Combining Decomposition and Reduction for State Space Analysis of a Self-Stabilizing System

Nils Müllner, Oliver Theel, Martin Fränzle
Department of Computer Science
Carl von Ossietzky University of Oldenburg
D-26111 Oldenburg, Germany
Email: nils.muellner|oliver.theel|martin.fraenzle@informatik.uni-oldenburg.de

***Abstract*—Verifying fault tolerance properties of a distributed system can be achieved by state space analysis via Markov chains. Yet, the power of such exact analytic methods is confined by exponential growth of the chain's state space in the size of the system modeled. We propose a method that alleviates this limit. Lumping is a well known reduction technique that can be applied to a Markov chain to prune redundant information. We propose a system decomposition to employ lumping piecewise on the considerably smaller Markov chains of the subsystems which are much more likely to be tractable. Recomposing the lumped Markov chains of the subsystems results in a state space that is likely to be considerably smaller. An example demonstrates how the limiting window availability (i.e. a fault tolerance property) can be computed for a system while exploiting the combination of lumping and decomposition.**

I. Introduction

Fault tolerance is a desirable property in distributed systems. System engineers can access a repertoire of *detectors* and *correctors* to make a distributed system fault tolerant. While detectors reveal corruptions, correctors let the system recover. A system becomes fail-safe when employing fault detectors only (before violating its safety predicate it shuts down). It becomes non-masking fault tolerant with correctors only (before violating its liveness predicate it exposes the user to undesired behavior). When a system is equipped with both, detectors and correctors, it becomes *masking fault tolerant* (for certain faults). While corruptions are detected, the system's service can be withheld from the system user until the system fully recovers. Detectors and correctors both commonly employ redundancy, either spatial (e.g. parity bits, checksums or other techniques from the coding theory domain) or temporal (e.g. self-stabilization) or a combination thereof (e.g. retry after detection). To ensure system liveness, the system must not recover indefinitely. To prevent starvation, the time for recovery must be limited. The relation between time for recovery and the probability of successful recovery is defined by the *limiting window availability* (*LWA*) [1]. Intuitively, allowing more time for recovery leads to a higher probability of a successful recovery.

a) Focus on Correctors via Temporal Redundancy: To keep the focus on the temporal dimension that accounts for the liveness property, an appropriate test case must utilize solely temporal redundancy to exclude external effects that arise when multiple sources of redundancy are exploited for detectors and correctors. Temporal redundancy can be employed either for detectors or correctors or a combination thereof. In order to compute the effects of time on recovery, the time used for redundancy must only be spent for correction. One suitable example, where only *temporal correction* takes place, can be found in self-stabilizing systems. While the system is able to recover from *transient faults* by correction, it cannot tell if it works according to its specification or not. It lacks fault detection. The only redundancy required by self-stabilization is time (i.e. if space requirements for additional code for self-stabilization are neglected). To measure the *LWA* of a self-stabilizing system, a fault detector is added, mounted between the system and its user. It passes inquiries from the user to the system. If the system response is assumed to be correct, the system response is passed to the user. Otherwise, the fault detector blocks the response and retries the inquiry. As discussed above, the number of retries is to be limited. For simplification, we assume a perfect fault detector (i.e. no false positives, no false negatives).

b) Markov chains, Lumping and Decomposition: The *LWA* can be computed by state space analysis. As the processes in a distributed system communicate, the effects of faults propagate through the system. The state space analysis computing the *LWA* considers all states that the system can be in as well as the transitions between them. The probabilities with which processes execute a computational step or a fault step define the transition probabilities between the states. The result is a discrete time Markov chain (DTMC) $\mathcal{D}$. The Markov chain reduction with lumping was presented in [1]. When a set of processes of a distributed system behaves equal in the sense that processes have the same effect on the system, the information *which* of these processes is in a certain condition is redundant. The information that *one of them* is in a certain condition suffices here (i.e. in the light of computing the *LWA*). Pruning this redundant information by lumping the relevant states can reduce the state space tremendously. As this lumping results in a strongly probabilistic bisimilar DTMC, the lumped DTMC does still compute the exact *LWA*. In this paper we propose system decomposition to apply lumping piecewise on the subsystems. Thereby, the necessity to consider the whole DTMC at once for reduction becomes obsolete. Piecewise lumping facilitates lumping and analysis of considerably larger systems than tractable without decomposition.

c) Classification of Fault Propagation: Fault propagation plays an important role for the system decomposition. We distinguish three classes of fault propagation: i) hierarchical, ii) semi-hierarchical and iii) heterarchical.

- In a hierarchical system, a designated root process is the accepted leader among the processes of a system and does not rely on any other process. The processes are semi-ordered according to their distance to this root. Each non-root process only accepts information from processes that are closer to the root than itself. Therefore, the effects of faults strictly propagate from the leader towards the processes that have maximal distance *hierarchically*.
- Semi-hierarchical systems follow the same principle, but the role of the designated leader might switch. Thereby, each case of possible leader is to be considered distinctively. Furthermore, faults can propagate in *any* direction during leader change.
- In heterarchical systems, all processes are continuously peers. The effects of faults propagate in any direction at any time.

In this paper we exploit the benefits of hierarchical fault propagation to introduce the proposed combination of lumping and decomposition. Hierarchical fault propagation transforms a system's architectural graph into a directed acyclic graph (where processes communicate only according to the hierarchy). When decomposing such a system, the property of *unidirectional fault propagation* becomes an asset. The main contributions of this paper are a decomposition and piecewise lumping scheme alleviating the burden of state space explosion while maintaining a strong probabilistic bisimulation, plus its formal correctness argument (Proof 2). After a suitable system model is introduced and related work is discussed in Section II, Section III shows how lumping and decomposition can be combined. Section IV demonstrates the proposed method when computing the LWA for a self-stabilizing system. Section V concludes this work and gives a short outlook on future advances.

II. Preliminaries and Related Work

Section II-A introduces the system model and Section II-B discusses related work.

A. Preliminaries

A set Π comprises n processes $\{\pi_1, \dots, \pi_n\}$ that are connected via bidirectional communication channels. Two processes that are connected are called neighbors. Each process has one memory register r_i and can read its neighbors' registers and write to its own register. The system state s_t at time t is the snapshot of the assigned values over all registers $s_t = \langle r_1, \dots, r_n \rangle$.

As a canonical simplification, the state space is abstracted with a three value based logic: $r_i = 0$ means $r_i \models \mathcal{P}$ (we say process π_i stores an intended value, it is in a *correct* state, or the register r_i satisfies the safety predicate $\mathcal{P}$). The case when $r_i \not\models \mathcal{P}$ further distinguishes two instances. Either $r_i = 1$ when π_i is provided with contradicting information (it reads both 0 and 2 values from neighbors). Then, it *knowingly* cannot compute a correct value. And $r_i = 2$ if it *unknowingly* stores an incorrect value, either as the consequence of a direct fault or by transient fault propagation[1]. The system satisfies the global safety predicate $s_t \models \mathcal{P}$ iff $\forall \pi_i \in \Pi : r_i = 0$.

The system executes the self-stabilizing broadcast algorithm introduced in [1, p.24]. It is self-stabilizing and establishes hierarchical fault propagation. The algorithm is described informally here. Process π_1 is selected for the distinguished *root* process. We assume that faults only perturb the register of the executing processes and that a probabilistic scheduler (equiprobably) selects processes under serial execution semantics to take execution steps. This means, that each time step exactly one process is randomly selected to take an execution step with probability $e = \frac{1}{n}$. With probability p it executes a computational step (i.e. as desired) and with probability $q = 1 - p$ it takes a fault step (i.e. the process' register is corrupted by a fault). In the latter case the executing process writes a 2 to its register.

Under a probabilistic scheduler and fault model, the system's DTMC $\mathcal{D}$ is ergodic. Hence, regardless of its initial state, $\mathcal{D}$ reaches the same stationary distribution. This stationary distribution is used as initial probability distribution for computing the LWA, that is, the probability that the system either satisfies the safety predicate $\mathcal{P}$ in the steady state or within at most w time steps after for at least one computation step.

Definition 1

$$LWA_w = prob\{\exists k, 0 \leq k \leq w : s_k \models \mathcal{P}\} \tag{1}$$

B. Related Work

The decomposition extends our earlier work [1] on lumping in the light of fault tolerance analysis. Finite Markov chains and lumping are introduced by Kemeny and Snell [2] in a general manner, neither with exploiting symmetries of fault tolerant systems for lumping, nor with a focus on system decomposition. Weak probabilistic bisimulations on various models like Markov chains and Petri nets are discussed in Mertsiotakis' PhD thesis [3] that reveals how lumping can be exploited until a model becomes tractable for analysis. Nevertheless, lumping while maintaining strong probabilistic bisimilarity could exploit symmetric properties like fault propagation that were not discussed there. Focusing on performance analysis, Capra et al. [4] apply lumping to reduce Markov chains but without system decomposition. Derisavi contributed on the optimality of lumping [5], [6], [7], [8] but with a general focus. Kulkarni [9] introduced a compositional approach to fault tolerant design in his PhD thesis. In this approach, a masking fault tolerant system is assembled by first adding correctors to an otherwise fault intolerant system, and detectors afterwards. This work was extended in several

[1]The abstraction does not distinguish between different faults. So if a process reads 2 twice, then the reading process reads them as the same fault and accepts that value as locally correct.

papers [10], [11], [12]. Although this decomposition divided the system's capabilities into detectors and correctors, it did not decompose the system itself but layers of correctors and detectors only. Furthermore, the work lacked a qualitative analysis of fault tolerance measures.

III. Decomposition and Lumping

The proposed approach contains three steps: System decomposition is discussed in Section III-A, reduction of the subsystems' DTMCs is shown in Section III-B, and recomposition of the DTMCs is presented in Section III-C. After introducing the approach informally, Section III-D provides the proofs that the proposed method provides a strongly probabilistic bisimilar model. Final remarks in Section III-E conclude this section.

Self-stabilizing systems have the benefit of unidirectional fault propagation. Given this property, decomposition gives leeway to piecewise analysis of the subsystems. During the analysis, symmetries within states can be identified. The respective states can be lumped. Thus, the size of the Markov chain of the subsystem decreases if lumping is applicable.

A. Decomposition

Two properties are important for the decomposition. For once, the system should be split into chunks that are as large as tractable. Secondly, certain processes should be in the same subsystems. Only those processes are bisimilar that have an equal effect on the system. Therefor, under unidirectional fault propagation, the decomposition should put processes into the same subsystems that have an equal (minimal) distance to the root.

Each subsystem shares at least one process with another subsystem. The shared process is referred to as *transient*. Two subsystems can share more than one transient and one transient can be shared by more than two subsystems (which are attributes derived from unidirectional fault propagation that excludes cycling faults). When a decomposition pattern has been chosen, the transformation from subsystem to its corresponding DTMC starts at the subsystem that contains the root process. The set of processes Π is split into subsets $\Pi_1, \ldots, \Pi_n$ with $\pi_1 \in \Pi_1$. We label the DTMC of subsystem Π_i with $\mathcal{D}_i$, the DTMC *excluding* any transients from lower subsystems[2] with $\mathcal{D}_{i,-}$ (the $-$ means exclusion of all lower transients), and the DTMC that models a transient π_i with $\mathcal{D}_{\pi_i}$. The successive transformation of the subsystems into DTMCs is accomplished in three recursive steps starting with Π_1.

Step 1: The initial subsystem analyzed is $\Pi_1 \subseteq \Pi$. The DTMC $\mathcal{D}_1$ of Π_1 is constructed as described in [1]. The relation between the scheduler election probabilities e between Π_1 and the rest of the system is important. Assume a system comprising a set of processes Π under serial execution semantics. As only a subset of Π is analyzed during the decomposition, chances are that either one process in Π_1 executes, or a process outside Π_1 executes. The DTMC of Π_1 itself does not take the possibility into account yet that a process outside Π_1 executes. In that case Π_1 remains in its state for one step. To take global scheduling probabilities into consideration, all transition probabilities of $\mathcal{D}_1$ are multiplied with $prob(\Pi_1)$, the chance that a process within Π_1 executes. The diagonal elements of the transition matrix $\mathcal{D}_1$ are the probabilities that the sub-system remains in its state. Subsequently, the probability $1-prob(\Pi_1)$ (the probability that a process outside Π_1 executes) is added to the diagonal elements. Thereby, the scheduler selection probabilities are taken into account and $\mathcal{D}_1$ is constructed. Subsystem Π_1 and its neighbors (the subsystems to which Π_1 propagates faults directly) share at least one mutual process. When constructing the Markov chains *below* $\mathcal{D}_1$, the neighbors of Π_1 take the intersecting process π_i as their local root. As we must regard each process only once, we exclude the intersecting process' Markov chain $\mathcal{D}_{\pi_i}$ from $\mathcal{D}_1$ and get $\mathcal{D}_{1,-}$. Afterwards, we can continue with construction of the *lower* Markov chains. The result is $\mathcal{D}_1 = \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_i}$ (analogously for multiple intersections). The $\otimes$ operator is used for the serial composition[3] of Markov chains. The algorithm for a serial composition of two DTMCs is shown in Figure 5. The example in Section IV shows its corresponding DTMCs in Figure 3.

Step 2: A subsystem $\Pi_i, i > 1$ shares one or more processes with subsystems that are closer to Π_1 than itself. The DTMCs for these subsystems Π_i are computed in a recursive manner. Further transients with underlying subsystems (i.e. subsystems that are farther away from Π_1 than Π_i itself) can be split as described above. One process can be shared by more than two subsystems, for example, if $\pi_i \in \Pi_1, \Pi_2, \Pi_3$ (Π_2 and Π_3 then both have an equal distance from Π_1 by definition). Process π_i propagates effects of faults into both subsystems Π_2 and Π_3. The transient π_i must only be regarded once. It is excluded from $\mathcal{D}_{1,-}$ and it must be further excluded from either $\mathcal{D}_3$, leaving it in $\mathcal{D}_2$, or vice versa.

Step 3: Eventually, the *lower-most* sub-systems (i.e, subsystems that do not propagate into other subsystems) are reached. They take the input from those neighboring subsystem(s) that are closer to the root process. They do not share transients with underlying subsystems, so only shared processes with subsystems that have an equal distance to Π_1 must be regarded for exclusion.

B. Reduction

A subsystem's DTMC comprises states and transitions, $\mathcal{D}_i = (\Pi_i, prob_i), prob : \Pi_i \times \Pi_i$. The conditions that qualify two states for lumping have been discussed in [13] and are formally defined in Definition 2. The reduced version of $\mathcal{D}_i$ is labeled with D'_i.

C. Recomposition

To recompose the DTMCs of the subsystems, the relevant (possibly reduced) sub-Markov chains $\mathcal{D}'_1, \ldots, \mathcal{D}'_m$ are combined using the $\otimes$ operator. The recomposed Markov chain

[2] Π_i is *lower* than Π_j if its minimal distance to $\mathcal{D}_1$ is greater.

[3] Serial composition is an interleaving-type parallel execution semantics, which operates on the Kronecker product of the components' transition matrices like true concurrency, yet excludes transitions in which more than one subsystem changes its register per time step.

is labeled $\overline{\mathcal{D}'} = \mathcal{D}'_1 \otimes \ldots \otimes \mathcal{D}'_m$. If the lumping was skipped or not applicable $\forall \mathcal{D}'_i : \mathcal{D}'_i = \mathcal{D}_i$, the result is the original Markov chain $\mathcal{D} = \overline{\mathcal{D}'}$.

D. Formal Method and Proof

We label the equivalence class of a state s under $\sim$ with $[s]_\sim$. The Markov chain $\mathcal{D}$ of a system comprises states S that are connected via transition probabilities $\mathcal{D} = (S, prob)$. We label the initial probability distribution over $\mathcal{D}$ with $prob^0(\mathcal{D})$, after k iterations with $prob^k(\mathcal{D})$, and the steady state with $prob^\infty(\mathcal{D})$.

Two states s_i and s_j belong to the same equivalence class if they either both satisfy or both dissatisfy the safety predicate $\mathcal{P}$, and have equal transition probabilities for each of their target states, as shown in Definition 2.

Definition 2

$$\begin{aligned} s_i \sim s_j :\Leftrightarrow \quad & ((s_i \models \mathcal{P} \vee s_j \models \mathcal{P}) \wedge \\ & (s_i \not\models \mathcal{P} \vee s_j \not\models \mathcal{P})) \vee \\ & \forall s \in S : prob(s_i, s) = prob(s_j, s) \end{aligned}$$

We reduce the Markov chain with $red(\mathcal{D}, \mathcal{P})$ and fit the safety predicate as shown in Definition 3.

Definition 3

$$red(\mathcal{D}, \mathcal{P}) = (\mathcal{D}', \mathcal{P}') \tag{2}$$

$$\mathcal{D}' = (S_{red}, prob_{red}) \tag{3}$$

$$S_{red} = \{[s]_\sim | s \in S\} \tag{4}$$

$$prob_{red}([s_i]_\sim, [s_j]_\sim) = \sum_{d_i \in [s_i]_\sim} prob(d_i, d_j), d_j \in [s_j]_\sim \tag{5}$$

$$[s]_\sim \models \mathcal{P}' :\Leftrightarrow \exists d \in [s]_\sim : d \models \mathcal{P} \tag{6}$$

Equation 4 describes the *state lumping* and equation 5 the *transition lumping*. The reduction by Definition 3 lumps those states that are in the same equivalence class $[s]_\sim$ (transitions respectively). The constraints that qualify states for equivalence classes are defined in Definition 2. The safety predicate $\mathcal{P}$ is defined for the state space of $\mathcal{D}$. We require an (analogously lumped) predicate $\mathcal{P}'$ that matches the reduced state space of $\mathcal{D}'$, shown in Equation 6, that describes the *predicate lumping*. In order to show that the same LWA is computed by $\mathcal{D}$ and $\mathcal{D}'$ (i.e. their strong probabilistic bisimilarity), we must first show that both have an equal steady state probability distribution.

Theorem 1

$$prob^\infty_{red}([s]_\sim) = \sum_{d \in [s]_\sim} prob^\infty(d) \tag{7}$$

We show that both the original and the reduced Markov chain have an equal steady state probability distribution according to their equivalence classes by induction.

Proof 1

Let $prob^0$ be an arbitrary initial distribution for $\mathcal{D}$ and let $prob^0_{red}([s]_\sim) = \sum_{d \in [s]_\sim} prob^0(d)$ be an initial distribution for $\mathcal{D}'$. Show that for $prob^k$ and $prob^k_{red}$, which are the probability distributions for $\mathcal{D}$ ($\mathcal{D}'$ respectively) at time point k with an initial distribution $prob^0$ ($prob^0_{red}$ respectively) the following holds:

$$\forall k : prob^k_{red}([s]_\sim) = \sum_{d \in [s]_\sim} prob^k(d) \tag{8}$$

Proof per induction over k.
Anchor: $k = 0$ holds by assumption.
Step: show that the following holds
Assumption:

$$prob^{k+1}_{red}([s]_\sim) = \sum_{[d]_\sim \in S_{red}} prob^k_{red}([d]_\sim) \cdot prob_{red}([d]_\sim, [s]_\sim) \tag{9}$$

$$= \sum_{[d]_\sim \in S_{red}} \Big(\sum_{e \in [d]_\sim} prob^k(e)\Big) \cdot \Big(\sum_{f \in [s]_\sim} prob(d, f)\Big) \tag{10}$$

$$= \sum_{[d]_\sim \in S_{red}} \sum_{e \in [d]_\sim} \sum_{f \in [s]_\sim} prob^k(e) \cdot prob(d, f) \tag{11}$$

and with $prob(e, f) = prob(d, f)$

$$= \sum_{[d]_\sim \in S_{red}} \sum_{e \in [d]_\sim} \sum_{f \in [s]_\sim} prob^k(e) \cdot prob(e, f) \tag{12}$$

$$= \sum_{e \in S} \sum_{f \in [s]_\sim} prob^k(e) \cdot prob(e, f) \tag{13}$$

$$= \sum_{f \in [s]_\sim} \sum_{e \in S} prob^k(e) \cdot prob(e, f) \tag{14}$$

$$= \sum_{d \in [s]_\sim} prob^{k+1}(d) \tag{15}$$

Thereby, $\forall k : prob^k_{red}([s]_\sim) = \sum_{d \in [s]_\sim} prob^k(d)$. □

Corollary 1

Theorem 7 and the first two conditions from Definition 2 imply that the limiting availability LWA_0 satisfies $LWA_0(\mathcal{D}, \mathcal{P}) = LWA_0(\mathcal{D}', \mathcal{P}')$. Thereby $LWA_0(\mathcal{D}, \mathcal{P}) = \sum_{s \models \mathcal{P}} prob^\infty(s)$ and consequently $LWA_0(\mathcal{D}', \mathcal{P}') = \sum_{[s]_\sim \models \mathcal{P}'} prob^\infty_{red}([s]_\sim)$.

Finally, we show that both Markov chains $\mathcal{D}$ and $\mathcal{D}'$ compute the exact same LWA.

Theorem 2

$LWA(\mathcal{D}, \mathcal{P}) \sim LWA(\mathcal{D}', \mathcal{P}') : red(\mathcal{D}, \mathcal{P}) = (\mathcal{D}', \mathcal{P}')$
with $\mathcal{D} = (S, prob), prob : S \times S \rightarrow \mathbb{R}$

Proof 2

Anchor: $LWA_0(\mathcal{D}, \mathcal{P}) = LWA_0(\mathcal{D}', \mathcal{P}')$ (cf. Corollary 1)

Step: Analogous to Proof 1, except $prob^k$ *becomes* $prob^{\infty}$.

E. Final Remarks

The method presented comprises four steps: i) system decomposition, ii) construction of the subsystems' DTMCs, iii) reduction of the DTMCs with lumping, and iv) the re-composition of the reduced DTMCs. With unidirectional fault propagation splitting is arbitrary. General recommendations (tractability and distance) were discussed in Section III-A. Issues that arise during the decomposition, like accounting for scheduling probabilities of subsystems, have been addressed. Notably, symmetries in DTMCs that arise with serial execution semantics have explicitly not been exploited. Hence, the methods presented are also capable to cope with truly concurrent execution semantics. Lumping now offers a chance to alleviate state space explosion *on the level of the subsystems*. The reduction can be applied *before* considering the whole system, i.e. piecewise, at a possibly considerably smaller amount of complexity.

IV. Example

We outline the example in Section IV-A and decompose it in Section IV-B.The results are summarized in Section IV-C.

A. Outline

Assume a distributed system comprising seven processes $\Pi = \{\pi_1, \ldots, \pi_7\}$ that are connected as shown in Figure 1. A probabilistic scheduler selects one of the processes in each computational step. All processes have an equal probability to be selected for execution. The processes execute the self-stabilizing broadcast algorithm introduced in [1]. For self-containment of the paper, the next paragraph contains a short informal description of the algorithm. The root process π_1,

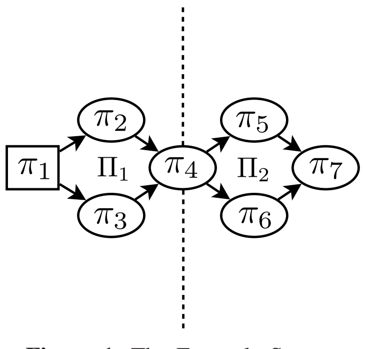

Figure 1: The Example System

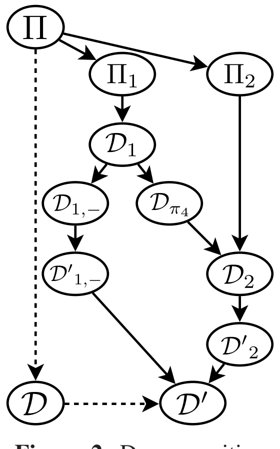

Figure 2: Decomposition

when executing a computation step, stores the value 0 in its register in absence of a fault, and a 2 if it is perturbed by a fault. Processes π_2 and π_3, when executing, copy the value of r_1 to their respective register. Process π_4 stores 0 when $(r_2 = 0 \wedge r_3 = 0) \vee (r_2 = 0 \wedge r_3 = 1) \vee (r_2 = 1 \wedge r_3 = 0)$. It stores 2 when $(r_2 = 2 \wedge r_3 = 2) \vee (r_2 = 2 \wedge r_3 = 1) \vee (r_2 = 1 \wedge r_3 = 2)$. The value 1 is stored otherwise, when both 0 and 2 are read. Process π_7 executes the same way with respect to r_5 and r_6. Processes π_5 and π_6, when executing a computation step, copy the value from r_4 to their respective register. The state space comprises states $s_i = \langle r_1, r_2, \ldots, r_7\rangle \in \{\langle 0,0,0,0,0,0,0\rangle, \ldots \langle 2,2,2,2,2,2,2\rangle\}$, which spans the state space to $2^3 \cdot 3^4 = 648$ states (processes π_1, π_2, and π_3 cannot derive 1). Transient faults perturb the executing process with a probability q. The registers of non-executing processes remain untouched by faults. Figure 2 shows the decomposition road map. The dotted arrows in Figure 2 show the common trail without decomposition ($\Pi \rightarrow \mathcal{D} \rightarrow \mathcal{D}'$). The solid arrows show the decomposition proposed, where lumping is applied piecewise, relieving the necessity to deal with the whole Markov chain. The system is split into $\Pi_1 = \{\pi_1, \ldots, \pi_4\}$ and $\Pi_2 = \{\pi_4, \ldots, \pi_7\}$. We compute $\mathcal{D}_1$ from S_1, and split it into $\mathcal{D}_{1,-}$ and $\mathcal{D}_{\pi_4}$. The state space of $\mathcal{D}_{1,-}$ with eight states is reduced to $\mathcal{D}'_{1,-}$ with six states. Then, $\mathcal{D}_{\pi_4}$ and the remaining processes form $\mathcal{D}_2$ with $3^4 = 81$ states. In $\mathcal{D}_2$ 54 states can be lumped to 27 states. The resulting reduced DTMC $\mathcal{D}'_2$ has 54 states. Re-composing $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ results in a DTMC with only 324 states.

B. Splitting and Lumping

The decomposition comprises five steps shown in Figure 2. Each following paragraph describes one step. We set the fault probability to $q = 0.05$.

Step 1: $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$*:*
First, DTMC $\mathcal{D}_1$ is computed (cf. left hand side of Figure 3 without transition probabilities). The probability that the scheduler selects a process of Π_1 is $\frac{4}{7}$. Hence, each transition in $\mathcal{D}_1$ is to be multiplied with $\frac{4}{7}$, the probability that a process within Π_1 is selected (cf. Section III-A Step 1). Then, all loop transitions (the diagonal entries of the transition matrix $\mathcal{D}_1$) gain the probability mass $\frac{3}{7}$, which is the probability that a process *outside* S_1 is selected for execution. The steady state probability distribution of $\mathcal{D}_1$ is shown in Table I.

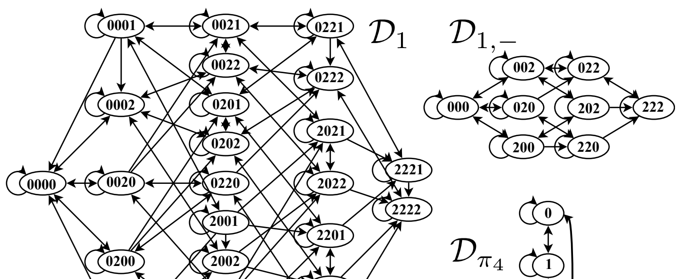

Figure 3: Markov Chain Decomposition

We employ lumping for both *splitting* Markov chains (e.g. $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$) and *pruning* of redundant information (e.g. $\mathcal{D}_{1,-} \rightarrow \mathcal{D}'_{1,-}$). For splitting we lump all states in $\mathcal{D}_1$ that have "the first three digits" in common (e.g. where $\langle r_1, r_2, r_3, r_4\rangle$

State	$\langle 0,0,0,0\rangle$	$\langle 2,0,0,0\rangle$	$\langle 0,2,0,0\rangle$	$\langle 0,0,2,0\rangle$
Probability	0.7238	0.0125	0.0208	0.0208
State	$\langle 0,0,0,2\rangle$	$\langle 0,0,0,1\rangle$	$\langle 2,2,0,0\rangle$	$\langle 2,0,2,0\rangle$
Probability	0.0469	0.0514	0.0046	0.0046
State	$\langle 2,0,0,2\rangle$	$\langle 2,0,0,1\rangle$	$\langle 0,2,2,0\rangle$	$\langle 0,2,0,2\rangle$
Probability	0.0008	0.0007	0.0022	0.0063
State	$\langle 0,2,0,1\rangle$	$\langle 0,0,2,2\rangle$	$\langle 0,0,2,1\rangle$	$\langle 2,2,2,0\rangle$
Probability	0.0308	0.0063	0.0308	0.0048
State	$\langle 2,2,0,2\rangle$	$\langle 2,2,0,1\rangle$	$\langle 2,0,2,2\rangle$	$\langle 2,0,2,1\rangle$
Probability	0.0005	0.0035	0.0005	0.0035
State	$\langle 0,2,2,2\rangle$	$\langle 0,2,2,1\rangle$	$\langle 2,2,2,2\rangle$	$\langle 2,2,2,1\rangle$
Probability	0.0077	0.0022	0.0104	0.0037

Table I: $\mathcal{D}_1$'s Steady State Distribution

= $\langle 0,0,0,0\rangle$, $\langle 0,0,0,1\rangle$, or $\langle 0,0,0,2\rangle$) and get $\mathcal{D}_{1,-}$. Afterwards, we conclude all states in $\mathcal{D}_1$ that have "the fourth digit" in common and get $\mathcal{D}_{\pi_4}$. When re-composing $\mathcal{D}_{1,-} \otimes M_{\pi_4}$, we get $\mathcal{D}_1$. The second way we exploit lumping prunes DTMC $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$. We abbreviate $[0,2]_\sim$ (the equivalence class that contains $\langle r_i, r_j\rangle = \{\langle 0,2\rangle, \langle 2,0\rangle\}$) with $\mathbb{1}$ (the doubly lined number indicates the number of faulty registers). For the analysis it is regardless which of π_2 and π_3 exactly is corrupted as they behave the same (same scheduler selection probability, same fault probability, same position in the system). The corresponding states (e.g. $\langle 0,0,2\rangle$ and $\langle 0,2,0\rangle$) have an equal role in the DTMC. The information that just one of them is corrupted suffices to compute the *LWA*. We split $\mathcal{D}_1$ into $\mathcal{D}_{1,-}$ (cf. Table II) and $\mathcal{D}_{\pi_4}$ (cf. Table III). The stationary distributions of the split DTMCs are simply the sums of the stationary distribution that the lumped states comprise. A recalculation of the stationary distribution is not necessary. We label the transition probabilities in $\mathcal{D}_{\pi_4}$ as shown in Table III to later refer to them when computing $\mathcal{D}_2$. To identify lumpable states we compare their transition probabilities to mutual target states (given they only have mutual target states). If these are equal, the tuple under consideration qualifies for lumping.

↓from/to→	$\langle 0,0,0\rangle$	$\langle 2,0,0\rangle$	$\langle 0,2,0\rangle$	$\langle 0,0,2\rangle$
$\langle 0,0,0\rangle$	0.978571	0.007143	0.007143	0.007143
$\langle 2,0,0\rangle$	0.135714	0.578571		
$\langle 0,2,0\rangle$	0.135714		0.850000	
$\langle 0,0,2\rangle$	0.135714			0.850000
$\langle 2,2,0\rangle$			0.135714	
$\langle 2,0,2\rangle$				0.135714
$\langle 0,2,2\rangle$			0.135714	0.135714
↓from/to→	$\langle 2,2,0\rangle$	$\langle 2,0,2\rangle$	$\langle 0,2,2\rangle$	$\langle 2,2,2\rangle$
$\langle 2,0,0\rangle$	0.142857	0.142857		
$\langle 0,2,0\rangle$	0.007143		0.007143	
$\langle 0,0,2\rangle$		0.007143	0.007143	
$\langle 2,2,0\rangle$	0.721429			0.142857
$\langle 2,0,2\rangle$		0.721429		0.142857
$\langle 0,2,2\rangle$			0.721429	0.007143
$\langle 2,2,2\rangle$			0.135714	0.864286

Table II: $\mathcal{D}_{1,-}$

↓from/to→	$\langle 0\rangle$	$\langle 1\rangle$	$\langle 2\rangle$
$\langle 0\rangle$	$r_4 = 0.982972$	$s_4 = 0.008687$	$t_4 = 0.008341$
$\langle 1\rangle$	$u_4 = 0.055813$	$v_4 = 0.930721$	$w_4 = 0.013466$
$\langle 2\rangle$	$x_4 = 0.081422$	$y_4 = 0.023461$	$z_4 = 0.895117$

Table III: $\mathcal{D}_{\pi_4}$

Step 2: $\mathcal{D}_{1,-} \rightarrow \mathcal{D}'_{1,-}$:
In $\mathcal{D}_{1,-}$ we spot the two states $\langle 0,0,2\rangle$ and $\langle 0,2,0\rangle$ to have an equal output. They are lumped into $\langle 0,\mathbb{1}\rangle$ (as are $\langle 2,0,2\rangle$ and $\langle 2,2,0\rangle$ to $\langle 2,\mathbb{1}\rangle$). The resulting DTMC $\mathcal{D}'_{1,-}$ is shown in Table IV.

↓from/to→	$\langle 0,0,0\rangle$	$\langle 2,0,0\rangle$	$\langle 0,\mathbb{1}\rangle$	$\langle 2,\mathbb{1}\rangle$	$\langle 0,2,2\rangle$	$\langle 2,2,2\rangle$
$\langle 0,0,0\rangle$	0.9786	0.0071	0.0143			
$\langle 2,0,0\rangle$	0.1357	0.5786		0.2857		
$\langle 0,\mathbb{1}\rangle$	0.1357		0.8500	0.0071	0.0071	
$\langle 2,\mathbb{1}\rangle$			0.1357	0.7214		0.1429
$\langle 0,2,2\rangle$			0.2714		0.7214	0.0071
$\langle 2,2,2\rangle$					0.1357	0.8643

Table IV: $\mathcal{D}'_{1,-}$

Step 3: $\mathcal{D}_{\pi_4} \otimes |\Pi_2 \setminus \{\pi_4\}| \rightarrow \mathcal{D}_2$:
With $\mathcal{D}_{\pi_4}$ at hand, we can easily compute $\mathcal{D}_2$. In Π_2 (the lower part of the system), each process stores either 0, 1 or 2. Therefore, with four processes, the state space of Π_2 comprises $3^4 = 81$ states. We exemplarily compute one transition in equation 16:

$$\langle 1,2,2,2\rangle \rightarrow \langle 2,2,2,2\rangle = \frac{1}{4} \cdot w_4 \tag{16}$$

The transition probability w_4 is given in Table III. Transitions between states where π_4 does not change its register are computed analogously to $\mathcal{D}_1$. Transitions that alter r_4, as exemplarily shown in Equation 16, are computed with the values in Table III.

Step 4: $\mathcal{D}_2 \rightarrow \mathcal{D}'_2$:
Reducing $\mathcal{D}_2 \rightarrow \mathcal{D}'_2$ offers 27 state pairs to be lumped. The sets can informally be described as pairs of states, where r_4 and r_7 store an equal value, while r_5 and r_6 store unequal values, and each state's r_5 is equal to the mutual other state's r_6. For instance, state $\langle 0,0,2,2\rangle$ and $\langle 0,2,0,2\rangle$ can be lumped to $\langle 0,\mathbb{1},2\rangle$. For the computation of the *LWA* we need not know which of the processes π_5 and π_6 actually is corrupted. Knowing that *one* of them is defective suffices to compute the *LWA*. The following pattern defines the sets of lumpable states formally:

A state of $\mathcal{D}_2$ is of the form $c_2 = \langle r_4, r_5, r_6, r_7\rangle$. States

- $\langle x,0,1,y\rangle$ and $\langle x,1,0,y\rangle$ form $\langle x,\mathbb{1},y\rangle$,
- $\langle x,0,2,y\rangle$ and $\langle x,2,0,y\rangle$ form to $\langle x,\mathbb{2},y\rangle$, and
- $\langle x,1,2,y\rangle$ and $\langle x,2,1,y\rangle$ form to $\langle x,\mathbb{3},y\rangle$.

State pairs s_i and s_j (with r_x^i of s_i, and r_x^j of s_j) are lumpable where

1) $r_4^i = r_4^j$ (or $r_7^i = r_7^j$ respectively), and
2) $r_5^i \neq r_6^i$, and
3) $r_5^i = r_6^j$ and $r_6^i = r_5^j$.

The first condition demands that the registers r_4 within both states must be equal (respectively r_7). The second condition demands, that the registers r_5 and r_6 must not be equal within each state. The third condition demands, that r_5 in both states is equal to r_6 of the other state.

After lumping states, the lumping of transitions is presented. To reduce a set of states, all their incoming and outgoing transitions must be aggregated. We exemplarily show the lumping of a transition set that aggregates transitions from one lump of states into another lump of states:

$\overrightarrow{\langle 1,\mathbb{1},0\rangle,\langle 0,\mathbb{1},0\rangle}$. The $\oplus$ operator is used to describe the *reduction* of states (similar to the $\otimes$ operator we used earlier). The first lump comprises the states $\langle 1,\mathbb{1},0\rangle = \langle 1,1,0,0\rangle \oplus \langle 1,0,1,0\rangle$. The second lump comprises the states $\langle 0,\mathbb{1},0\rangle = \langle 0,1,0,0\rangle \oplus \langle 0,0,1,0\rangle$. While some transition probabilities are zero $prob(\overrightarrow{\langle 1,1,0,0\rangle,\langle 0,0,1,0\rangle}) = prob(\overrightarrow{\langle 1,0,1,0\rangle,\langle 0,1,0,0\rangle}) = 0$, others form the aggregated transition probability $prob(\overrightarrow{\langle 1,1,0,0\rangle,\langle 0,1,0,0\rangle}) = u_4 \cdot prob(\pi_4)\cdot \frac{4}{7}$, and $prob(\overrightarrow{\langle 1,0,1,0\rangle,\langle 0,0,1,0\rangle}) = u_4 \cdot prob(\pi_4)\cdot \frac{4}{7}$. The variable u_4 is the transition probability that r_4 changes its value from 1 to 0 as shown in Table III, and $prob(\pi_4)$ is the execution probability of π_4 within the sub-system, i.e. here $\frac{1}{4}$. As discussed in $Step1 : \mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$, the distinction between the possibilities that either a process *in* the DTMC (here $\mathcal{D}_2$) is selected for execution, or a process *outside* is selected (which is $\mathcal{D}_{1,-}$ in this case), is important. We regard this distinction *before* lumping. Hence, the transition probabilities are multiplied with $\frac{4}{7}$.

↓from/to→	$\langle 0,1,0,0\rangle$	$\langle 0,0,1,0\rangle$
$\langle 1,1,0,0\rangle$	$u_4 \cdot \pi_4 \cdot \frac{4}{7}$	0
$\langle 1,0,1,0\rangle$	0	$u_4 \cdot \pi_4 \cdot \frac{4}{7}$

Table V: Example Transition Lumping: Equations

With steady state probabilities

- $prob^{\infty}(\langle 1,1,0,0\rangle) = 0.0041598$ and
- $prob^{\infty}(\langle 1,0,1,0\rangle) = 0.0025722$

and the above transition probabilities, Equation (1) from [1, p.27] computes the lumped transition probability shown in Equation 17 (cf. Figure 4):

$$\overrightarrow{\langle 1,\mathbb{1},0\rangle,\langle 0,\mathbb{1},0\rangle} = \frac{(\overrightarrow{\langle 1,1,0,0\rangle,\langle 0,1,0,0\rangle} \cdot prob^{\infty}(\langle 1,1,0,0\rangle))}{prob^{\infty}(\langle 1,1,0,0\rangle + prob^{\infty}(\langle 1,0,1,0\rangle)} + \frac{(\overrightarrow{\langle 1,0,1,0\rangle,\langle 0,0,1,0\rangle} \cdot prob^{\infty}(\langle 1,0,1,0\rangle))}{prob^{\infty}(\langle 1,1,0,0\rangle + prob^{\infty}(\langle 1,0,1,0\rangle)} \tag{17}$$

All equivalence classes are lumped this way. The resulting

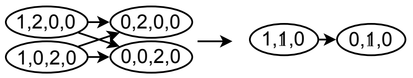

Figure 4: Reduction Example

DTMC is $\mathcal{D}'_2$.

Step 5: Re-Composition:
With $\mathcal{D}'_{1,-}$ and $\mathcal{D}'_2$ at hand, $\overline{\mathcal{D}'}$ is composed which has the perk to be strongly probabilistic bisimilar to $\mathcal{D}$ and therefor can compute the LWA of the original system. Notably, both DTMCs $\mathcal{D}'_{1,-}$ and $\mathcal{D}'_2$ execute computation steps parallel as their probabilities have been weighted. For re-composition, each entry of $\mathcal{D}'_{1,-}$ is multiplied with each entry of $\mathcal{D}'_2$. The coordinates are labeled row i and column j in $\mathcal{D}'_{1,-}$, and k and l in $\mathcal{D}'_2$ respectively. The algorithm in Figure 5 computes the re-composition: The final step to let $\overline{\mathcal{D}'}$ compute the LWA (the stationary distribution is known), is to set the transition probability $\overline{\mathcal{D}'}(1,1) := 1$, and $\forall m, 1 < m \leq 324 : \mathcal{D}'(1,m) := 0$ as discussed in [1, ch.4] (the state aggregate that satisfies the (lumped) safety predicate becomes an absorbing state).

```
$\mathcal{D}' = zeros(324)$;
for $j = 1 : 6$ do
    for $l = 1 : 54$ do
        for $i = 1 : 6$ do
            for $k = 1 : 54$ do
                if $i \neq \wedge l \neq k$ then
                    $\mathcal{D}'((j-1)\cdot 54 + l, (i-1)\cdot 54 + l) =$
                    $\mathcal{D}'_{1,-}((j-1)\cdot 54 + l, (i-1)\cdot 54 + l) +$
                    $\mathcal{D}'_{1,-}(j,i)\cdot \mathcal{D}'_2(l,k)\cdot \frac{3}{7}$;
                    $\mathcal{D}'((j-1)\cdot 54 + l, (j-1)\cdot 54 + k) =$
                    $\mathcal{D}'_{1,-}((j-1)\cdot 54 + l, (j-1)\cdot 54 + k) +$
                    $\mathcal{D}'_{1,-}(j,i)\cdot \mathcal{D}'_2(l,k)\cdot \frac{4}{7}$;
                else
                    $\mathcal{D}'((j-1)\cdot 54 + l, (i-1)\cdot 54 + k) =$
                    $\mathcal{D}'_{1,-}((j-1)\cdot 54 + l, (i-1)\cdot 54 + k) +$
                    $\mathcal{D}'_{1,-}(j,i)\cdot \mathcal{D}'_2(l,k)$;
```

Figure 5: Recomposition of $\mathcal{D}'_{1,-}$ and $\mathcal{D}'_2$ to $\overline{\mathcal{D}'}$

C. Conclusion

The DTMC $\overline{\mathcal{D}'}$ computes the LWA. Figure 6 shows the strictly monotonous increase of probability mass over the first 1000 computation steps. If the system is supposed to obtain a certain amount of availability within a distinct number of computation steps, we can now refer to Figure 6 that tells the associated amount of time that would be required to meet the demand. Furthermore, we can investigate the *probability mass drain* throughout the states over time. For reference purposes we order the lumped states $\langle 0,0,0,0,0,0,0\rangle, \ldots \langle 2,2,2,2,2,2,2\rangle$. Comparing the degree of probability mass drain within each state, as shown in Figure 7, exposes two prominent states for discussion. These are namely the 43^{rd} ($\langle 0,0,0,1,1,1,1\rangle$) and the 109^{th} ($\langle 0,\mathbb{1},0,0,0,0\rangle$) state. Although initially equipped with a similar amount of probability mass, the 109^{th} state looses its probability mass rapidly, while the 43^{rd} state is drained at a slower pace as shown in Figures 8 and 9. The initial

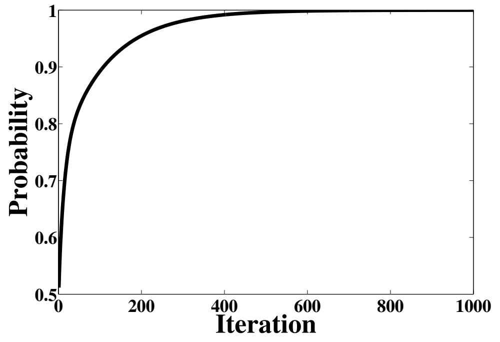

Figure 6: LWA of the Example System

motivation of computing the LWA was to find the required amount of time to achieve a certain probability with which a non-masking fault tolerant system can mask faults under fault detection. Knowing about the probability mass drain of *all* states allows yet for much more. Some systems offer the possibility for certain states to be either prevented or instantly repaired (cf. *snap stabilization*). When looking for state candidates for which counter measures should be applied,

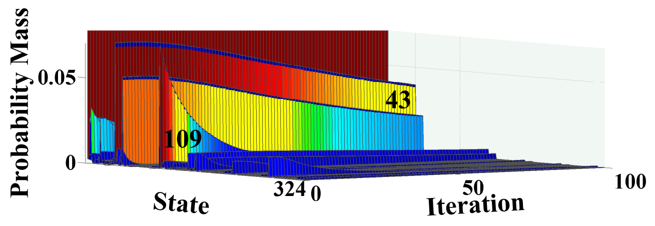

Figure 7: Probability Mass Drain

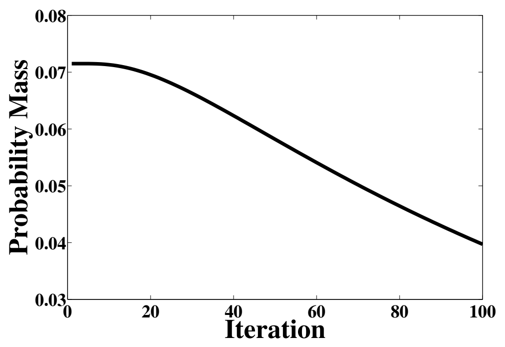

Figure 8: 43^{rd} State

the 43^{rd} state is obviously a more suitable point of attack than the 109^{th} state. Either preventing the 43^{rd} state to collect probability mass in the first place, or employing measures that help drain this state at a faster pace both seem desirable targets. Further analyses, especially when applying upper temporal boundaries, let all states be sorted in this manner. For instance, a constraint might be that the system must be stable after at most 30 computation steps with a maximal probability. The designer may have the possibility to *prevent* certain states at an extra cost. Which states should be prevented? Having resources for a distinct number n of states to tackle, as well as a the list of all states sorted by the probability mass they withhold at time point 30, the top n states (considering that lumped states comprise two states in our case) are in the solution set.

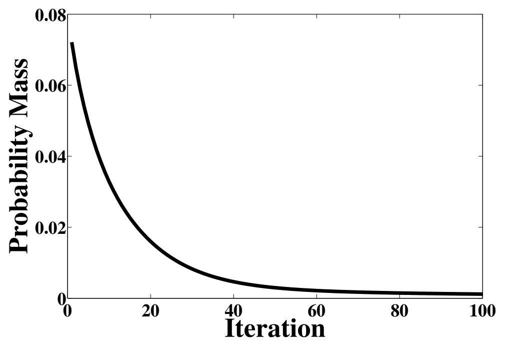

Figure 9: 109^{th} State

V. Conclusion

The likelihood of a fault-tolerant distributed system, be it masking or non-masking, to provide service can be computed with the help of Markov models. A common limiting factor is the complexity of a system's corresponding Markov model growing exponentially with the system's size. Hence, Markov model analysis soon becomes intractable. Lumping is a popular means of reducing a Markov model to a possibly considerably smaller size via either weak or strong probabilistic bisimulation. To evade the construction of the whole Markov chain in the first place, decomposition allows for piecewise application of lumping on the sub-Markov chains (i.e. the Markov chains of the decomposed subsystems). Combining decomposition with lumping allows for construction of strongly probabilistic bisimilar sub-Markov chains. The suggested combination of lumping and decomposition supports the quantitative analysis of fault-tolerance mechanisms, as has been demonstrated on a self-stabilizing system. Future work will investigate the classes of semi-hierarchical and heterarchical fault propagation on the one hand, and the cooperative effect of spatial and temporal redundancy on the other hand.

Acknowledgment

This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center Automatic Verification and Analysis of Complex Systems (SFB/TR 14 AVACS), and by the European Commission under the MoVeS project, FP7-ICT-2009-257005.

References

[1] Nils Müllner and Oliver Theel. The Degree of Masking Fault Tolerance vs. Temporal Redundancy. *Proc. of WAINA'11*, pages 21–28, 2011.

[2] John George Kemeny and James Laurie Snell. *Finite Markov chains*. University series in undergraduate mathematics. VanNostrand, New York, repr edition, 1969.

[3] Vassilios Mertsiotakis. *Approximate Analysis Methods for Stochastic Process Algebras*. PhD thesis, Universität Erlangen-Nürnberg, 1998.

[4] L. Capra, C. Dutheillet, G. Franceschinis, and J. M. Ilié. On the Use of Partial Symmetries for Lumping Markov Chains. *SIGMETRICS Perform. Eval. Rev.*, 28:33–35, March 2001.

[5] Salem Derisavi. Signature-based Symbolic Algorithm for Optimal Markov Chain Lumping. In *QEST'07*, pages 141–150, 2007.

[6] Salem Derisavi. A Symbolic Algorithm for Optimal Markov Chain Lumping. In *TACAS'07*, pages 139–154, 2007.

[7] Salem Derisavi, Peter Kemper, and William H. Sanders. Lumping Matrix Diagram Representations of Markov Models. In *DSN'05*, 2005.

[8] Salem Derisavi, Holger Hermanns, and William H. Sanders. Optimal State-space Lumping in Markov Chains. *Inf. Process. Lett.*, 87:309–315, September 2003.

[9] Sandeep Kulkarni. *Component Based Design of Fault-Tolerance*. PhD thesis, The Ohio State University, Columbus Ohio, USA, 1999.

[10] Sandeep Kulkarni and Anish Arora. Compositional Design of Multitolerant Repetitive Byzantine Agreement. *Lecture Notes in Computer Science*, 1346:169–181, 1997.

[11] Anish Arora and Sandeep Kulkarni. Designing Masking Fault-Tolerance via Nonmasking Fault-Tolerance. *IEEE Transactions on Software Engineering*, 24(6):435–450, 1998.

[12] Sandeep Kulkarni and Anish Arora. Automating the Addition of Fault-Tolerance. In *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRTFTS'2000)*, 2000.

[13] Nils Müllner and Oliver Theel. The Degree of Masking Fault Tolerance vs. Temporal Redundancy -Erratum. In *http://www.informatik.uni-oldenburg.de/~phoenix/docs/erratum.pdf* [1], pages 21–28.