Combining Decomposition and Reduction for State Space Analysis of a Self-Stabilizing System[1]

Nils Müllner, Oliver Theel, Martin Fränzle

Carl von Ossietzky Universität Oldenburg
D-26111 Oldenburg, Germany
Email: nils.muellner | theel | martin.fraenzle@informatik.uni-oldenburg.de

Abstract

Fault tolerance measures of distributed systems can be calculated precisely by state space analysis of the Markov chain corresponding to the product of the system components. The power of such an approach is inherently confined by the state space explosion, i.e. the exponential growth of the Markov chain in the size of the underlying system. We propose a decompositional method to alleviate this limitation. Lumping is a well-known reduction technique facilitating computation of the relevant measures on the quotient of the Markov chain under lumping equivalence. In order to avoid computation of lumping equivalences on intractably large Markov chains, we propose a system decomposition supporting local lumping on the considerably smaller Markov chains of the subsystems. Recomposing the lumped Markov chains of the subsystems constructs a lumped transition model of the whole system, thus avoiding the construction of the full product chain. An example demonstrates how the limiting window availability (i.e. a particular fault tolerance measure) can be computed for a self-stabilizing system exploiting lumping and decomposition.

[1]This is an expanded version of a paper with the same title which was presented at the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA2012), Fukuoka-shi, Japan, March 26-29, 2012.

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 as part of the project on Modeling, Verification and Control of Complex Systems (FP7-ICT-2009-257005 MoVeS).

Keywords: fault tolerance, self stabilization, probabilistic model checking, probabilistic bisimilarity, Markov chains, limiting window availability

1. Introduction

Fault tolerance is a desirable feature of distributed systems. System engineers can access a repertoire of *detectors* and *correctors* to make a distributed system fault-tolerant. While detectors reveal faults, correctors let the system recover.

We follow the classification by Avižienis et al. [1] and assume that faults are undetected corruptions that become errors upon detection. When errors are corrected *in time*, the system recovers. Otherwise, they become failures. Detectors and correctors both commonly employ redundancy, either spatial (e.g. error detecting or correcting codes) or temporal (e.g. detection via sequential dual modular redundancy or correction via self-stabilization) or a combination thereof (e.g. retry after detection). To ensure liveness, the system must not recover indefinitely. We employ a fault model of transient faults that strike the system probabilistically. The relation between time for recovery and the probability of successful recovery is defined by the measure of *limiting window availability* (*LWA*) as introduced in our previous work [2]. More time for recovery increases the probability of successful eventual recovery. The probabilistic analysis is precise in the sense that it provides the exact probability distribution for a successful recovery over time. We focus on *correction* via *time* to measure the recovery rate using *LWA*. To comply, external effects that distort the measurement (i.e. detection) are to be excluded. Correction without detection means *ignorant* correction, that *washes* the effects of faults out of the system over time intrinsically. A concept to cope with faults without the explicit necessity of detectors is self-stabilization. The dominant resource required by self-stabilization is time. Yet, detection is required to determine whether the system has recovered successfully or not. It must fulfill two requirements: It must be perfect and it must not interact with the recovery process itself. The latter requirement has been addressed. The first requirement excludes false positives and false negatives that naturally arise with unreliable detectors. A *perfect* fault detector is mounted between the system and its user to fulfill both requirements as shown in Figure 1. It passes inquiries from the user to the system. If the system response is correct, then it is passed on to the user. Otherwise, the fault detector blocks the response and retries the inquiry.

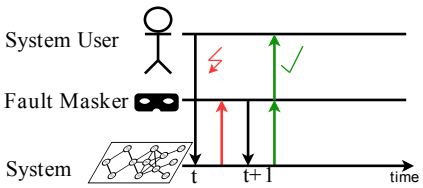

Figure 1: The Fault Masker [3]

Markov chains, Lumping, and Decomposition.
The *LWA* (cf. [2]) 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 possibly be in as well as the transitions between them. The probabilities with which processes execute a computation step or a fault step define the transition probabilities between the states. The result is a discrete-time Markov chain (DTMC) $\mathcal{D}$. The size of $\mathcal{D}$ is exponential to the size of the system. A reduction is therefore highly desired. *Lumping* is a popular method to coalesce states and thus reduce the size of a DTMC. The challenge in lumping is to prune only irrelevant information and to conserve necessary properties, which is in our case the ability to compute the *LWA*. Lumping in the light of fault tolerance analysis was presented in [2].

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. Lumping states accordingly allows to reduce the state space. Fortunately, fault-tolerant systems often rely on parallel, uniform structures and multiply instantiated components which makes them a benign target. These uniform components likely offer great potential for lumping. As lumping results in a *strongly probabilistic bisimilar* DTMC, the reduced DTMC does still compute the precise *LWA*. In this paper, we extend our earlier work by system decomposition to allow for *local* lumping on the subsystems. Thereby, the necessity to construct the whole DTMC at once in the first place becomes obsolete. *Local* lumping allows to analyze systems that would be too large and intractable to tackle with traditional methods.

Classification of Fault Propagation.
Fault propagation plays an important role in system decomposition. We distinguish three classes of fault propagation: i) hierarchical, ii) semi-hierarchical

and iii) heterarchical.

i) 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*.

ii) *Semi-hierarchical* systems follow the same principle, but the role of the designated leader might switch over time. Thereby, each case of a possible leader is to be considered distinctively. Furthermore, faults can propagate in *any* direction during leader change.

iii) In *heterarchical* systems, all processes are 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 decomposition and local lumping. Hierarchical fault propagation transforms the system topology 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. Notably, successive composition of the transition models of mutually independent subsystems (i.e. subsystems that do not propagate the effects of faults like e.g. energy grids) is also tractable.

Contribution and Structure.
The main contributions of this work are the decomposition that allows for local lumping, and the proof (Proof 3.2 in Section 3.4), that lumping provides for a strong probabilistic bisimulation and thus is neutral on computing the LWA. After a suitable system model is introduced and related work is discussed in Section 2, Section 3 shows how lumping and decomposition can be combined. Section 4 demonstrates the proposed method when computing the *LWA* for a self-stabilizing system. Section 5 concludes the proposed method and discusses possible extensions in the domain of power grid analyses.

2. Preliminaries and Related Work

Section 2.1 introduces the system model together with the properties we are interested in. Section 2.2 discusses related work.

2.1. Preliminaries

A system Π is a set of n processes $\{\pi_1, \ldots, \pi_n\}$ that are connected via bidirectional communication channels. Two processes that are connected are called neighbors. Each process has a segment R_i of memory — in the remainder called its register — that is visible to its neighbors. Consequently, π_i can read its neighbors' registers R_j and write to its own register R_i. The system state s_t at time t is the snapshot of the assigned values over all registers $s_t = \langle R_1, \ldots, R_n \rangle$. The communication channels only define mutual register access and are not relevant for the system state.

As a canonical simplification, the visible state space of each process in the model is abstracted to a three-valued domain, as proposed by Katoen et al. [4]. In this abstraction, a process π_i stores just one of three possible values (0, 1, 2) in its single (abstract) register R_i. If $R_i = 0$ applies, then process π_i resides in a correct state in the sense that its concrete state R_i currently satisfies its *local* safety predicate $\mathcal{P}_i$, denoted by $R_i \models \mathcal{P}_i$. When $R_i \not\models \mathcal{P}_i$, then π_i either *knowingly* cannot determine the correct value its register should hold (e.g. due to dependencies on unavailable data from other processes) or it is *unknowingly* perturbed by a fault, either directly or via fault propagation. In the abstraction, R_i takes the value 1 (i.e. $R_i = 1$) when π_i *knowingly* cannot compute a correct value for its register and $R_i = 2$ when π_i has *unknowingly* stored[2] an incorrect value. The system state s_t satisfies the global safety predicate, denoted by $s_t \models \mathcal{P}$, iff $\forall \pi_i \in \Pi : R_i = 0$.

Strict Stabilization in a Probabilistic Environment.
Despite the correct value 0 and the (undetected) incorrect value 2, a third value 1 is required to prove the property of self-stabilization for an algorithm (e.g. [2, Proof 1]). Assume that each non-root process relies on information by all neighbors that have an equal or shorter minimal distance to the root[3] than itself. When the executing process is provided with contradicting information (i.e. it reads both 0 and 2 values), it requires the possibility to reach a state from which it cannot propagate the faulty information. Thereby, it cannot perturb its neighbors, and the 0 values — provided by the root process —

[2] The abstraction does not distinguish between faults. When a process in the abstraction reads 2 twice, the abstraction pessimistically assumes that it might also read consistent values (e.g. originating from the same fault) and hence accepts that value as locally correct.

[3] Non-root processes in the example in Section 4 even consider only information from processes that are strictly closer to the root.

propagates eventually (during the absence of faults). By excluding mutual fault propagation, cyclic fault propagation is also excluded. Faults propagate strictly unidirectional in the direction from the root process towards the leaf processes (i.e. the processes farthest away from the root process).

We consider status 1 explicitly *not* as fault detection as it does not trigger specific countermeasures. Furthermore, the 1 status is also considered as a fault status. Although a *probabilistically* self-stabilizing variant (in which the executing process probabilistically chooses one of the provided values) would suffice, we consider the deterministic variant for the sake of consistency with our previous approach [2] and to enforce unidirectional fault propagation.

Liveness.
Liveness means that "the good thing" happens eventually [5]. Here, "the good thing" is convergence, which includes that recovery succeeds within finite time. There are execution traces that continuously visit illegal states. As the length of the execution traces (the set of which unfolds like a tree over time) reaches infinity (i.e. the limit), the *probability* for these traces approaches zero. In the limit, these traces are *improbable* but *possible*. Assume for example, that all processes are in a legal state except one process that does not inflict faults on other processes due to the hierarchical structure. To succeed with stabilization, it is only required that this one process executes and is not perturbed by a fault. Yet, the *probabilistic* scheduler might continuously ignore this process, or whenever the process executes, it is perturbed by a fault. The *probability* that the process does not recover eventually in an infinite trace is zero, but there is at least one execution trace that allows for it. Hence, the system under consideration is exposed to probabilistic influence (scheduler, faults and possibly even algorithm) and therefore cannot satisfy *strict* liveness (i.e. on *all* execution traces). Here, *probabilistic* liveness suffices that allows for *possible* continuous violations of safety as long as these violations are *improbable* (i.e. their aggregated probability is zero).

Execution Semantics.
In our example in Section 4.1, the system executes the self-stabilizing broadcast algorithm (BASS) (cf. [2, p.24]) that is discussed in Section 4.1. Process π_1 is selected as 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 computation steps. This means that at each time step, exactly one process π_i

of n processes is randomly selected to take a computation step (i.e. execute an atomic guarded command), with selection probability $\mathfrak{s}_i = \frac{1}{n}$ for process π_i. The guarded commands are deterministic. When selected to execute a computation step, π_i executes its desired action with probability p, and with probability $q = 1 - p$ it instead takes a fault step. In the latter case, the executing process' register is corrupted by a fault, which is reflected by writing a 2 to its register R_i in the abstraction. Note that these component models together with the scheduler provide a discrete-time Markov chain over state space $S \leq \{0, 1, 2\}^n$.

By constructing this abstract model $\mathcal{D}$ (or its lumping $\mathcal{D}'$, as proposed later on) we can easily verify that under a probabilistic scheduler and fault model, the system's DTMC is ergodic. Hence, regardless of its initial state, $\mathcal{D}$ reaches the same *stationary distribution* (for which we require probabilistic liveness). This stationary distribution serves as initial probability distribution for computing the LWA_w, which is the probability that the system satisfies the safety predicate $\mathcal{P}$ in the current state or within at most w time steps afterwards for at least one computation step. LWA_w can be computed iteratively as follows. LWA_0 is the aggregated probability mass of the legal (i.e. satisfying) states in the stationary distribution. LWA_1 is the same probability mass plus the probability mass that drains from the illegal states into the legal states within one computation step. The sequence LWA^n contains the first $n - 1$ iterations.

Definition 2.1 (Limiting Window Availability)**.**

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

$$LWA^n = \langle LWA_0, LWA_1, \ldots, LWA_{n-1} \rangle \tag{2}$$

2.2. Related Work

This article extends our previous work [6] by enhancing the level of detail and discussing that not only lumping, but also that the decomposition preserves a strong probabilistic bisimilarity. The proposed decomposition provides a more efficient approach to lumping in the light of fault tolerance analysis [2]. Finite Markov chains and lumping were introduced by Kemeny and Snell [7] in a general form, however, without exploiting structural properties of fault-tolerant systems for decompositional reasoning. Hillston develops a stochastic process algebra in her PhD thesis [8] to evaluate the performance of a system. She employs strong bisimilar lumping to reduce continuous-time

Markov chains. Yet, evaluating fault tolerance measures is not in her focus. Approximate probabilistic bisimulations on various models like Markov chains and Petri nets are discussed in Mertsiotakis' PhD thesis [9]. It reveals, how lumping can be exploited until a model becomes tractable for analysis, yet again without focus on the structure of fault-tolerant systems. Focusing on performance analysis, Capra et al. [10] apply lumping to reduce Markov chains without considering decomposition. Derisavi contributed on the optimality of lumping [11, 12, 13, 14] by investigating coarsest lumping quotients, which is orthogonal to our approach. It addresses efficient (and thus in general non-optimal w.r.t. size of the quotient) computation of lumping quotients by decomposition. In a qualitative rather than probabilistic setting, Kulkarni [15] 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 papers [16, 17, 18], yet not lifted to a probabilistic setting.

3. Decomposition and Lumping

The combination of decomposition and lumping comprises three parts. The first part is system decomposition, which is discussed in Section 3.1. The state space reduction of the subsystems' DTMCs is shown in Section 3.2. Section 3.3 explains the recomposition of the locally lumped DTMCs. Section 3.4 proves that the reduction is an isomorphic abstraction which is strongly probabilistic bisimilar with regards to the *LWA*. The preservation of strong probabilistic bisimilarity during the decomposition is discussed in Section 3.5.

3.1. Decomposition

Two properties are important for the decomposition: The system should be split into "chunks" that are as large as tractable, and bisimilar processes should be in the same subsystems. Processes are bisimilar when they have an *equal effect* on the system. Under unidirectional fault propagation, processes must have an equal distance to the root process to be bisimilar. The first rule of decomposition is to split the system *layerwise* such that processes which have an equal distance to the root are in the same subsystem (if possible).

In our decomposition, each subsystem shares at least one process with another subsystem. The shared process is referred to as *transient*. Two sub-

systems can share more than one transient and one transient can be shared by more than two subsystems (since unidirectional fault propagation prevents cyclic fault propagation). When a decomposition pattern has been selected, the transformation for each subsystem to its corresponding DTMC starts with the subsystem that contains the root process and continues from the root towards the leaves in the direction in which faults can propagate. The set of processes Π is split into subsets $\Pi_1, \ldots, \Pi_n$ with the root process $\pi_1 \in \Pi_1$. We label the DTMC of subsystem Π_i with $\mathcal{D}_i$, and the DTMC of Π_i with all *lower* transients excluded[4] with $\mathcal{D}_{i,-}$ (the "$-$" represents exclusion of all relevant 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 recursively in three steps starting with Π_1.

Step 1 - The Root Subsystem.
The initial subsystem analyzed is $\Pi_1 \subseteq \Pi$. The DTMC $\mathcal{D}_1$ of Π_1 is constructed as described in [2]. To construct $\mathcal{D}_1$, the transition probabilities between each two states are computed taking the scheduling and fault probabilities and the algorithm into account. The example in the following section shows, how the DTMC of a (sub-)system is constructed. The relation between the scheduler selection probabilities $\mathfrak{s}$ of Π_1 and the rest of the system is important. Assume a system comprising processes Π under serial execution semantics. As the local analysis only targets subsets of Π one by one, chances are that either a process in the subset currently under investigation (here Π_1) executes, or a process *outside* executes. When $\mathcal{D}_1$ is constructed, that *relative* scheduling probability is not regarded at first. In case a process *outside* Π_1 executes, $\mathcal{D}_1$ remains in its state for one step. To take global scheduling probabilities into consideration, all transition probabilities in $\mathcal{D}_1$ are multiplied with $\mathfrak{s}_{\Pi_1}$, the chance that any process within Π_1 executes. Thereby, the line sum in Π_1's transition probability matrix becomes $\mathfrak{s}_{\Pi_1}$. The diagonal elements of the transition matrix $\mathcal{D}_1$ are the probabilities that the subsystem remains in its state. Subsequently, the probability $1 - \mathfrak{s}_{\Pi_1}$ (the probability that a process outside Π_1 executes) is added to the diagonal elements. Thereby, the line sums in $\mathcal{D}_1$ become one, global scheduling has been accounted for, and the construction of $\mathcal{D}_1$ is complete. Subsystem Π_1 and its (lower) neighbors (the subsystems to which Π_1 propagates faults directly)

[4] Π_j is *lower* than Π_i if its minimal distance to $\mathcal{D}_1$ is greater. A transient π_i between upper Π_i and lower Π_j is a *lower* transient to Π_i and an *upper* transient to Π_j.

share at least one mutual transient 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' (e.g. π_i) share in $\mathcal{D}_1$ which is $\mathcal{D}_{\pi_i}$ and get $\mathcal{D}_{1,-}$. Afterwards, we can continue with the construction of the *lower* Markov chains. The splitting result is $\mathcal{D}_1 = \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_i}$ (analogously for multiple intersections). The $\otimes$-operator represents the serial composition[5] of DTMCs. An algorithm computing $\otimes$ (i.e. *serially* composing two DTMCs) is shown in Algorithm 1. The example in Section 4 shows the splitting (i.e. the exclusion of a transient process) in Figure 4.

Step 2 - The Intermediate Subsystems.
A subsystem Π_i 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 even lower 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 $\pi_i \in \Pi_j, j = 1, 2, 3$ (Π_2 and Π_3 are neighbors and both have an equal distance from Π_1 by definition). Subsystem Π_1 propagates effects of faults through process π_i into both subsystems Π_2 and Π_3. The transient π_i must only be regarded once. It is excluded from $\mathcal{D}_1$ (that becomes $\mathcal{D}_{1,-}$) and it must be joined exclusively to either $\mathcal{D}_3$, excluding it from $\mathcal{D}_2$, or vice versa.

Step 3 - The Leaf Subsystems.
Eventually, the *lowest* subsystems (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.

3.2. Reduction

A subsystem's DTMC comprises states and transition probabilities:

$$\mathcal{D}_i = (\Pi_i, prob(S_i \times S_i)) \tag{3}$$

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

with S_i being the state space of sub-Markov chain $\mathcal{D}_i$. The reduced (lumped) version of $\mathcal{D}_i$ obtained by coalescing states satisfying these conditions is denoted $\mathcal{D}'_i$ in the remainder.

3.3. Recomposition

To recompose the (now through lumping reduced) DTMCs of the subsystems, the relevant sub-Markov chains $\mathcal{D}'_1, \ldots, \mathcal{D}'_m$ are combined using the $\otimes$-operator (cf. Algorithm 1 in Section 4.2). The recomposed Markov chain is labeled $\overline{\mathcal{D}'} = \mathcal{D}'_1 \otimes \ldots \otimes \mathcal{D}'_m$. In case $\overline{\mathcal{D}'}$ contains no further potential for lumping, we write $\overline{\mathcal{D}'} = \mathcal{D}'$.

3.4. Formal Method and Proof

This section provides proof that lumping of strong probabilistic bisimilar states maintains the DTMC's ability to *precisely* compute the *LWA*. 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, which are $\mathcal{D} = (S, prob(S \times S))$. We label the initial probability distribution over $\mathcal{D}$ with $prob^0(S)$, after k iterations with $prob^k(S)$, and the steady state with $prob^\infty(S)$. 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 3.1.

Definition 3.1 (Conditions)**.**

$$s_i \sim s_j :\Leftrightarrow ((s_i \models \mathcal{P} \wedge s_j \models \mathcal{P}) \vee \tag{4}$$

$$(s_i \not\models \mathcal{P} \wedge s_j \not\models \mathcal{P})) \wedge \tag{5}$$

$$\forall s \in S : prob(s_i, s) = prob(s_j, s) \tag{6}$$

We reduce the Markov chain with $red(\mathcal{D}, \mathcal{P})$ and fit the safety predicate as shown in Definition 3.2.

Definition 3.2 (Reduction)**.**

$$red(\mathcal{D}, \mathcal{P}) = (\mathcal{D}', \mathcal{P}') \tag{7}$$

$$\mathcal{D}' = (S', prob(S' \times S')) \tag{8}$$

$$S' = \{[s]_\sim | s \in S\} \tag{9}$$

$$prob([s_o]_\sim, [s_t]_\sim) = \frac{\sum\limits_{d_i \in [s_o]_\sim} \sum\limits_{d_j \in [s_t]_\sim} prob(d_i, d_j) \cdot prob^\infty(d_i)}{\sum\limits_{d_i \in [s_o]_\sim} prob^\infty(d_i)} \tag{10}$$

$$[s]_\sim \models \mathcal{P}' :\Leftrightarrow \exists d \in [s]_\sim : d \models \mathcal{P} \tag{11}$$

The reduction $red(\mathcal{D}, \mathcal{P})$ reduces the DTMC $\mathcal{D}$ and adapts the predicate $\mathcal{P}$ accordingly as shown in Equation 7. The reduced DTMC $\mathcal{D}'$ consists of a reduced state space S' and changes the transitions accordingly as shown in Equation 8. Equation 9 describes the *state lumping* and Equation 10 the *transition lumping*. States that belong to the same equivalence class $[s]_\sim$ are aggregated and their transitions are computed respectively as shown in Equation 10. The conditions that qualify states for equivalence classes are defined in Definition 3.1. The safety predicate $\mathcal{P}$ is defined for the state space of $\mathcal{D}$. When $\mathcal{D}$ is reduced, we require an (analogously lumped) predicate $\mathcal{P}'$ that matches the reduced state space of $\mathcal{D}'$, as shown in Equation 11. In order to show that the same *LWA* is computed by $\mathcal{D}$ and $\mathcal{D}'$ (i.e. their strong probabilistic bisimilarity regarding the computation of the *LWA*), we must first show that both DTMCs have an equal stationary distribution.

Theorem 3.1 (Equivalence of Stationary Distributions)**.**

$$prob^\infty(S') = \sum_{d \in [s]_\sim} prob^\infty(d) \tag{12}$$

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

Proof 3.1 (Equivalence of Stationary Distributions)**.**
Let $prob^0(S)$ *be an arbitrary initial distribution for* $\mathcal{D}$ *and let* $prob^0([s]_\sim) = \sum\limits_{d \in [s]_\sim} prob^0(d)$ *be an initial distribution for* $\mathcal{D}'$*. Show that for* $prob^k(S)$ *and* $prob^k([s]_\sim)$*, which are the probability distributions for* $\mathcal{D}$ *and* $\mathcal{D}'$ *at time step* k*, the following holds:*

$$\forall k : prob^k([s]_\sim) = \sum_{d \in [s]_\sim} prob^k(d) \tag{13}$$

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

$$prob^{k+1}([s]_\sim) = \sum_{[d]_\sim \in S'} prob^k([d]_\sim) \cdot prob([d]_\sim, [s]_\sim) \tag{14}$$

$$= \sum_{[d]_\sim \in S'} (\sum_{e \in [d]_\sim} prob^k(e)) \cdot (\sum_{f \in [s]_\sim} prob(d, f)) \tag{15}$$

$$= \sum_{[d]_\sim \in S'} \sum_{e \in [d]_\sim} \sum_{f \in [s]_\sim} prob^k(e) \cdot prob(d, f) \tag{16}$$

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

$$= \sum_{[d]_\sim \in S'} \sum_{e \in [d]_\sim} \sum_{f \in [s]_\sim} prob^k(e) \cdot prob(e, f) \tag{17}$$

$$= \sum_{e \in S} \sum_{f \in [s]_\sim} prob^k(e) \cdot prob(e, f) \tag{18}$$

$$= \sum_{f \in [s]_\sim} \sum_{e \in S} prob^k(e) \cdot prob(e, f) \tag{19}$$

$$= \sum_{d \in [s]_\sim} prob^{k+1}(d) \tag{20}$$

Thereby, $\forall k : prob^k([s]_\sim) = \sum_{d \in [s]_\sim} prob^k(d)$. The corresponding equality for the stationary distributions follows. □

Corollary 3.1 (Equivalence of LWA_0)**.**
Theorem 3.1 and Conditions 4 and 5 of Definition 3.1 imply that the limiting availability LWA_0 satisfies $LWA_0(\mathcal{D}, \mathcal{P}) = LWA_0(\mathcal{D}', \mathcal{P}')$ (with respect to the equivalence classes). Thereby, $LWA_0(\mathcal{D}, \mathcal{P}) = \sum_{s \models \mathcal{P}} prob^\infty(s)$ and consequently $LWA_0(\mathcal{D}', \mathcal{P}') = \sum_{[s]_\sim \models \mathcal{P}'} prob^\infty([s]_\sim)$.

The final step is to show that both $\mathcal{D}$ and $\mathcal{D}'$ compute the *LWA*.

Theorem 3.2 (Equivalence of Limiting Window Availability)**.**
For each $n \in \mathbb{N}$, $LWA_n(\mathcal{D}, \mathcal{P}) = LWA_n(red(\mathcal{D}, \mathcal{P}))$.

Proof 3.2 (Equivalence of Limiting Window Availability)**.**
Follows immediately from Definitions 3.1 and 3.2 plus Theorem 3.1, applied to the steady state distribution as initial distributions. □

3.5. Strong Probabilistic Bisimilarity vs. Decomposition

We have shown that lumping according to Definition 3.1 preserves the precise *LWA*: $\mathcal{D} \sim \mathcal{D}'$. This section discusses that the decomposition does not violate the strong probabilistic bisimilarity either. Both decomposition and recomposition are conducted via the $\otimes$-operator: $\mathcal{D} = \mathcal{D}_1 \otimes \ldots \otimes \mathcal{D}_n$. With $\mathcal{D} \sim \mathcal{D}'$, we derive that $\forall i : \mathcal{D}_i \sim \mathcal{D}'_i$. Consequently, with $\overline{\mathcal{D}'} = \mathcal{D}'_1 \otimes \ldots \otimes \mathcal{D}'_n$ follows that $\overline{\mathcal{D}'} \sim \mathcal{D}$. Thereby, the decomposition and recomposition preserve strong probabilistic bisimilarity regarding the *LWA*. As discussed above, $|\mathcal{D}| \geq |\overline{\mathcal{D}'}| \geq |\mathcal{D}'|$. The unreduced DTMC $\mathcal{D}$ is larger than (or in the case of unfavorable splitting as large as) the locally reduced DTMC $\overline{\mathcal{D}'}$, which is larger than (or equally large if already lumped to the full extent as) the DTMC $\mathcal{D}'$. When bisimilar processes are awarded to different subsystems, they cannot be lumped locally, for the corresponding bisimilar states do not arise within the sub-Markov chains. Then, the potential of lumping cannot be exploited to its full extent. These *missed* opportunities of strongly probabilistic bisimilar states can yet be identified during the recomposition. The successive tracking of bisimilar states during the recomposition allows for the identification and lumping of further strongly probabilistic bisimilar states. Notably, the $\otimes$-operator conserves strong probabilistic bisimilarity under serial execution semantics, but also accounts for parallel execution semantics. The Kronecker product is applicable under maximal parallel execution semantics but not under lesser parallel execution semantics.

4. Example

We outline the example in Section 4.1 and decompose it in Section 4.2. The results are summarized in Section 4.3.

4.1. Outline

Assume a distributed system comprising seven processes $\Pi = \{\pi_1, \ldots, \pi_7\}$ that are connected as shown in Figure 2. A probabilistic scheduler selects one of the processes in each computation step. All processes have an equal probability to be selected for execution. The processes execute the self-stabilizing broadcast algorithm introduced in [2]. For self-containment of

the paper, the next paragraph contains a short informal description of the algorithm. The root process π_1, when executing a computation step, stores

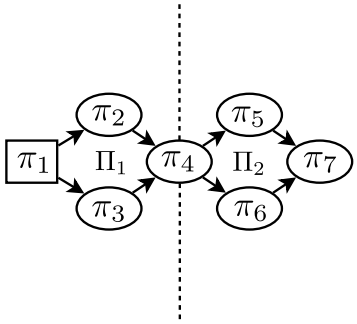

Figure 2: The Example System

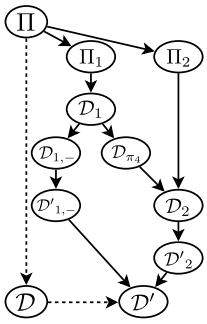

Figure 3: Decomposition

the value 0 in its register in absence of a fault, and a value of 2 if it is perturbed by a fault. Processes π_2 and π_3, when executing and not corrupted, copy the value of R_1 to their respective register.

- π_4 stores value 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 value 2 when $(R_2 = 2 \wedge R_3 = 2) \vee (R_2 = 2 \wedge R_3 = 1) \vee (R_2 = 1 \wedge R_3 = 2)$ or when it is directly corrupted.
- The value 1 is stored otherwise, when both values 0 and 2 are read.

Process π_7 executes the same way with respect to R_5 and R_6 (extending the third option, it stores 1 also when it only reads 1). 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 of $q = 1 - p$. The registers of non-executing processes remain untouched by faults. Figure 3 shows the "decomposition road map". The dotted arrows show the trail without decomposition. The solid arrows show the proposed decomposition with local lumping. The system is split into two subsystems $\Pi_1 = \{\pi_1, \ldots, \pi_4\}$ and $\Pi_2 = \{\pi_4, \ldots, \pi_7\}$ which makes π_4 the only transient. We compute $\mathcal{D}_1$ from subsystem Π_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 reduced DTMC $\mathcal{D}'_2$ has 54 states. Re-composing $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$ results in a DTMC[6] with only 324 states.

4.2. Splitting and Lumping

The decomposition comprises five steps also shown in Figure 3. Each following paragraph describes one step. We set the fault probability[7] to $q = 0.05$.

Step 1: $\mathcal{D}_1 \to \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
First, DTMC $\mathcal{D}_1$ is computed (cf. left-hand side of Figure 4). The probability that the scheduler selects a process of Π_1 is $\mathfrak{s}_{\Pi_1} = \frac{4}{7}$. Hence, each transition in $\mathcal{D}_1$ is to be multiplied by $\frac{4}{7}$, the probability that a process *within* Π_1 is selected (cf. Section 3.1, *Step 1*). Then, all loop transitions (the diagonal entries that reflect the self-targeting transitions in $\mathcal{D}_1$) gain the probability mass of $\frac{3}{7}$, which is the probability that a process *outside* Π_1 is selected for execution. The steady state probability distribution of $\mathcal{D}_1$ is shown in Table 1.

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

Table 1: $\mathcal{D}_1$ Stationary Distribution

We exploit lumping for both *splitting* Markov chains (e.g. *Step 1:* $\mathcal{D}_1 \to \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$) and *pruning* of redundant information (e.g. *Step 2:* $\mathcal{D}_{1,-} \xrightarrow{[]_\sim} \mathcal{D}'_{1,-}$). For splitting, we lump all states in $\mathcal{D}_1$ that have *the first three digits*

[6]The number of states are preferred over the number of transitions to evaluate the reduction as the number of transition relies upon the execution semantics.

[7]Solving DTMCs symbolically includes handling very large terms. The symbolical computation with MatLab consumes about one week at 2.6 GHz on a Pentium 7, single threaded, and about 48 GBytes of main memory. The numerical computation takes less than a second on the same hardware.

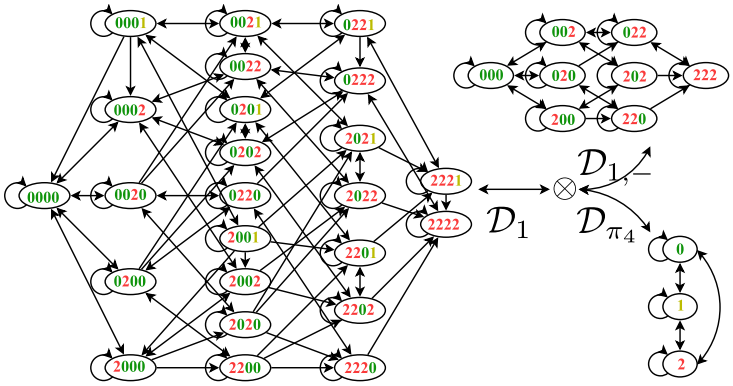

Figure 4: Markov Chain Decomposition

in common (e.g. where $\langle R_1, R_2, R_3, R_4\rangle = \langle 0,0,0,0\rangle$, $\langle 0,0,0,1\rangle$, 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 $\mathcal{D}_{\pi_4}$. The splitting is reversible $\mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4} \to \mathcal{D}_1$. 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 $\langle R_2, R_3\rangle = \{\langle 0,2\rangle, \langle 2,0\rangle\}$) with 2 (the double stroke number indicates the sum within the lumped registers[8]). For the analysis, it is regardless which one of the processes π_2 and π_3 is corrupted as they behave the same (i.e. 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 only one of them is corrupted, suffices to compute the *LWA*. We split $\mathcal{D}_1$ into $\mathcal{D}_{1,-}$ (cf. Table 2) and $\mathcal{D}_{\pi_4}$ (cf. Table 3). The stationary distributions of the split DTMCs are trivially the sums of the stationary distribution that the lumped states comprises. A re-calculation of the stationary distribution is not necessary. We label the transition probabilities in $\mathcal{D}_{\pi_4}$ as shown in Table 3 to later refer to them when computing $\mathcal{D}_2$. To identify lumpable states, we compare their transition probabilities to mutual target states (cf. Definition 3.1). If these are equal, then the tuple under consideration qualifies for lumping.

[8]The double stroke notation is not universally applicable. When two bisimilar processes are not *neighbors* in s, then the position of their lump in s' is unclear. Second, when more than two processes are bisimilar, the notation is not applicable, for instance $\langle 0,0,2\rangle \notin [\langle 0,1,1\rangle]_\sim$. In the present example, it is applicable, though.

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

Table 2: DTMC $\mathcal{D}_{1,-}$, Comprising States $\langle R_1, R_2, R_3\rangle$

↓ 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 3: DTMC $\mathcal{D}_{\pi_4}$, Comprising States $\langle R_4\rangle$

Step 2: $\mathcal{D}_{1,-} \xrightarrow{[\!]_\sim} \mathcal{D}'_{1,-}$.

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 $\langle 2,2,0\rangle$ lumped into $\langle 2, \mathbb{1}\rangle$). The resulting DTMC $\mathcal{D}'_{1,-}$ is shown in Table 4.

↓ 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 4: DTMC $\mathcal{D}'_{1,-}$, Comprising States $\langle R_1, R_2, R_3\rangle$

Step 3: $\mathcal{D}_{\pi_4} \otimes |\Pi_2 \setminus \{\pi_4\}| \to \mathcal{D}_2$.
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 21:

$$\langle 1,2,2,2\rangle \to \langle 2,2,2,2\rangle = \frac{1}{4} \cdot w_4 \tag{21}$$

The transition probability w_4 is given in Table 3. 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 21, are computed with the values in Table 3.

Step 4: $\mathcal{D}_2 \to \mathcal{D}_2'$.
In $\mathcal{D}_2$, 27 state tuples share equivalence classes. The sets can informally be described as pairs of states, where R_4 and R_7 store an equal value respectively, while R_5 and R_6 store unequal values respectively, and each state's R_5 is equal to the mutual other state's R_6 as shown in Figure 5. For in-

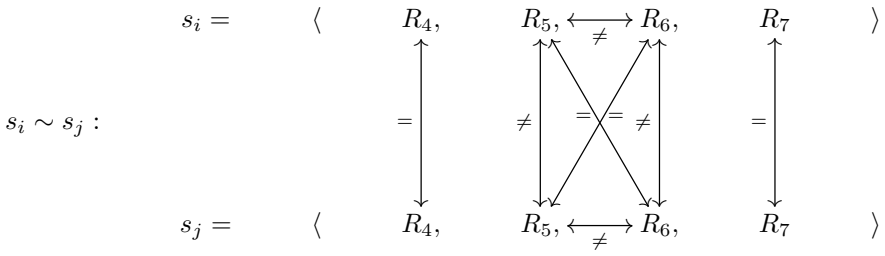

Figure 5: Equivalence Class Identification

stance, states $\langle 0, 0, 2, 2\rangle$ and $\langle 0, 2, 0, 2\rangle$ can be lumped to $\langle 0, \mathbb{1}, 2\rangle$. We write $\langle 0, 0, 2, 2\rangle \oplus \langle 0, 2, 0, 2\rangle = \langle 0, \mathbb{1}, 2\rangle$. Contrary to the $\otimes$-operator (i.e. lumping for splitting) the $\oplus$-operator (lumping for reduction) is irreversible. We explain the $\oplus$-operator in detail at the end of this section. 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 corrupted 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 $\langle x, 2, y\rangle$, and
- $\langle x, 1, 2, y\rangle$ and $\langle x, 2, 1, y\rangle$ form $\langle x, \mathbb{3}, y\rangle$.

Notably, state $\langle R_4, R_5, R_6, R_7\rangle = \langle x, 1, 1, y\rangle$ is not bisimilar to any other state so there is no ambiguous state $\langle x, 2, y\rangle$. 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 into another lump: $\overrightarrow{\langle 1, \mathbb{1}, 0\rangle, \langle 0, \mathbb{1}, 0\rangle}$ (cf. Figure 7). 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 ($prob(\overrightarrow{\langle 1, 1, 0, 0\rangle, \langle 0, 0, 1, 0\rangle})$, $prob(\overrightarrow{\langle 1, 0, 1, 0\rangle, \langle 0, 1, 0, 0\rangle})$), others form the aggregated transition probability ($prob(\overrightarrow{\langle 1, 1, 0, 0\rangle, \langle 0, 1, 0, 0\rangle}) = u_4 \cdot \mathfrak{s}_4(\Pi_2) \cdot \mathfrak{s}_{\Pi_2} = prob(\overrightarrow{\langle 1, 0, 1, 0\rangle, \langle 0, 0, 1, 0\rangle})$). The variable u_4 is the transition probability that R_4 changes its value from 1 to 0 as shown in Table 3, and $\mathfrak{s}_4(\Pi_2)$ is the execution probability of π_4 *within the subsystem*, i.e. $\frac{1}{4}$ in this case. As discussed in *Step 1*, the distinction between the possibilities that either a process *within* 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 regarded this distinction *before* lumping. Hence, the transition probabilities are multiplied by $\frac{4}{7}$.

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

Figure 6: Reduction Example - Transitions

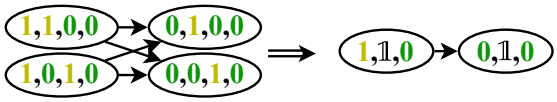

Figure 7: Reduction Example - States

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 (cf. Figure 6), Equation 10 computes the lumped transition probability (cf. Figure 7) given by Equation 22:

$$prob(\overrightarrow{\langle 1, \mathbb{1}, 0\rangle, \langle 0, \mathbb{1}, 0\rangle}) = \frac{prob(\overrightarrow{\langle 1,1,0,0\rangle,\langle 0,1,0,0\rangle}) \cdot prob^{\infty}(\langle 1,1,0,0\rangle) + prob(\overrightarrow{\langle 1,0,1,0\rangle,\langle 0,0,1,0\rangle}) \cdot prob^{\infty}(\langle 1,0,1,0\rangle)}{prob^{\infty}(\langle 1,1,0,0\rangle) + prob^{\infty}(\langle 1,0,1,0\rangle)} \tag{22}$$

All equivalence classes in $\mathcal{D}_2$ are lumped this way, resulting in $\mathcal{D}'_2$.

Step 5: $\mathcal{D}'_{1,-} \otimes \mathcal{D}'_2 \rightarrow \mathcal{D}'$.
With $\mathcal{D}'_{1,-}$ and $\mathcal{D}'_2$ at hand, $\mathcal{D}'$ is constructed. Notably, both DTMCs $\mathcal{D}'_{1,-}$ and $\mathcal{D}'_2$ execute computation steps *in parallel* as their probabilities have been weighted. For re-composition, each entry of $\mathcal{D}'_{1,-}$ is multiplied by each entry of $\mathcal{D}'_2$ and parallel steps are split that exclusively either a process in $\mathcal{D}'_{1,-}$ executes or in $\mathcal{D}_2$. This distinguishes the $\otimes$-operator from the Hadamard/Kronecker product for concurrent composition (i.e. maximal parallel execution semantics). The coordinates are labeled row i and column j

in $\mathcal{D}'_{1,-}$, and k and l in $\mathcal{D}'_2$, respectively. Algorithm 1 is generally applicable. The cardinality of the composed state space is $|S'|$.

$\mathcal{D}' = zeros(|S'|);$
for $j = 1 : |S'_{1,-}|$ **do**
  for $l = 1 : |S'_2|$ **do**
    for $i = 1 : |S'_{1,-}|$ **do**
      for $k = 1 : |S'_2|$ **do**
        if $i \neq j \wedge l \neq k$ **then**
          $\mathcal{D}'((j-1)\cdot|S'_2|+l,(i-1)\cdot|S'_2|+l) =$
            $\mathcal{D}'_{1,-}((j-1)\cdot|S'_2|+l,(i-1)\cdot|S'_2|+l)+$
            $\mathcal{D}'_{1,-}(j,i)\cdot\mathcal{D}'_2(l,k)\cdot\mathfrak{s}(\Pi_{1,-});$
          $\mathcal{D}'((j-1)\cdot|S'_2|+l,(j-1)\cdot|S'_2|+k) =$
            $\mathcal{D}'_{1,-}((j-1)\cdot|S'_2|+l,(j-1)\cdot|S'_2|+k)+$
            $\mathcal{D}'_{1,-}(j,i)\cdot\mathcal{D}'_2(l,k)\cdot\mathfrak{s}(\Pi_2);$
        else
          $\mathcal{D}'((j-1)\cdot|S'_2|+l,(i-1)\cdot|S'_2|+k) =$
            $\mathcal{D}'_{1,-}((j-1)\cdot|S'_2|+l,(i-1)\cdot|S'_2|+k)+$
            $\mathcal{D}'_{1,-}(j,i)\cdot\mathcal{D}'_2(l,k);$

Algorithm 1: The $\otimes$-Operator

Here, $|S'_{1,-}|$ is six, $|S'_2|$ is 54 (therefore, $|S'| = |S'_{1,-}| \cdot |S'_2| = 324$), $\mathfrak{s}(\Pi_{1,-})$ computes to $\frac{3}{7}$, and $\mathfrak{s}(\Pi_2)$ computes to $\frac{4}{7}$. The final step to let $\mathcal{D}'$ compute the *LWA* (the stationary distribution is known, cf. Table 1), is to set the transition probability $\mathcal{D}'(1,1) := 1$, and $\forall m, 1 < m \leq 324 : \mathcal{D}'(1,m) := 0$ (cf. [2, Sec.4]). Thereby, the legal state becomes absorbing. The *probability mass drainage* starting in the limit (i.e. with the stationary distribution) computes the *LWA*.

4.3. Conclusion

The DTMC $\mathcal{D}'$ computes the *LWA*. Figure 8 shows the strictly monotonous increase of probability mass over the first 1000 computation steps. If the sys-

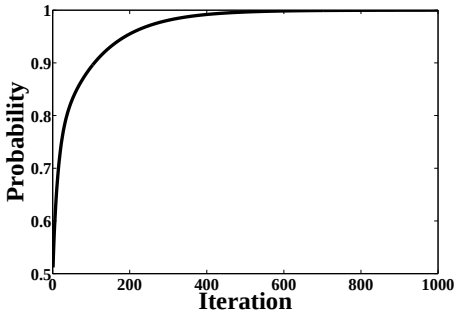

Figure 8: LWA^{1001} of the Example System

tem is supposed to exceed a certain lower availability bound within a distinct number of computation steps, we then can use Figure 8 that gives the associated amount of time that must be spent to meet the demand. Furthermore, we can investigate the *probability mass drainage* throughout the states over time. For reference purposes in the subsequent discussion, we order the lumped states $\langle 0,0,0,0,0,0,0\rangle, \ldots \langle 2,2,2,2,2,2,2\rangle$. When comparing the

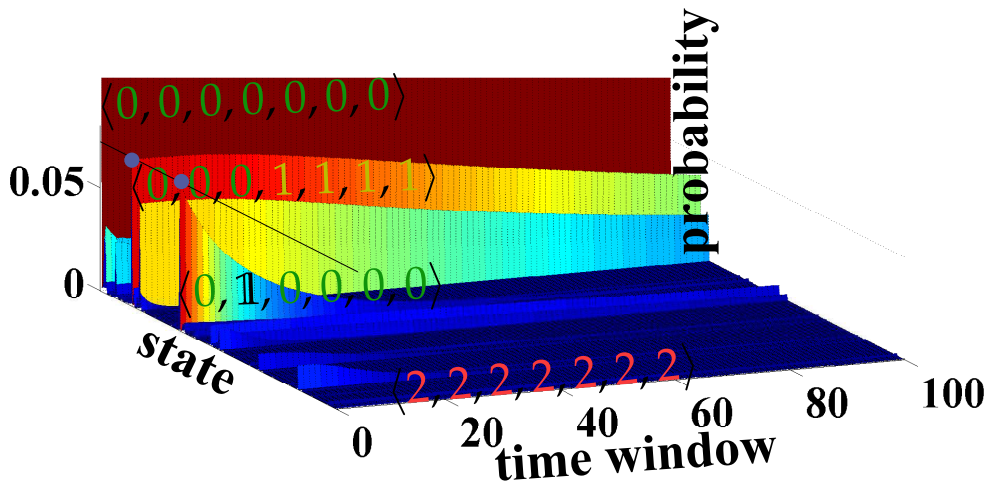

Figure 9: Probability Mass Drain

degree of probability mass drainage within each state, as shown in Figure 9, then two prominent states for discussion can be identified, namely the 43^{rd} state $\langle 0,0,0,1,1,1,1\rangle$ and the 109^{th} state $\langle 0,\mathbb{1},0,0,0,0\rangle$. Although initially equipped with a similar amount of probability mass of about 0.07, the 109^{th} state leaks its probability mass rapidly, while the 43^{rd} state is drained at a slower pace as shown in Figures 10 and 11. The initial motivation of computing the *LWA* was to find the minimal required amount of time to obtain a certain probability with which a non-masking fault-tolerant system can mask faults under perfect 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*, [19]). When looking for candidate states for which counter measures should be applied, then the 43^{rd} state is obviously a more suitable point of attack than the 109^{th} state. Either preventing the 43^{rd} state

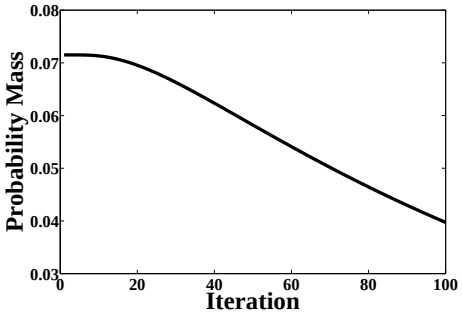

Figure 10: 43^{rd} State

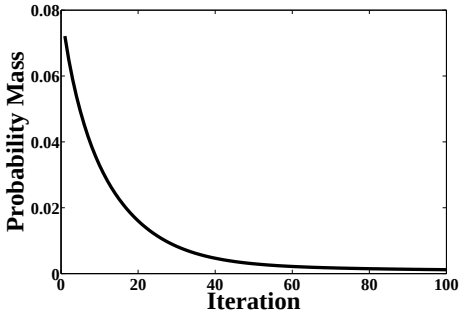

Figure 11: 109^{th} State

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 goal might be that the system must be stable after at most 30 computation steps with a maximal probability. The designer might have the possibility to *prevent* certain states[9] at an extra cost. But which states should be prevented? Having resources for a distinct number of states to counter together with a list of all states sorted by the probability mass they withhold at time point 30, the top states (considering that lumped states comprise multiple states) are the right targets.

5. Conclusion

The likelihood of a fault-tolerant distributed system to provide its service correctly can be computed with the help of Markov models. One of the main obstacles in precisely computing this measure is the size of a system's corresponding Markov model which grows exponentially in system size (i.e. number of states of the corresponding Markov model). Hence, Markov model analysis soon becomes intractable. Although lumping is a well-established method for reducing a Markov model to a possibly tractable size, when applying lumping, the whole Markov model must be constructed in the first place before (or upon) reduction. In order to circumvent the construction of the whole Markov model, which is prone to state space explosion,

[9]In a probabilistic context, the designer might be able to lower the stationary probability for certain states.

we introduced a decomposition that allows for local application of lumping on the Markov chains of decomposed subsystems. Combining both decomposition and local lumping allows for the construction of strongly probabilistic bisimilar sub-Markov chains. The approach has been demonstrated by the example of a self-stabilizing system. Future work will investigate the classes of semi-hierarchical and heterarchical systems on the one hand, and the cooperative effect of spatial and temporal redundancy on the other hand.

References

[1] A. Avižienis, J.-C. Laprie, B. Randell, C. Landwehr, Basic Concepts and Taxonomy of Dependable and Secure Computing, IEEE Transactions on Dependable and Secure Computing 1 (2004) 11–33.

[2] N. Müllner, O. Theel, The Degree of Masking Fault Tolerance vs. Temporal Redundancy, Proc. of WAINA'11 (2011) 21–28,doi:http://doi.ieeecomputersociety.org/10.1109/WAINA.2011.137.

[3] N. Müllner, A. Dhama, O. Theel, Deriving a Good Trade-off Between System Availability and Time Redundancy, in: Symposium on UbiCom Frontiers - Innovative Research Systems and Technologies, 2009.

[4] J.-P. Katoen, D. Klink, M. Leucker, V. Wolf, Three-Valued Abstraction for Probabilistic Systems, Journal on Logic and Algebraic Programming (2012) 1–55.

[5] L. Lamport, Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers, Addison-Wesley, 2002.

[6] N. Müllner, O. Theel, M. Fränzle, Combining Decomposition and Reduction for State Space Analysis of a Self-Stabilizing System, Proc. of AINA'12 0 (2012) 936–943, Best Paper Award. doi:http://doi.ieeecomputersociety.org/10.1109/AINA.2012.127.

[7] J. G. Kemeny, J. L. Snell, Finite Markov chains, reprint Edition, University Series in Undergraduate Mathematics, VanNostrand, New York, 1969.

[8] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, New York, NY, USA, 1996.

[9] V. Mertsiotakis, Approximate Analysis Methods for Stochastic Process Algebras, Ph.D. thesis, Universität Erlangen-Nürnberg - Technische Fakultät (1998).

[10] L. Capra, C. Dutheillet, G. Franceschinis, J. Ilié, On the Use of Partial Symmetries for Lumping Markov Chains, SIGMETRICS Perform. Eval. Rev. 28 (2001) 33–35.

[11] S. Derisavi, Signature-based Symbolic Algorithm for Optimal Markov Chain Lumping, in: QEST'07, 2007, pp. 141–150.

[12] S. Derisavi, A Symbolic Algorithm for Optimal Markov Chain Lumping, in: TACAS'07, 2007, pp. 139–154.

[13] S. Derisavi, P. Kemper, W. H. Sanders, Lumping Matrix Diagram Representations of Markov Models, in: Proc. of DSN'05, 2005.

[14] S. Derisavi, H. Hermanns, W. H. Sanders, Optimal State-space Lumping in Markov Chains, Inf. Process. Lett. 87 (2003) 309–315.

[15] S. Kulkarni, Component Based Design of Fault-Tolerance, Ph.D. thesis, Department of Computer and Information Science, The Ohio State University, Columbus Ohio, USA (1999).

[16] S. Kulkarni, A. Arora, Compositional Design of Multitolerant Repetitive Byzantine Agreement, LNCS 1346 (1997) 169–181.

[17] A. Arora, S. Kulkarni, Designing Masking Fault-Tolerance via Nonmasking Fault-Tolerance, IEEE Trans. on Soft. Eng. 24 (6) (1998) 435–450.

[18] S. S. Kulkarni, A. Arora, Automating the Addition of Fault-Tolerance, in: Proceedings of the 6th International Symposium on Formal Techniques in Real-Time and Fault-Tolerant Systems, FTRTFT '00, Springer-Verlag, London, UK, 2000, pp. 82–93.

[19] S. Tixeuil, Algorithms and Theory of Computation Handbook, Second Edition, Chapman & Hall/CRC Applied Algorithms and Data Structures, CRC Press, Taylor & Francis Group, 2009, Ch. Self-stabilizing Algorithms, pp. 26.1–26.45.