a volatile state in $\mathcal{D}$ is only eliminated after all non-volatile states.

Definition 4. *A* reconfiguration *for a VPMC* $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \mathsf{Vol})$ *is a PMC* $\mathcal{D}^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, V)$*, where* $\mathcal{S}^R$ *is a set of states with* $\mathcal{S}^R \cap \mathcal{S} \neq \{s_0\}$*,* s_0 *and*

V are the initial state and the finite set of parameters for $\mathcal{D}$. The reconfigured probability matrix $\mathbf{P}^R$ is a total function $\mathbf{P}^R : \mathcal{S}^R \times \mathcal{S}^R \to \mathcal{F}_V$ such that, for all $s, s' \in \mathcal{S}^R$ where $\mathbf{P}(s, s')$ is defined, $\mathbf{P}(s, s') \neq \mathbf{P}^R(s, s')$ implies $s, s' \in \mathsf{Vol}$.

Given a VPMC $\mathcal{D}$ and a reconfiguration $\mathcal{D}^R$ for $\mathcal{D}$ we say that a state s in $\mathcal{S}$ is *consistent* in $\mathcal{D}^R$ if s is also in $\mathcal{S}^R$, and the set of all predecessors and successors of s remains unchanged in $\mathcal{D}^R$ (that is $\mathsf{pre}_{\mathcal{D}}(s) = \mathsf{pre}_{\mathcal{D}^R}(s)$, $\mathsf{post}_{\mathcal{D}}(s) = \mathsf{post}_{\mathcal{D}^R}(s)$, $\mathbf{P}(s_1, s) = \mathbf{P}^R(s_1, s)$ for every $s_1 \in \mathsf{pre}_{\mathcal{D}}(s)$, and $\mathbf{P}(s, s_2) = \mathbf{P}^R(s, s_2)$ for every $s_2 \in \mathsf{post}_{\mathcal{D}}(s)$). We say that a state s in $\mathcal{S}$ is *reconfigured* in $\mathcal{D}^R$ if s is also in $\mathcal{S}^R$ and s is not consistent. Finally, we say that a state s in $\mathcal{S}^R$ is *introduced* in $\mathcal{D}^R$ if s is neither consistent nor reconfigured. We denote the sets of all consistent, reconfigured, and introduced states by $\mathsf{Con}(\mathcal{D}, \mathcal{D}^R), \mathsf{Rec}(\mathcal{D}, \mathcal{D}^R)$, and $\mathsf{Int}(\mathcal{D}, \mathcal{D}^R)$, respectively. Figure 2 shows the consistent, reconfigured, and introduced states for $\mathcal{D}$ and $\mathcal{D}^R$.

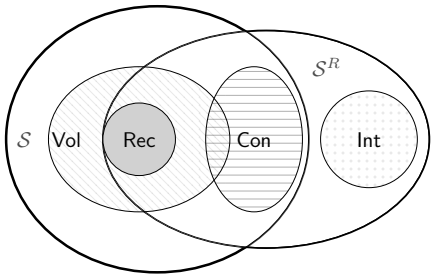


Fig. 2: Venn diagram showing the consistent, reconfigured, and introduced states for a VPMC $\mathcal{D}$ and reconfiguration $\mathcal{D}^R$.

Algorithm 2 computes the parametric reachability probability of some target state in a VPMC $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V, \mathsf{Vol})$ for a given elimination ordering for $\mathcal{D}$. Observe that we compute the reachability probability with respect to a single target state. The reachability of a set of target states can be computed by first removing all outgoing transitions from existing target states, and then introducing a new target state to which a transition is taken from any existing target state with probability 1. We introduce a new initial state to the model, from which a transition is taken to the original initial state with probability 1. The algorithm computes a partial probability matrix $\mathbf{P}'$, initialised as a zero matrix, that stores the probability of reaching s_2 from s_1 via any eliminated non-volatile state, where s_1, s_2 are either volatile states, the initial state, or the target state. It also computes an *elimination map* $m_{\mathcal{D}}^{\mathsf{Vol}}$, a function mapping tuples of the form (s_e, s_1, s_2), where s_e is an eliminated volatile state and s_1, s_2 are either volatile states, the initial state, or the target state, to the value computed during state elimination for the probability of reaching s_2 from s_1 via s_e. We are only interested in transitions between volatile states, the initial state, or the target state, since all non-volatile states in any reconfiguration of $\mathcal{D}$ will be eliminated first. Computed values for transitions to or from these states therefore serve no purpose once they have been eliminated.

Given a reconfiguration $\mathcal{D}^R = (\mathcal{S}^R, s_0, \mathbf{P}^R, V)$ for $\mathcal{D}$, an elimination ordering for $\mathcal{D}$, and the partial probability matrix and mapping computed using Algorithm 2, Algorithm 3 computes the parametric reachability probability for $\mathcal{D}^R$ as follows. Firstly the set of all non-volatile states of $\mathcal{D}$ and incident transitions are removed from $\mathcal{D}^R$, though state elimination itself does not occur. A set of

Algorithm 2 Parametric Reachability Probability for VPMCs

procedure ParametricReachability($\mathcal{D}$, $\prec_{\mathcal{D}}$, s_t)
  requires: a target state $s_t \in \mathcal{S}$, and for all $s \in \mathcal{S}$ it holds $\mathsf{reach}_{\mathcal{D}}(s_0, s)$ and $\mathsf{reach}_{\mathcal{D}}(s, s_t)$.
  $E \leftarrow \mathcal{S} \setminus \{s_0, s_t\}$ *// states to be eliminated from $\mathcal{D}$*
  $\mathbf{P'} \leftarrow 0_{|\mathcal{S}|,|\mathcal{S}|}$ *// partial probability matrix*
  $m_{\mathcal{D}}^{\mathsf{Vol}} \leftarrow \emptyset$ *// elimination map*
  while $E \neq \emptyset$ **do**
    $s_e \leftarrow \arg\min \prec_{\mathcal{D}}{\restriction_E}$
    for all $(s_1, s_2) \in \mathsf{pre}_{\mathcal{D}}(s_e) \times \mathsf{post}_{\mathcal{D}}(s_e)$ **do**
      $p = \mathbf{P}(s_1, s_e)\frac{1}{1-\mathbf{P}(s_e,s_e)}\mathbf{P}(s_e, s_2)$
      if $s_1 \in \mathsf{Vol} \cup \{s_0, s_t\}$ and $s_2 \in \mathsf{Vol} \cup \{s_0, s_t\}$ **then**
        if $s_e \notin \mathsf{Vol}$ **then**
          $\mathbf{P'}(s_1, s_2) \leftarrow \mathbf{P'}(s_1, s_2) + p$
        else
          $m_{\mathcal{D}}^{\mathsf{Vol}} \leftarrow m_{\mathcal{D}}^{\mathsf{Vol}} \oplus \{(s_e, s_1, s_2) \mapsto p\}$
        end if
      end if
      $\mathbf{P}(s_1, s_2) \leftarrow \mathbf{P}(s_1, s_2) + p$
    end for
    $\mathsf{Eliminate}(\mathcal{D}, s_e)$ *// remove s_e and incident transitions from $\mathcal{D}$*
    $E \leftarrow E \setminus \{s_e\}$
  end while
  return $(\mathbf{P}(s_0, s_t), \mathbf{P'}, m_{\mathcal{D}}^{\mathsf{Vol}})$
end procedure

infected states is then initialised to be the set of all states that are reconfigured in $\mathcal{D}^R$. Then, for every other remaining state that is not introduced in $\mathcal{D}^R$, if that state or its neighbours are not infected we treat this state as a non-volatile state. That is, we update $\mathbf{P'}$ with the corresponding values in $m_{\mathcal{D}}^{\mathsf{Vol}}$ and remove the state and its incident transitions without performing state elimination. If the state, or one of its neighbours, is infected then the probability matrix is updated such that all transitions to and from that state are augmented with the corresponding values in $\mathbf{P'}$. These entries are then removed from the mapping. Subsequently, state elimination (Algorithm 1) is applied, and the infected area is expanded to include the immediate neighbourhood of the eliminated state. Finally, state elimination is applied to the set of all remaining introduced states in $\mathcal{D}^R$.

Example 1. Consider again the Zeroconf models from Figures 1a and 1b. Let $Z_k = (\mathcal{S}, s_0, \mathbf{P}, V, \mathsf{Vol})$ be a VPMC for $n = k$, such that $\mathcal{S} = \{1, \ldots, k\} \cup \{s_0, \mathrm{i}, \mathrm{err}\}$, $V = \{p, q\}$, and $\mathsf{Vol} = \{\mathrm{i}, k\}$. We are interested in the parametric reachability probability of the state err. Note that preprocessing removes the state ok from Z_k since $\mathsf{reach}_{Z_k}(\mathrm{ok}, \mathrm{err})$ does not hold. Now define $\prec_{Z_k} = \{1 \mapsto 1, 2 \mapsto 2, \ldots, k \mapsto k, \mathrm{i} \mapsto k+1\}$ to be an elimination ordering for Z_k. State elimination then proceeds according to $\prec_{Z_k}$, and after the first $k-1$ states have

Algorithm 3 Parametric Reachability Probability for reconfigured VMPC

procedure ReconfiguredParametricReachability($\mathcal{D}$, $\mathcal{D}^R$, $\prec_{\mathcal{D}}$, $\mathbf{P}'$, $m_{\mathcal{D}}^{\mathsf{Vol}}$, s_t)
  requires: absorbing target state s_t such that $s_t \in \mathcal{S}$ and $s_t \in \mathcal{S}^R$, for all $s \in \mathcal{S}$ it holds $\mathsf{reach}_{\mathcal{D}^R}(s_0, s)$ and $\mathsf{reach}_{\mathcal{D}^R}(s, s_t)$, and for all $s' \in \mathcal{S}^R$ it holds $\mathsf{reach}_{\mathcal{D}^R}(s_0, s')$ and $\mathsf{reach}_{\mathcal{D}^R}(s', s_t)$.
  $M \leftarrow (\mathsf{Vol} \cap \mathcal{S}^R) \cup \{s_0, s_t\}$
  $\mathsf{Elim} \leftarrow \mathsf{Con}(\mathcal{D}, \mathcal{D}^R) \setminus M$
  $\mathsf{Eliminate}(\mathcal{D}^R, \mathsf{Elim})$ // *remove all* $s_e \in \mathsf{Elim}$ *and incident transitions from* $\mathcal{D}$
  $\mathsf{Elim} \leftarrow \mathsf{Vol} \cap \mathcal{S}^R$
  $\mathsf{Infected} \leftarrow \mathsf{Rec}(\mathcal{D}, \mathcal{D}^R)$
  $\mathbf{P}^R(s_0, s_t) = \mathbf{P}'(s_0, s_t)$
  while $\mathsf{Elim} \neq \emptyset$ **do**
    $s_e \leftarrow \arg\min \prec_{\mathcal{D}} \restriction_{\mathsf{Elim}}$
    if $\mathsf{Infected} \cap \mathsf{Neigh}(s_e) = \emptyset$ **then**
      for all $(s'_e, s_1, s_2) \in \mathsf{Dom}(m_{\mathcal{D}}^{\mathsf{Vol}} \restriction_{\{s_e\} \times M^2})$ **do**
        $\mathbf{P}'(s_1, s_2) \leftarrow \mathbf{P}'(s_1, s_2) + m_{\mathcal{D}}^{\mathsf{Vol}}(s'_e, s_1, s_2)$
      end for
      $\mathsf{Eliminate}(\mathcal{D}^R, s_e)$ // *remove* s_e *and incident transitions from* $\mathcal{D}^R$
    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
      $\mathcal{D}^R \leftarrow \mathsf{StateElimination}(\mathcal{D}^R, s_e)$
      $\mathsf{Infected} \leftarrow \mathsf{Infected} \cup \mathsf{Neigh}(s_e)$
    end if
    $\mathsf{Elim} \leftarrow \mathsf{Elim} \setminus \{s_e\}$
  end while
  for all $s_e \in \mathsf{Int}(\mathcal{D}, \mathcal{D}^R)$ **do**
    $\mathcal{D}^R \leftarrow \mathsf{StateElimination}(\mathcal{D}^R, s_e)$
  end for
  return $\mathbf{P}^R(s_0, s_t)$
end procedure

been eliminated we have

$$\mathbf{P}'(k, \mathrm{err}) = p^k, \quad \mathbf{P}'(k, \mathrm{i}) = \sum_{j=1}^{k-1} (p^j - p^{j+1}).$$

Eliminating the remaining volatile states k and i then yields

$$m_{Z_k}^{\mathsf{Vol}} = \{(k, \mathrm{i}, \mathrm{err}) \mapsto qp^k, (k, \mathrm{i}, \mathrm{i}) \mapsto q(1 - p^k), (\mathrm{i}, s_0, \mathrm{err}) \mapsto \frac{qp^k}{1 - q(1 - p^k)}\}.$$

Now let $Z_{k+1} = (\mathcal{S}^R, s_0, \mathbf{P}^R, V)$ be a reconfiguration for Z_k such that $\mathcal{S}^R = \mathcal{S} \cup \{k{+}1\}$. We then have $\mathsf{Con}(Z_k, Z_{k+1}) = \{1 \ldots k{-}1\} \cup \{s_0, \mathrm{err}\}$, $\mathsf{Rec}(Z_k, Z_{k+1}) =$

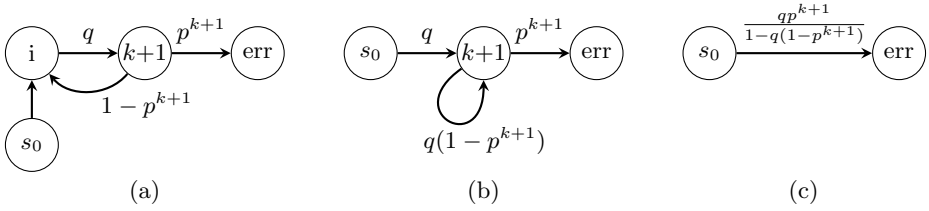


Fig. 3: Z_{k+1} after the elimination of states k (a), i (b), and $k+1$ (c).

$\{k, i\}$, and $\mathsf{Int}(Z_k, Z_{k+1}) = \{k+1\}$. 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 we update the probability matrix as follows,

$$\begin{aligned}
\mathbf{P}^R(k, \mathrm{err}) &\leftarrow \mathbf{P}^R(k, \mathrm{err}) + \mathbf{P}'(k, \mathrm{err}) \\
&\leftarrow 0 + p^k = p^k, \\
\mathbf{P}^R(k, \mathrm{i}) &\leftarrow \mathbf{P}^R(k, \mathrm{i}) + \mathbf{P}'(k, \mathrm{i}) \\
&\leftarrow (1-p) + \sum_{j=1}^{k-1} (p^j - p^{j+1}) = \sum_{j=0}^{k-1} (p^j - p^{j+1}) = 1 - p^k.
\end{aligned}$$

State elimination is then applied to state k and the corresponding entries in $\mathbf{P}'$ are set to zero. The state of the model after this step is shown in Figure 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 3b. Finally, state elimination is applied to the single introduced state $k+1$, resulting in the model shown in Figure 3c, and the algorithm terminates.

3.3 Correctness

The correctness of the approach follows as an easy corollary from the correctness of Hahn's general state elimination approach [11]. We outline the simple inductive argument, starting with the first parameter under consideration—which serves as the induction basis—and then look at incrementing the parameter value—which serves as the induction step.

For the induction basis, the first parameter considered, there is really nothing to show: we would merely choose a particular order in which states are eliminated, and the correctness of Hahn's state-elimination approach does not depend on the order in which states are eliminated.

For the induction step, consider that we have an order for one parameter value, and that we have an execution of the state elimination along this given order $<_o$. Our approach then builds a new order for the next parameter value. The new order $<_n$ is quite closely linked to the old order $<_o$, but for correctness, a very weak property suffices.

To prepare our argument, let us consider a set E of states with the following properties: the neighbourhood of E is the same in the Markov chains for the old and new parameter; the restriction of $<_o$ and $<_n$ to E define the same order; and E is the set of smallest states w.r.t. $<_o$ and $<_n$ ($s \in E$ and $(s' <_o s \vee s' <_n s)$ implies $s' \in E$). In this case, the initial sequence of the first $|E|$ reductions for the new Markov chain (along $<_n$) are the same as the first $|E|$ state eliminations along the old Markov chain (along $<_o$). Consequently, these elimination steps can be re-used, rather than re-done.

In Algorithm 3 we require less: we still require that the neighbourhood of E is the same in the Markov chains for the old and new parameter and the restriction of $<_o$ and $<_n$ to E define the same order, but relax the third requirement to $s \in E$ and $(s' <_o s \vee s' <_n s)$ implies that $s' \in E$ or s' is no neighbour of s. The result is the same: for the states in E, the $|E|$ state eliminations for the new Markov chain (along $<_n$) are the same as $|E|$ state eliminations along the old Markov chain (along $<_o$). Consequently, these elimination steps can be re-used.

3.4 Extension to Parametric Markov Reward Models

We now describe how we can extend the algorithms to PMCs annotated with rewards.

Definition 5. *A Parametric Markov Reward Model (PMRM) is a tuple $\mathcal{R} = (\mathcal{D}, r)$ where $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$ is a PMC and $r : \mathcal{S} \to \mathcal{F}_V$ is the reward function.*

The reward function labels states in $\mathcal{R}$ with a rational function over V that corresponds to the reward that is gained if that state is visited. Given a PMRM $\mathcal{R} = (\mathcal{D}, r)$ with $\mathcal{D} = (\mathcal{S}, s_0, \mathbf{P}, V)$, we are interested in the *parametric expected accumulated reward* [19] until some target state $s_t \in \mathcal{S}$ is reached. This is defined as the expectation of the random variable $X^{\mathcal{R}} : \mathsf{Paths}(s_0) \to \mathbb{R} \cup \{\infty\}$ over the infinite paths of $\mathcal{R}$. Given the set $\omega_{s_t} = \{i \mid w[i] = s_t\}$ we define

$$X^{\mathcal{R}}(\omega) = \begin{cases} \infty & \text{if } \omega_{s_t} = \emptyset \\ \sum_{i=0}^{k-1} r(\omega[i]) & \text{otherwise, where } k = \min \omega_{s_t}, \end{cases}$$

and define the expectation of $X^{\mathcal{R}}$ with respect to Pr_{s_0} as

$$E[X^{\mathcal{R}}] = \sum_{\omega \in \mathsf{Paths}(s_0)} X^{\mathcal{R}}(\omega) \mathsf{Pr}_{s_0}(\omega).$$

We extend our notion of volatility to PMRMs as follows. We say that a state is volatile if structural changes might occur in that state *or* if the reward labelling that state might change. Because of space limitations we omit the full definitions for volatile PMRMs, but the constructions are straightforward. Algorithms 1 to 3 are extended to incorporate rewards. For Algorithm 1, in addition to updating the probability matrix for the elimination of some state s_e, we also update the reward function as follows,

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

The updated value for $r(s_1)$ reflects the reward that would be accumulated if a transition would be taken from s_1 to s_e, where the expected number of self transitions would be $\frac{\mathbf{P}(s_e,s_e)}{1-\mathbf{P}(s_e,s_e)}$. Algorithm 2 then constructs additional mappings to record these computed expected reward values, which are then used for reconfiguration in Algorithm 3.

4 Case Studies

We provide a prototypical implementation[1] of the technique and define the metric that we will use for the evaluation of different models to be the total number of arithmetic operations performed for the elimination of all states in a model. Our implementation serves only to illustrate the potential of the method, and we will integrate the technique into the probabilistic model checker ePMC [12].

Due to space limitations we restrict our analysis to two classes of models. Firstly we consider the family of Zeroconf protocols described in Section 1, and secondly we consider a family of models used for the analysis of biologically inspired firefly synchronisation protocols—the class of protocols that inspired this work.

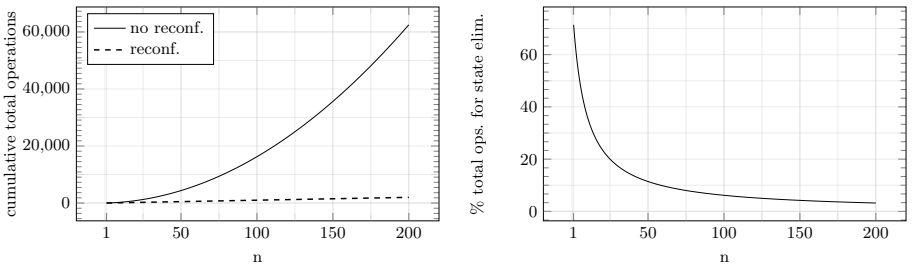
4.1 Zeroconf

We are interested in the reachability of the error state for the family of Zeroconf models, parameterised in the number n of attempts, after which the protocol will (potentially incorrectly) assume that it has selected a unique address. The initial model for $n = 1$ is defined, its volatile region is determined as in Example 1, and Algorithm 2 is applied. In each incremental step we increment n and apply Algorithm 3 to the model. Volatile states can be identified in each step.

Figures 4a and 4b show the total number of performed arithmetic operations accumulated during the incremental analysis of the models and the ratio of the number of arithmetic operations performed for regular state elimination, respectively. This ratio shows the small share of the number of iterations required when the values are calculated for a range of parameters in our approach (repeated applications of Algorithm 3), when compared to the naïve approach to re-calculate all values from scratch (applying Algorithm 2).

Figure 4a shows that the total number of operations is quadratic in the parameter when regular state elimination (applying Algorithm 2) is repeatedly applied from scratch. This is a consequence of the number of operations for *each* parameter being linear in the parameter value when naïvely applying Algorithm 2. This is in stark contrast to the number of operations needed when the parameter is stepwise incremented using Algorithm 3, stepwise capitalising on the analysis of the respective predecessor model. Here the update cost is constant: since the extent of structural change at each step is constant. This leads to dramatic savings (quadratic vs. linear) when exploring the parameter space, as illustrated by Figure 4b.

[1] `https://github.com/PaulGainer/PMC`



(a) (b)

Fig. 4: Cumulative total of arithmetic operations performed for iterative analysis of Zeroconf for $n = 1 \dots 200$ (a), and the ratio of total operations for reconfiguration to total operations for regular state elimination, given as a percentage (b).

4.2 Oscillator Synchronisation

We now consider the models developed in [8] and [9] to analyse protocols for the clock synchronisation of nodes in a network. In these protocols, consensus on clock values emerges from interactions between the nodes. The underlying mathematical model is that of coupled oscillators. This family of models is parametric in the number N of nodes that form the network; the granularity T of the discretisation of the oscillation cycle; the length R of the refractory period, during which nodes ignores interactions with their neighbours; the strength 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. Our prototypical implementation only considers low-level models defined explicitly as a set of states and a transition matrix, which trivialises the identification of volatile areas. Future implementation into ePMC, however, will allow volatile states to be clearly identified by analysing the guards present in high-level model description languages [1]. This works in particular for the parameters N and T we have studied.

Changing the parameter ϵ results in such severe changes in the structure of the model that we do not see how the synergistic effects we have observed can be ported to analysing its parameter space, while changing μ does not change the structure of the underlying graph and hence is not interesting for what we want to show.

In this paper, we therefore focus on the incremental analysis for the parameter R. We arbitrarily fix N to be 5 and ϵ to be 0.1, and repeat the incremental

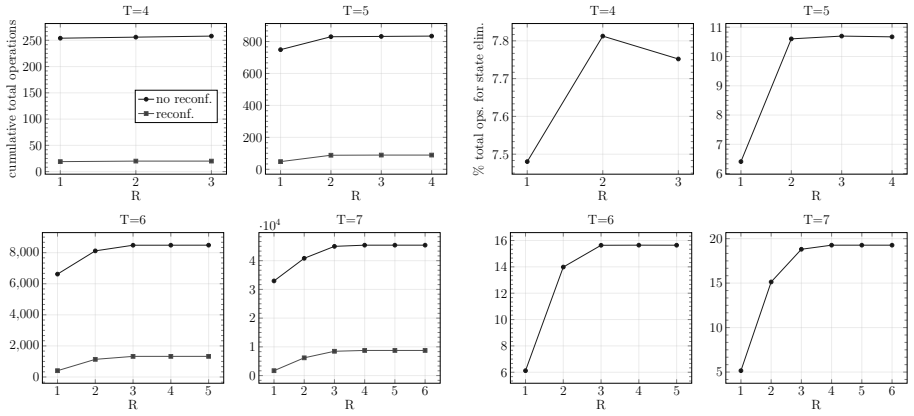


Fig. 5: Cumulative total of arithmetic operations performed for iterative analysis of synchronisation models with respect to R (a), and the ratio of totals for reconfiguration to totals for regular state elimination given as a percentage (b).

analysis for four different values for T. The parameter R varies from 1 to T (for each of the different values of T we have considered).

Figures 5a and 5b show the total number of performed arithmetic operations and the ratio of the totals for regular state elimination to the totals for reconfiguration given as a percentage, respectively. The effectiveness of the approach lessens as T increases, a result of the rounding of real values to discrete integer values that occurs when generating the transitions for the initial model [8]. Higher values of T result in an increase in the number of possible successor states for global states of the network, which in turn leads to an increase in the number of transitions in the model. Similarly, incrementing R results in reduced effectiveness as fewer interactions between nodes are ignored, and again more transitions are introduced to the model.

Overall it is clear that, while still substantial, the gains here are not as pronounced as those seen for the analysis of the Zeroconf protocol. This is to be expected, since the structural changes induced by changing the parameter R are not constant for each iteration—the higher the value of R the greater the extent of the structural changes incurred.

5 Conclusion and Future Work

It is clear—and, in hindsight, unsurprising—that our approach works well for structured Markov chains, such as chain-, ring-, or tree-like structures. Our experiments have lent evidence to this by showing that where the cost of model-checking an individual model grows linearly with a parameter, model checking up to a parameter becomes linear in the maximal parameter considered, whereas

the overall costs grow quadratically if all models are considered individually. Thus, we expect significant gain wherever changes can be localised and isolated. Moreover, we expect this to be the norm rather than the exception. After all, chains, rings, and trees are common structures in models.

It is quite striking that *very* specialised structures have enjoyed a lot of attention, and so have *absolutely* general ones. The standard example for very specialised structures is waiting queues. Fixed length waiting queues, for example, have closed form solutions. Thus, when the system analyst creates a structure, which is so standard that it has a known closed form solution *and*—and this is a big 'and'—realises that this is the case and looks up the closed form solution, then this analysis is the unicorn. However, if the structure is slightly different, if she fails to see that the problem has a closed form solution, or if she does not want to invest the time to research the closed form solution, then she would currently have to fall back to the naïve solution. Here our technique is a nice sweet spot between these extremes: the speed is close to evaluating closed form solutions, but applying our method does not put any burden on the system analyst who creates the parametrised model.

The limitations of our model are that it loses much of its advantage when a change in a parameter induces severe structural changes in the model. For the synchronisation protocol, some parameters severely change the structure. This is because most of the nodes are connected by an edge, and for such dense graphs, structural changes can have a huge cone of influence. In the worst case, e.g. a fully connected graph, a cubic overhead is incurred [10].

The next step of our work will be to tap the full potential of our approach by integrating it into the probabilistic model checker ePMC [12]. Here the symbolic description of the system will expose the volatile areas and—more importantly—the non-volatile areas that appear to be stable under successive increments of the parameter values. We also expect to obtain synergies by combining our method with the approach of [7], extending our approach to models with nondeterminism, such as interactive Markov chains and Markov decision processes.

6 Acknowledgements

This work was supported by the Sir Joseph Rotblat Alumni Scholarship at Liverpool, EPSRC grants EP/M027287/1 and EP/N007565/1, and by the Marie Skłodowska Curie Fellowship *Parametrised Verification and Control*.

References

1. Alur, R., Henzinger, T.A.: Reactive modules. Formal methods in system design 15(1), 7–48 (1999)
2. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: TACAS. pp. 326–340 (2011)
3. Bohnenkamp, H., van der Stok, P., Hermanns, H., Vaandrager, F.: Cost-optimization of the IPv4 zeroconf protocol, pp. 531–540. IEEE Computer Society Press (2003)

4. Daws, C.: Symbolic and parametric model checking of discrete-time markov chains. In: International Colloquium on Theoretical Aspects of Computing. pp. 280–294. Springer (2004)
5. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J., Ábrahám, E.: PROPhESY: A probabilistic parameter synthesis tool. In: CAV. pp. 214–231 (2015)
6. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: A modern probabilistic model checker. In: CAV. pp. 592–600. Springer (2017)
7. Forejt, V., Kwiatkowska, M., Parker, D., Qu, H., Ujma, M.: Incremental runtime verification of probabilistic systems. In: International Conference on Runtime Verification. pp. 314–319. Springer (2012)
8. Gainer, P., Linker, S., Dixon, C., Hustadt, U., Fisher, M.: Investigating parametric influence on discrete synchronisation protocols using quantitative model checking. In: QEST. pp. 224–239. Springer (2017)
9. Gainer, P., Linker, S., Dixon, C., Hustadt, U., Fisher, M.: The power of synchronisation: Formal analysis of power consumption in networks of pulse-coupled oscillators. arXiv preprint arXiv:1709.04385 (2017)
10. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: Param: A model checker for parametric markov models. In: CAV. pp. 660–664. Springer (2010)
11. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric markov models. STTT 13(1), 3–19 (2011)
12. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscas m c: a web-based probabilistic model checker. In: FM. pp. 312–317. Springer (2014)
13. Hopcroft, J.E.: Introduction to automata theory, languages, and computation. Pearson Education India (2008)
14. Jansen, N., Corzilius, F., Volk, M., Wimmer, R., Ábrahám, E., Katoen, J., Becker, B.: Accelerating parametric probabilistic verification. In: QEST. pp. 404–420 (2014)
15. Johnson, B., Kress-Gazit, H.: Probabilistic analysis of correctness of high-level robot behavior with sensor error. In: Robotics: Science and Systems (2011)
16. Johnson, B., Kress-Gazit, H.: Probabilistic guarantees for high-level robot behavior in the presence of sensor error. Autonomous Robots 33(3), 309–321 (2012)
17. Johnson, B.L.: Synthesis, analysis, and revision of correct-by-construction controllers for robots with sensing and actuation errors. Ph.D. thesis, Cornell University (2015)
18. Kemeny, J.G., Snell, J.L., Knapp, A.W.: Denumerable Markov chains: with a chapter of Markov random fields by David Griffeath, vol. 40. Springer Science & Business Media (2012)
19. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: International School on Formal Methods for the Design of Computer, Communication and Software Systems. pp. 220–270. Springer (2007)
20. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591. Springer (2011)
21. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for markov decision processes. In: International Conference on Dependable Systems & Networks. pp. 359–370. IEEE (2011)
22. Mirollo, R.E., Strogatz, S.H.: Synchronization of pulse-coupled biological oscillators. SIAM Journal on Applied Mathematics 50(6), 1645–1662 (1990)
23. Quatmann, T., Dehnert, C., Jansen, N., Junges, S., Katoen, J.: Parameter synthesis for markov models: Faster than ever. In: ATVA. pp. 50–67 (2016)