Lumping Matrix Diagram Representations of Markov Models *

Salem Derisavi, Peter Kemper, William H. Sanders
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign, IL, U.S.A.
Email: {derisavi, whs}@crhc.uiuc.edu

Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany
Email: peter.kemper@udo.edu

Abstract

Continuous-time Markov chains (CTMCs) have been used successfully to model the dependability and performability of many systems. Matrix diagrams (MDs) are known to be a space-efficient, symbolic representation of large CTMCs. In this paper, we identify local conditions for exact and ordinary lumpings that allow us to lump MD representations of Markov models in a compositional manner. We propose a lumping algorithm for CTMCs that are represented as MDs that is based on partition refinement, is applied to each level of an MD directly, and results in an MD representation of the lumped CTMC. Our compositional lumping approach is complementary to other known model-level lumping approaches for matrix diagrams. The approach has been implemented, and we demonstrate its efficiency and benefits by evaluating an example model of a tandem multi-processor system with load balancing and failure and repair operations.

1 Introduction

Model-based evaluation of computer and communication systems is often done through simulation. In many cases, the desired results could, in principle, also be derived through analysis of the underlying CTMCs; however, in practice, the size of the systems of equations that would need to be solved is prohibitive. This "largeness problem" has motivated much research in the construction and numerical solution of CTMCs. State-space lumping (e.g., [2, 13]) is a well-known approach that reduces the size of a CTMC by considering the quotient of the CTMC with respect to an equivalence relation that preserves the Markov property and supports the desired reward measures defined on the CTMC. By solving the smaller lumped CTMC, we can compute exact results for the larger CTMC, and therefore measures of interest for the model. We classify the many publications on lumping into three categories.

State-level lumping applies directly to a given generator matrix $\mathbf{Q}$ of a CTMC and computes $\tilde{\mathbf{Q}}$ of the lumped CTMC. It yields an optimal partition, i.e., the smallest possible lumped CTMC. However, the size of $\mathbf{Q}$ limits its application. Efficient algorithms have been designed, e.g., see [9] for the fastest known algorithm.

Model-level lumping is used to generate $\widetilde{\mathbf{Q}}$ directly from a model description. Hence, it is specific to a modeling formalism. The approach is based on symmetry detection among components of a compositional model. Results are known for a variety of formalisms, such as stochastic well-formed networks (SWN) [4, 8], stochastic activity networks [18], stochastic automata networks [1], and Kronecker representations, among others. While the second approach manages to avoid processing a large matrix $\mathbf{Q}$, it is limited to those symmetries that can be identified from a given model description. Hence, in general, it does not obtain an optimal lumping, unlike the first approach.

Compositional lumping applies the state-level lumping approach to individual components of a compositional model. The original components are replaced by lumped and "equivalent" components during generation of $\widetilde{\mathbf{Q}}$. Like model-level lumping, this approach is formalism-dependent; specifically, it relies on properties of the composition operators. For instance, based on the fact that lumping is a congruence with respect to parallel composition in a number of process algebra formalisms and stochastic automata networks (SANs), compositional lumping can be used in those formalisms, e.g., see [12, 3].

Note that in principle, approaches from different categories can be combined. For example, a compositional approach may yield smaller lumped components that can then be fed to a model-level technique for further reduction of the CTMC. Finally, state-level lumping can be applied to obtain optimal reduction of the resulting matrix.

Despite all the state-space reductions the techniques mentioned above offer, they may still yield a very large ma-

*This material is based upon work supported in part by Pioneer Hi-Bred, the National Science Foundation under Grant No. 0086096, by DAAD and Deutsche Forschungsgemeinschaft, SFB 559. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the supporting organisations.

trix $\widetilde{\mathbf{Q}}$. Symbolic data structures like multi-terminal binary decision diagrams (MTBDD) and matrix diagrams (MDs) as well as Kronecker representations are suitable for representing $\mathbf{Q}$ in a space-efficient manner for numerical analysis. In particular, MD representations of $\mathbf{Q}$ [6] can be derived directly from many formalisms with the help of a symbolic state-space exploration as well as from a given sparse matrix or Kronecker representation of $\mathbf{Q}$; see [16] for a recent overview paper on symbolic representations. Generation of an MD for models with state spaces on the order of 10^{1000} states can be achieved, thanks to symbolic data structures and the so-called "saturation technique" [5]. However, the numerical solution is limited to models with state spaces on the order of 10^8 states, with iteration vectors being the bottleneck for space. There are only a few results that address the problem of lumping of Markov chains represented as MDs. In [10], a model-level lumping technique is proposed for MDs that result from state-sharing compositional models. Like other model-level lumping techniques that can only exploit lumpings that occur due to symmetric composition of components, our technique in [10] can only find symmetries that occur in MD levels that correspond to identical components of a model. To the best of our knowledge, there is no algorithm that can exploit lumpings in the individual levels of an MD.

In this paper, we present a new compositional lumping algorithm that is useful for exact and ordinary lumping of Markov chains represented as MDs without knowledge of the modeling formalism from which the MDs were generated. Our approach relies on local conditions, i.e., conditions on individual levels of the MD. Since our algorithm locally processes MD nodes that are dramatically smaller than the matrix represented by the MD, it is computationally inexpensive (compared to state-space generation and numerical solution) at the price that it does not necessarily achieve an optimal lumping for the overall CTMC. As we will see in Section 5, our algorithm significantly reduces the space and time requirements of MD-based numerical solution algorithms while incurring a negligible time overhead. Our compositional technique is complementary to the model-level lumping presented in [10]. In other words, we can apply compositional and model-level lumping techniques simultaneously on a compositional model whose underlying CTMC is represented as an MD. Our work is related to [11] in that we argue on the level of a block structured matrix to observe lumpability, but unlike [11], is not limited to Kronecker matrices and stochastic automata networks. It is related to [3] in that we have a local condition but do not separate local and synchronized actions as was done for the automata theoretic approach in [3].

The paper is organized as follows. We recall basic definitions in Section 2. In Section 3, we present the main result of the paper, which is two theorems in which lumpability conditions on the matrices of a single level of an MD are proved to be sufficient for the entire MD to be lumpable. Section 4 presents an algorithm that lumps the MD of a CTMC level by level, and Section 5 analyzes a tandem multi-processor system model to illustrate the applicability and benefits of the approach.

2 Background and Definitions

In this section, we recall fundamental definitions. For notation, all matrices and vectors are typeset with bold characters (upper-case letters for matrices and lower-case letters for vectors), and their rows and columns are indexed starting from 1. The element of matrix $\mathbf{A}^{(n\times m)}$ (of size $n \times m$) in row i and column j is referred to as $\mathbf{A}(i,j)$. For a matrix $\mathbf{A}^{(n\times m)}$, $rs(\mathbf{A}) = \mathbf{B}^{(n\times n)}$ is a diagonal matrix such that $\mathbf{B}(i,i) = \sum_{1\le j\le m} \mathbf{A}(i,j)$. Sets are assumed to be finite.

We specify a CTMC by a state space $\mathcal{S} = \{1,\ldots,|\mathcal{S}|\}$ and a state transition rate matrix $\mathbf{R}$, or a generator matrix $\mathbf{Q} = \mathbf{R} - rs(\mathbf{R})$, in which $\mathbf{R}(i,j)$ is the rate of the transition that changes the state of the CTMC from i to j. Consider a partition $\mathcal{P} = \{C_1,\ldots,C_{\widetilde{n}}\}$ of $\mathcal{S}$. $C_1,\cdots,C_{\widetilde{n}}$ are the equivalence classes of partition $\mathcal{P}$, or in short, *classes* of $\mathcal{P}$. Any two states $x, \hat{x}$ in a class C_i of $\mathcal{P}$ are called *equivalent*, and that is indicated by $x \approx \hat{x}$.

Often, the final goal of a CTMC analysis is not the computation of the stationary or transient probability of its states. Instead, it is the computation of high-level measures such as performance, dependability, and/or availability. Many of those high-level measures can be computed using reward values associated with each state of the CTMC (i.e., rate rewards) and the stationary and transient probability vectors. By incorporating rate rewards and an initial probability vector with the CTMC, we obtain an MRP.

Definition 1. *A Markov reward process (MRP) M is a 4-tuple $(\mathcal{S},\mathbf{Q},\mathbf{r},\boldsymbol{\pi}^{ini})$ where $\mathcal{S} = \{1,\ldots,|\mathcal{S}|\}$ is the state space of a CTMC, $\mathbf{Q}^{(|\mathcal{S}|\times|\mathcal{S}|)}$ is the generator matrix of the CTMC, $\mathbf{r}$ is a row vector $\mathbf{r}$ (each of size $|\mathcal{S}|$ assigning reward value $\mathbf{r}(i)$ to state i), and $\boldsymbol{\pi}^{ini}(i)$ is the probability that the CTMC is in state i at time 0 $(1 \le i \le |\mathcal{S}|)$.* ■

Given an MRP and a number of high-level measures expressed in terms of $\mathbf{r}$ and $\boldsymbol{\pi}^{\text{ini}}$, we can compute the measures by numerically solving the underlying CTMC. The larger the state space of the CTMC, the more time-consuming that numerical solution will be. Sometimes a lumped CTMC can be used to obtain the desired measures. That can only be done if the MRP that is based on that CTMC satisfies a set of conditions. Three of the most important sets of conditions (and the types of lumpings they lead to) are outlined in Definition 2. The lumped MRP, which includes the lumped CTMC, is obtained using Theorem 2. For a compact notation, we use the identities $\mathbf{A}(i,C) = \sum_{j\in C} \mathbf{A}(i,j)$,

$\mathbf{A}(C,j) = \sum_{i\in C}\mathbf{A}(i,j)$, and $\mathbf{a}(C) = \sum_{i\in C}\mathbf{a}(i)$, in which C is a set of valid column or row indices.

Definition 2. *Consider an MRP $M = (\mathcal{S}, \mathbf{Q}, \mathbf{r}, \boldsymbol{\pi}^{ini})$ and a partition $\mathcal{P}$ of $\mathcal{S}$. Then, with respect to $\mathcal{P}$, M is*
1. ordinarily lumpable if $\mathbf{Q}(s,C') = \mathbf{Q}(\hat{s},C')$ and $\mathbf{r}(s) = \mathbf{r}(\hat{s})$ for all $C, C' \in \mathcal{P}$, all (equivalent states) $s, \hat{s} \in C$.
2. exactly lumpable if $\mathbf{Q}(C',s) = \mathbf{Q}(C',\hat{s})$ and $\boldsymbol{\pi}^{ini}(s) = \boldsymbol{\pi}^{ini}(\hat{s})$ for all $C, C' \in \mathcal{P}$, all (equivalent states) $s, \hat{s} \in C$. ■

In Definition 2, we gave lumpability conditions in terms of $\mathbf{Q}$. Since we will use the $\mathbf{R}$ matrix in the following, we state the lumpability conditions in terms of $\mathbf{R}$ instead of $\mathbf{Q}$. Theorem 1 specifies such conditions. Strictly speaking, Definition 2 defines lumping for an MRP and not for the CTMC embedded in it. However, in the rest of the paper, we will also speak of lumping of CTMCs, of their corresponding $\mathbf{Q}$ or $\mathbf{R}$ matrices, of the MD representation of those matrices, or of nodes and levels (defined in Section 3) of the MD, under the assumption that the reward vectors and initial probability distribution are such that they satisfy the requirements of Definition 2.

Theorem 1. *Consider an MRP $M = (\mathcal{S}, \mathbf{Q}, \mathbf{r}, \boldsymbol{\pi}^{ini})$ such that $\mathbf{Q} = \mathbf{R} - rs(\mathbf{R})$ where $\mathbf{R}$ is the state transition rate matrix of the CTMC of M. Then, with respect to a partition $\mathcal{P}$, M is*
1. ordinarily lumpable if $\mathbf{R}(s,C') = \mathbf{R}(\hat{s},C')$ and $\mathbf{r}(s) = \mathbf{r}(\hat{s})$ for all $C, C' \in \mathcal{P}$, all (equivalent states) $s, \hat{s} \in C$.
2. exactly lumpable if $\mathbf{R}(C',s) = \mathbf{R}(C',\hat{s})$, $\mathbf{R}(s,\mathcal{S}) = \mathbf{R}(\hat{s},\mathcal{S})$, and $\boldsymbol{\pi}^{ini}(s) = \boldsymbol{\pi}^{ini}(\hat{s})$ for all $C, C' \in \mathcal{P}$ and all (equivalent states) $s, \hat{s} \in C$.

Proof. (a) By the definition of $\mathbf{Q}$, we have $\mathbf{Q}(s,C') = \mathbf{R}(s,C') = \mathbf{R}(\hat{s},C') = \mathbf{Q}(\hat{s},C')$, if $s, \hat{s} \notin C'$. If $s, \hat{s} \in C'$, we have $\mathbf{Q}(s,C') = \mathbf{R}(s,C') - \mathbf{R}(s,\mathcal{S}) = \mathbf{R}(s,C') - \sum_{C''\in\mathcal{P}}\mathbf{R}(s,C'') = \mathbf{R}(\hat{s},C') - \sum_{C''\in\mathcal{P}}\mathbf{R}(\hat{s},C'') = \mathbf{Q}(\hat{s},C')$ since condition (a) holds for all $C', C'' \in \mathcal{P}$.
(b) Similarly, if $s, \hat{s} \notin C'$ the result is immediate. Otherwise, $\mathbf{Q}(C',s) = \mathbf{R}(C',s) - \mathbf{R}(s,\mathcal{S}) = \mathbf{R}(C',\hat{s}) - \mathbf{R}(\hat{s},\mathcal{S}) = \mathbf{Q}(C',\hat{s})$ since $\mathbf{R}(C',s) = \mathbf{R}(C',\hat{s})$ and $\mathbf{R}(s,\mathcal{S}) = \mathbf{R}(\hat{s},\mathcal{S})$. ■

Note that the converse of Theorem 1 does not hold, since $\mathbf{Q}$ tolerates different rates for self-loops of equivalent states while those rates are distinguishable in $\mathbf{R}$. Now that we know when a partition satisfies the ordinary and/or exact lumpability conditions, we need to know how to derive a lumped MRP.

Theorem 2. *Let $M = (\mathcal{S}, \mathbf{Q}, \mathbf{r}, \boldsymbol{\pi}^{ini})$ be (ordinarily or exactly) lumpable with respect to a partition $\mathcal{P} = \{C_1, \ldots, C_{\tilde{n}}\}$ of $\mathcal{S}$. Then $\widetilde{M} = (\widetilde{\mathcal{S}}, \widetilde{\mathbf{Q}}, \widetilde{\mathbf{r}}, \widetilde{\boldsymbol{\pi}}^{ini})$ is the lumped MRP such that $\widetilde{\mathcal{S}} = \{1, \ldots, \tilde{n}\}$*

$$\widetilde{\mathbf{Q}}(\tilde{i},\tilde{j}) = \begin{cases} \mathbf{Q}(i, C_{\tilde{j}}) & \text{for arbitrary } i \in C_{\tilde{i}} \quad \textit{(ordinary)} \\ \mathbf{Q}(C_{\tilde{i}}, j) & \text{for arbitrary } j \in C_{\tilde{j}} \quad \textit{(exact)} \end{cases}$$

$\widetilde{\mathbf{r}}(\tilde{i}) = \mathbf{r}(C_{\tilde{i}})/|C_{\tilde{i}}|$ and $\widetilde{\boldsymbol{\pi}}^{ini}(\tilde{i}) = \boldsymbol{\pi}^{ini}(C_{\tilde{i}})$.

For a proof of Theorem 2, we refer to [2]. Using Theorem 2, one can show that all reward measures of M that are based on $\mathbf{r}$ and $\boldsymbol{\pi}^{\text{ini}}$ can be computed using $\widetilde{\mathbf{r}}$ and $\widetilde{\boldsymbol{\pi}}^{\text{ini}}$.

3 Compositional Lumping of MDs

In this section, we first define MDs [6, 15] and introduce a notation for them. Then, we describe the concepts based on which the compositional lumping works. In particular, we will describe a set of sufficient lumpability conditions on the nodes of one level of the MD, and prove that it satisfies lumpability conditions on the overall MD.

An MD [6] is a symbolic data structure that represents real-valued matrices. MDs are related to MDDs (Multi-valued Decision Diagrams), edge-valued decision diagrams, and Kronecker representations [17]. MDs have proven to be particularly useful in representing large state transition matrices of compositional Markovian models. Refer to [6, 15] for details on MDs and algorithms for manipulating them. The formal definition of MDs that we use in the following is slightly different from that in [6, 15]. The resulting data structures remain the same, but the formal treatment is more concise.

An ordered MD is a connected directed acyclic graph (DAG) with a finite number of nodes and a unique root node. Each node in an MD has a unique index. We refer to the node with index n as R_n. R_1 is the root node. All paths from R_1 to any given node are of the same length. Consequently, the set of nodes can be partitioned into L subsets, called *levels* of the MD, based on their distance from the root node. Nodes in level $i \in \{1, \ldots, L\}$ have distance $i-1$ from the root node, and the set of their indices is represented by N_i. The top level (level 1) has only the root node in it, i.e., $N_1 = \{1\}$. By the definition of levels, arcs only connect nodes of adjacent levels i and $i+1$.

R_{n_i}, a node at level i ($n_i \in N_i$), is a matrix whose row index set S_{n_i} and column index set S'_{n_i} are subsets of a finite index set $\mathcal{S}_i$ that is the same for all nodes at level i[1]. $R_{n_i}^{(S_{n_i}\times S'_{n_i})}$ indicates that S_{n_i} and S'_{n_i} are respectively the row and column index sets for R_{n_i}. As a result, we can refer to row $s_i \in S_{n_i}$, column $s'_i \in S'_{n_i}$, or entry (s_i, s'_i) of R_{n_i}. Matrix entries are finite formal sums $R_{n_i}(s_i,s'_i) = \sum_{n_{i+1}\in N_{i+1}} r_{n_i,n_{i+1}}(s_i,s'_i)\cdot R_{n_{i+1}}$ with real values $r_{n_i,n_{i+1}}(s_i,s'_i)$ and references to nodes at level n_{i+1}, if $i \neq L$. At level $i = L$, $R_{n_i}(s_i,s'_i)$ is a real value, and therefore R_{n_i} is a real-valued matrix. Terms $r_{n_i,n_{i+1}}(s_i,s'_i)\cdot R_{n_{i+1}}$ with $r_{n_i,n_{i+1}}(s_i,s'_i) \neq 0$ correspond to an arc from node R_{n_i} to $R_{n_{i+1}}$. We assume that the MD is *reduced*, i.e., at any level i, no two nodes are equal. Otherwise, one node can be removed, and references at level $i-1$ can be appropriately renamed. The assumption that the MD has been reduced is the basis of the efficiency of

[1] Matrices of different dimensions may result from the computation of reachable state space by projections on an MD.

the MD data structure. In MD theory, the removal of duplicates is called "quasi-reduction".

In the following paragraphs, we describe how to merge adjacent levels of an MD in bottom-up and top-down manners. We do so to show that (1) each MD node R_{n_i} results in a real-valued matrix $\mathbf{R}_{n_i}$, and (2) we can avoid an overwhelming notation by considering MDs with 3 levels instead of MDs with an arbitrary number of levels, with no loss of generality.

We first consider merging adjacent levels from the bottom up and show that item (1) holds. Observe that it holds for nodes at level $i = L$ by definition. At level $i < L$, $\mathbf{R}_{n_i}$ is a block-structured matrix in which each block is defined as $\mathbf{R}_{n_i}(s_i, s_i') = \sum_{n_{i+1} \in N_{i+1}} r_{n_i,n_{i+1}}(s_i, s_i') \cdot \mathbf{R}_{n_{i+1}}$. In other words, we replace node references with matrix references and resolve the formal sum by scalar matrix multiplication and matrix addition. We can thus merge levels i through L by replacing nodes R_{n_i} with matrices $\mathbf{R}_{n_i}$ (for all $n_i \in N_i$) and removing all levels $i+1$ to L without affecting the matrix that the overall MD represents. Note that the row and column index sets of $\mathbf{R}_{n_i}$ will be subsets of (and not necessarily equal to) $\times_{k=i}^{L} \mathcal{S}_k$. Therefore, we need special definitions for the abovementioned operators to resolve the sum. In particular, consider $a \in \mathbb{R}$, $\mathbf{A}^{(S_A \times S_A')}$, $\mathbf{B}^{(S_B \times S_B')}$, and $S_A, S_A', S_B, S_B' \subseteq S$. We then define the scalar multiplication of a and $\mathbf{A}$ as $(a \cdot \mathbf{A})^{(S_A \times S_A')}(s, s') = a \cdot \mathbf{A}(s, s')$ for all $s \in S_A$ and $s' \in S_A'$. We also define the matrix addition of $\mathbf{A}$ and $\mathbf{B}$ as $\mathbf{C}^{(S_A \cup S_B \times S_A' \cup S_B')}(s, s') = \mathbf{A}(s, s') + \mathbf{B}(s, s')$ in which we assume, for simplicity of notation, that $\mathbf{A}(s, s') = 0$ if $s \in S \backslash S_A$ or $s' \in S \backslash S_A'$. We make a similar assumption for $\mathbf{B}$. Terms in a formal sum with $r_{n_i,n_{i+1}}(s_i, s_i') = 0$ result in a zero matrix and need not be considered in a summation. Furthermore, $\mathbf{R}_{n_i}$ matrices need not be square matrices in general, since R_{n_i} nodes are not necessarily square. For $i = 1$, we reduce the number of levels to one and end up with the real-valued matrix $\mathbf{R}_1$, which corresponds to the root node R_1. It is a flat representation of the matrix that the original MD represents. $\mathbf{R}_1$ has a nested block structure, and each of its column and row indices is a subset of $\times_{i=1}^{L} \mathcal{S}_i$. Therefore, a row or a column index can be represented as $s = (s_1, \ldots, s_L)$. When an MD is used to represent the $\mathbf{R}$ or the $\mathbf{Q}$ matrix of a CTMC, $s = (s_1, \ldots, s_L)$ is a state of the CTMC, and we call s_i a *substate* of s. Much as levels can be merged in a bottom-up manner, we can reduce the number of levels by merging in a top-down manner, i.e., starting at the top level. For this paper, we need to test an arbitrary level l for lumpability conditions local to that level. Therefore, it is natural to merge all levels $l+1, \ldots, L$ into a single level and all levels $1, \ldots, l-1$ into another single level when $1 < l < L$. The cases where $l = 1$ (and $l = L$) need special attention. We first add an artificial level 0 (or $L+1$) to the MD that has a single node of size 1×1 consisting of entry 1, and establish appropriate pointers from level 0 to 1 (or L to $L+1$) such that the new MD represents the same overall matrix. Then, we perform the merging operations mentioned above. Eventually, the merging leads us to consider, for most of our discussion and without loss of generality, an MD of 3 levels and to focus on level 2 for local lumping purposes instead of considering an MD of L levels and focusing on level l. Once again, note that the sole purpose of our merging argument is to simplify the notation and understanding of the material. The implementation of the algorithm does not perform any merging operation.

Compositional Lumping. We have shown so far how an MD can represent the state transition rate matrix of a CTMC. To discuss lumping conditions according to Theorem 1, we need to augment the CTMC to an MRP by specifying the rewards and the initial probability distribution. In particular, to discuss local lumping conditions, it is necessary to make rewards and probability distribution decomposable. More specifically, $\mathbf{r}$ is an arbitrary reward vector on $\mathcal{S}$, and when we discuss ordinary lumping, we restrict its representation to $\mathbf{r}(s) = g(f_1(s_1), f_2(s_2), f_3(s_3))$ (i.e., a function built upon functions of substates at each level), in which $s = (s_1, s_2, s_3)$, $g : \mathbb{R}^3 \to \mathbb{R}$, $f_1 : \mathcal{S}_1 \to \mathbb{R}$, $f_2 : \mathcal{S}_2 \to \mathbb{R}$, and $f_3 : \mathcal{S}_3 \to \mathbb{R}$. Likewise, $\boldsymbol{\pi}^{\text{ini}}$ is also an arbitrary initial probability distribution on $\mathcal{S}$, and when we discuss exact lumping, we restrict its representation to $\boldsymbol{\pi}^{\text{ini}}(s) = g_\pi(f_{\pi,1}(s_1), f_{\pi,2}(s_2), f_{\pi,3}(s_3))$, in which $g_\pi : \mathbb{R}^3 \to [0, 1]$, $f_{\pi,1} : \mathcal{S}_1 \to \mathbb{R}$, $f_{\pi,2} : \mathcal{S}_2 \to \mathbb{R}$, and $f_{\pi,3} : \mathcal{S}_3 \to \mathbb{R}$. To illustrate the flexibility of the representation in defining arbitrary initial probability vectors, we consider a general function $h_\pi : \mathcal{S}_1 \times \mathcal{S}_2 \times \mathcal{S}_3 \to [0, 1]$. To represent h_π, one possibility for defining g_π and f_π is $f_{\pi,i}(s_i) = s_i$ for $i = 1, 2, 3$ and $g_\pi(a_1, a_2, a_3) = h_\pi(a_1, a_2, a_3)$ if $a_1 = s_1, a_2 = s_2$, and $a_3 = s_3$ and 0 otherwise. As a typical example, consider a $\boldsymbol{\pi}^{\text{ini}}$ such that $\boldsymbol{\pi}^{\text{ini}}(s^0) = 1$ for a given initial state $s^0 \in \mathcal{S}$, and therefore, $\boldsymbol{\pi}^{\text{ini}}(s) = 0$ for all $s \neq s^0$. One possibility for defining g_π and f_π is the following: $f_{\pi,i}(s_i) = 1$ if $s_i = s_i^0$ and 0 otherwise, for $i = 1, 2, 3$, and $g_\pi(a_1, a_2, a_3) = \prod_{i=1}^{3} a_i$.

In the rest of this section, we first present two local equivalence relations $\approx_{lo}$ and $\approx_{le}$ on the state space of level 2 of an MD. We then prove that lumping level 2 of the MD with respect to each of those relations results in an MD that is lumped with respect to (global) equivalence relations $\approx_{go}$ or $\approx_{ge}$.

Definition 3. *Local equivalences $\approx_{lo}$ and $\approx_{le}$ (and corresponding partitions $\mathcal{P}_{lo}$ and $\mathcal{P}_{le}$) are defined on $\mathcal{S}_2$ as follows:*

$s_2 \approx_{lo} \hat{s}_2$ *if*

$$f_2(s_2) = f_2(\hat{s}_2) \text{ and} \tag{1}$$

$$\forall\, n_2 \in N_2, C_2 \in \mathcal{P}_{lo} : \mathbf{R}_{n_2}(s_2, C_2) = \mathbf{R}_{n_2}(\hat{s}_2, C_2) \tag{2}$$

$s_2 \approx_{le} \hat{s}_2$ *if*

$$f_{\pi,2}(s_2) = f_{\pi,2}(\hat{s}_2) \text{ and} \quad (3)$$

$$\forall\, n_2 \in N_2 : \mathbf{R}_{n_2}(s_2, \mathcal{S}_2) = \mathbf{R}_{n_2}(\hat{s}_2, \mathcal{S}_2) \text{ and} \quad (4)$$

$$\forall\, n_2 \in N_2, C_2 \in \mathcal{P}_{le} : \mathbf{R}_{n_2}(C_2, s_2) = \mathbf{R}_{n_2}(C_2, \hat{s}_2) \quad (5)$$

■

In $\approx_{lo}$ and $\approx_{le}$, the letters l, o, and e stand for local, ordinary, and exact, respectively. Note that the symbol "=" in Eqs. (2), (4), and (5) means equality of matrices (all elements equal), and that the matrices on both sides of each equality are at most of size $|\mathcal{S}_3|\times|\mathcal{S}_3|$. The following proposition states when the equalities in (2), (4), and (5) will hold in terms of matrix elements.

Proposition 1. *Eq. (2) holds iff* $\forall\, n_2 \in N_2, C_2 \in \mathcal{P}_{lo}$, $s_3, s_3' \in \mathcal{S}_3$*:*

$$\sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, s_2') \mathbf{R}_{n_3}(s_3, s_3') = \sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(\hat{s}_2, s_2') \mathbf{R}_{n_3}(s_3, s_3')$$

Eq. (4) holds iff $\forall\, n_2 \in N_2, s_3, s_3' \in \mathcal{S}_3$*:*

$$\sum_{s_2' \in \mathcal{S}_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, s_2') \mathbf{R}_{n_3}(s_3, s_3') = \sum_{s_2' \in \mathcal{S}_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(\hat{s}_2, s_2') \mathbf{R}_{n_3}(s_3, s_3')$$

And Eq. (5) holds iff $\forall\, n_2 \in N_2, C_2 \in \mathcal{P}_{le}, s_3, s_3' \in \mathcal{S}_3$*:*

$$\sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2', s_2) \mathbf{R}_{n_3}(s_3, s_3') = \sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2', \hat{s}_2) \mathbf{R}_{n_3}(s_3, s_3')$$

■

In Definition 4, we "extend" $\approx_{lo}$ and $\approx_{go}$ to define another pair of equivalence relations $\approx_{go}$ and $\approx_{ge}$ that imply a partition on $\mathcal{S}$, the state space of the overall MD.

Definition 4. *Global equivalences* $\approx_{go}$ *and* $\approx_{ge}$ *(and corresponding partitions* $\mathcal{P}_{go}$ *and* $\mathcal{P}_{ge}$*) are defined on* $\mathcal{S}$ *for states* $s = (s_1, s_2, s_3), \hat{s} = (\hat{s}_1, \hat{s}_2, \hat{s}_3) \in \mathcal{S}$ *as*
a) $s \approx_{go} \hat{s}$ *if* $s_1 = \hat{s}_1, s_2 \approx_{lo} \hat{s}_2,$ *and* $s_3 = \hat{s}_3$,
b) $s \approx_{ge} \hat{s}$ *if* $s_1 = \hat{s}_1, s_2 \approx_{le} \hat{s}_2,$ *and* $s_3 = \hat{s}_3$ ■

Notice that for each class $C \in \mathcal{P}_{go}$ (resp. $C \in \mathcal{P}_{ge}$) there is a corresponding class $C_2 \in \mathcal{P}_{lo}$ ($C_2 \in \mathcal{P}_{go}$) such that $C_2 = \{s_2 |\ (s_1, s_2, s_3) \in C\}$. On the other hand, given s_1 and s_3 and a class $C_2 \in \mathcal{P}_{lo}$ ($C_2 \in \mathcal{P}_{le}$), there is a corresponding class $C \in \mathcal{P}_{go}$ ($C \in \mathcal{P}_{ge}$) such that $C = \{(s_1, s_2, s_3) |\ s_2 \in C_2\}$.

In Theorems 3 and 4, we prove that $\approx_{go}$ and $\approx_{ge}$ equivalence relations satisfy the ordinary and exact lumpability conditions on an MD, respectively. That implies that the set of conditions on the matrices and rewards of one level of an MD given by $\approx_{lo}$ and extended to the complete MD by $\approx_{go}$ implies ordinary lumpability on the MD (similarly $\approx_{le}$ and $\approx_{ge}$ for exact lumpability).

Theorem 3. $(\mathcal{S}, \mathbf{Q}, \mathbf{r}, \boldsymbol{\pi}^{ini})$ *is ordinarily lumpable with respect to partition* $\mathcal{P}_{go}$ *(and its corresponding equivalence relation* $\approx_{go}$*).*

Proof. Based on Theorem 1, we need to prove:

$$\forall\, s \approx_{go} \hat{s} : \quad g(f_1(s_1), f_2(s_2), f_3(s_3)) = \quad (6)$$

$$g(f_1(\hat{s}_1), f_2(\hat{s}_2), f_3(\hat{s}_3)) \quad (7)$$

$$\text{and} \quad \forall\, C \in \mathcal{P} : \mathbf{R}(s, C) = \mathbf{R}(\hat{s}, C) \quad (8)$$

Consider two states $s = (s_1, s_2, s_3) \approx_{go} \hat{s} = (\hat{s}_1, \hat{s}_2, \hat{s}_3)$. By Definition 4, we have $s_1 = \hat{s}_1$ and $s_3 = \hat{s}_3$, and by Definition 3, we know that $f_2(s_2) = f_2(\hat{s}_2)$. Therefore, (6) holds. To prove (8), we transform the left-hand side of the equality to its right-hand side, as follows:

$$\mathbf{R}(s, C) = \sum_{s' \in C} \mathbf{R}(s, s') =$$

$$\sum_{s' \in C} \sum_{n_2 \in N_2} r_{1,n_2}(s_1, s_1') \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, s_2') \mathbf{R}_{n_3}(s_3, s_3')$$

(by Definition 4) $=$

$$\sum_{s_2' \in C_2} \sum_{n_2 \in N_2} r_{1,n_2}(s_1, s_1') \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, s_2') \mathbf{R}_{n_3}(s_3, s_3')$$

(moving the sums) $=$

$$\sum_{n_2 \in N_2} r_{1,n_2}(s_1, s_1') \underbrace{\sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, s_2') \mathbf{R}_{n_3}(s_3, s_3')}_{A}$$

($A = B$ by Prop. (1)) $=$

$$\sum_{n_2 \in N_2} r_{1,n_2}(s_1, s_1') \underbrace{\sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(\hat{s}_2, s_2') \mathbf{R}_{n_3}(s_3, s_3')}_{B}$$

(reverse argument) $= \sum_{s' \in C} \mathbf{R}(\hat{s}, s') = \mathbf{R}(\hat{s}, C)$ ■

Theorem 4. $(\mathcal{S}, \mathbf{Q}, \mathbf{r}, \boldsymbol{\pi}^{ini})$ *is exactly lumpable with respect to partition* $\mathcal{P}_{ge}$ *(and its corresponding equivalence relation* $\approx_{ge}$*).*

Proof. Based on Theorem 1, we need to prove:

$$\forall\, s \approx_{ge} \hat{s} : \quad g_\pi(f_{\pi,1}(s_1), f_{\pi,2}(s_2), f_{\pi,3}(s_3)) =$$

$$g_\pi(f_{\pi,1}(\hat{s}_1), f_{\pi,2}(\hat{s}_2), f_{\pi,3}(\hat{s}_3)) \quad (9)$$

$$\text{and} \quad \mathbf{R}(s, \mathcal{S}) = \mathbf{R}(\hat{s}, \mathcal{S}) \quad (10)$$

$$\text{and} \quad \forall\, C \in \mathcal{P} : \mathbf{R}(C, s) = \mathbf{R}(C, \hat{s}) \quad (11)$$

Equalities (9) and (11) can respectively be proved much like (6) and (8) in Theorem 3. To prove (10), we use the same approach that we used in the proof of Theorem 3, but replace C with $\mathcal{S}$. In particular,

$$\mathbf{R}(s,\mathcal{S}) = \sum_{s'\in\mathcal{S}} \mathbf{R}(s,s') = \sum_{(s'_1,s'_2,s'_3)\in\mathcal{S}} \sum_{n_2\in N_2} \left[r_{1,n_2}(s_1,s'_1) \sum_{n_3\in N_3} r_{n_2,n_3}(s_2,s'_2)\mathbf{R}_{n_3}(s_3,s'_3) \right]$$

$$\text{(moving the sums)} \quad = \sum_{s'_1\in\mathcal{S}_1} \sum_{n_2\in N_2} r_{1,n_2}(s_1,s'_1) \underbrace{\sum_{s'_3\in\mathcal{S}_3} \sum_{s'_2\in\mathcal{S}_2} \sum_{n_3\in N_3} r_{n_2,n_3}(s_2,s'_2)\mathbf{R}_{n_3}(s_3,s'_3)}_{A}$$

$$(A = B \text{ by Prop. (1)}) \quad = \sum_{s'_1\in\mathcal{S}_1} \sum_{n_2\in N_2} r_{1,n_2}(\hat{s}_1,s'_1) \underbrace{\sum_{s'_3\in\mathcal{S}_3} \sum_{s'_2\in\mathcal{S}_2} \sum_{n_3\in N_3} r_{n_2,n_3}(\hat{s}_2,s'_2)\, \mathbf{R}_{n_3}(\hat{s}_3,s'_3)}_{B}$$

$$\text{(reverse argument)} \quad = \sum_{s'\in\mathcal{S}} \mathbf{R}(\hat{s},s') = \mathbf{R}(\hat{s},\mathcal{S}) \quad \blacksquare$$

Theorems 3 and 4 ensure the lumpability of an MRP given a set of sufficient conditions on the lumpability of the matrices of a specific level of the corresponding MD. The next natural question is how to efficiently compute the local lumpable partition $\mathcal{P}_{lo}$ or $\mathcal{P}_{le}$ that satisfies the conditions in those theorems. That question is answered in the following section.

4 Compositional Lumping Algorithm

We now know, by Theorems 3 and 4, that given a relation $\approx_{lo}$ ($\approx_{le}$) on the states of one level, MRP M is ordinarily (exactly) lumpable with respect to $\approx_{go}$ ($\approx_{ge}$), which was derived from $\approx_{lo}$ ($\approx_{le}$). Therefore, the problem of lumping M is reduced to the computation of $\approx_{lo}$ ($\approx_{le}$). This section describes an algorithm to compute those relations. Given an MD with its associated rewards and initial probability distribution vectors, we run the algorithm for all levels of the MD, once for each level, to obtain a lumped MRP.

The algorithm we develop is based on the lumping algorithm given in [9]. In [9], an efficient algorithm based on the partition refinement approach is presented for computing the optimal (i.e., coarsest) equivalence relation that gives an ordinary lumping on a CTMC. In this paper, we extend that algorithm in two ways: (1) to support exact lumpability and (2) to support conditions given in Definition 3 instead of Definition 2, as in [9]. The two extensions give us the machinery we need to compute $\approx_{lo}$ and $\approx_{le}$.

Extension to Exact Lumpability. Figure 1 shows the pseudocode for the partition refinement-based lumping algorithm of [9], with the modifications necessary for our purpose. In particular, (1) without concerning ourselves with the implementation as in [9], we have rewritten the algorithm to make it easier to understand, especially when we make the above extensions; and (2) we support ordinary and exact lumpability by using the function K in the LUMP and COMPLUMPING procedures and by choosing the appropriate value for the initial partition $\mathcal{P}^{\text{ini}}$. The ways we choose the appropriate K and compute $\mathcal{P}^{\text{ini}}$ will be explained shortly. Detailed information regarding the correctness of LUMP and its running time analysis can be found in [9].

At the highest level, the LUMP procedure uses COMPLUMPING to compute the coarsest lumpable partition for the CTMC represented by state transition rate matrix $\mathbf{R}$ and state space $\mathcal{S}$ starting with initial partition $\mathcal{P}^{\text{ini}}$. It then builds, according to Theorem 2, the state transition rate matrix of the lumped CTMC in lines 3 and 4 for ordinary lumping and in lines 3′ and 4′ for exact lumping.

The main idea of COMPLUMPING is that a given initial partition $\mathcal{P}^{\text{ini}}$ is refined repeatedly (lines 3-6) until the lumpability conditions are satisfied. That is done by refining partition $\mathcal{P}$ (initialized to $\mathcal{P}^{\text{ini}}$ in line 1) with respect to a set of potential *splitters* L. For each splitter $C \in L$, $K(\mathbf{R}, s, C)$ is computed for each state $s \in \mathcal{S}$ and stored in vector element $\mathbf{sum}(s)$. Then, SPLIT partitions each class $C \in \mathcal{P}$ into subclasses according to vector $\mathbf{sum}$.

Function K is the key to generalizing this algorithm. Its arguments are respectively the state transition rate matrix, a state index, and a set of state indices. By choosing K appropriately, we can customize the algorithm to compute partitions that satisfy a set of desired conditions. The range of K can be any arbitrary "data type"[2] T on which equality testing is well-defined. Consequently, the elements of $\mathbf{sum}$ are of data type T too. By looking at Theorem 1, we realize that $K(\mathbf{R}, s, C)$ should be set to $\mathbf{R}(s, C)$ for ordinary lumping and to $\mathbf{R}(C, s)$ for exact lumping. Later on, we will see how choosing K helps us compute $\approx_{lo}$ and $\approx_{le}$ using COMPLUMPING.

At each iteration of the for loop of lines 2-7 of SPLIT, D, which is computed in ADDPAIR, gives the subclasses into which each class $C \in \mathcal{P}$ should be partitioned. Formally, D is a set of pairs such that
1. $\bigcup_{(sum,C')\in D} C' = C$,
2. $\forall\, (sum_1, C'_1) \neq (sum_2, C'_2) \in D : sum_1 \neq sum_2 \Rightarrow C'_1 \cap C'_2 = \emptyset$, and
3. $\forall\, (sum, C') \in D, i \in C' : C' \neq \emptyset$,
sum is of data type T, and $\mathbf{sum}[i] = sum$.

[2] We do not formally introduce data types here because we do not believe it would improve the readability of the paper. A concept of data types similar to that in the C language suffices for our discussion.

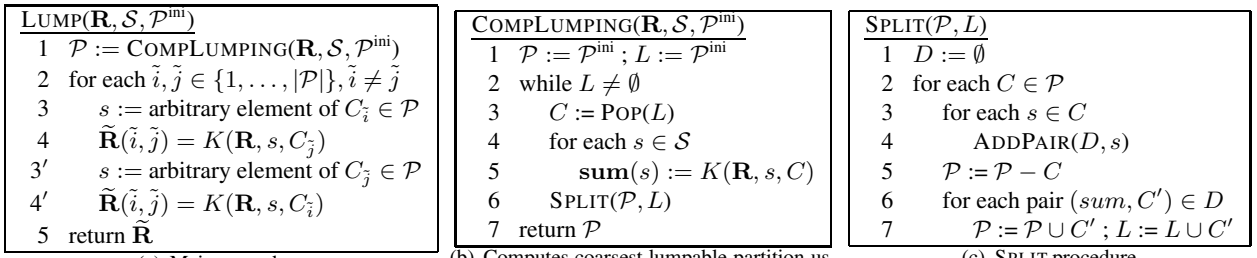

Figure 1. Algorithm for computing ordinarily and exactly lumpable partitions for a single matrix R

The definition of D requires equality to be well-defined on data type T, as we mentioned above. D is built in lines 3-4 of SPLIT. In lines 5-7, each class $C \in \mathcal{P}$ is replaced with subclasses taken from the second components of elements of D. Those subclasses are also added to the set of potential splitters L so that COMPLUMPING will work correctly. $\mathcal{P}^{\text{ini}}$ is computed with respect to ordinary or exact lumpability. For ordinary lumpability, two states i and j are in the same equivalence class of $\mathcal{P}^{\text{ini}}$ iff $\mathbf{r}(i) = \mathbf{r}(j)$ (Theorem 1(a)). For exact lumpability, two states i and j are in the same equivalence class of $\mathcal{P}^{\text{ini}}$ iff $\boldsymbol{\pi}^{\text{ini}}(i) = \boldsymbol{\pi}^{\text{ini}}(j)$ and $\mathbf{R}(i, \mathcal{S}) = \mathbf{R}(j, \mathcal{S})$ (Theorem 1(b)).

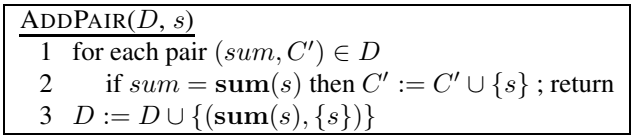

Figure 2. AddPair procedure

Extension to $\approx_{lo}$ and $\approx_{le}$. Computation of $\approx_{lo}$ and $\approx_{le}$ equivalence relations using the COMPLUMPING procedure requires the choice of an appropriate K function that takes Definition 3 into consideration. The first obvious way to make $K(\mathbf{R}_{n_2}, s_2, C_2)$ satisfy Eqs. (2) and (5) is to set $K(\mathbf{R}_{n_2}, s_2, C_2)$ to $\mathbf{R}_{n_2}(s_2, C_2)$ for ordinary lumpability and to $\mathbf{R}_{n_2}(C_2, s_2)$ for exact lumpability, when we apply COMPLUMPING to R_{n_2}.

Therefore, data type T is the set of matrices of size at most $|\mathcal{S}_3| \times |\mathcal{S}_3|$. Considering the fact that level 3 of the MD is built through merging of at most $L - 1$ levels of the original MD, the computation of function K and equality testing for T are prohibitively time-consuming for levels $i < L$. Therefore, we do not follow that approach.

The other choice is to have K compute a formal sum represented as a set of (coefficient, node index) pairs. Note that for any node R_{n_2},

$$\mathbf{R}_{n_2}(s_2, C_2) = \sum_{s_2' \in C_2} \sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, s_2') \cdot \mathbf{R}_{n_3} = \sum_{n_3 \in N_3} \underbrace{\left(\sum_{s_2' \in C_2} r_{n_2,n_3}(s_2, s_2') \right)}_{= r_{n_2,n_3}(s_2, C_2)} \cdot \mathbf{R}_{n_3}, \quad (12)$$

which implies $\mathbf{R}_{n_2}(s_2, C_2) = \mathbf{R}_{n_2}(\hat{s}_2, C_2)$ if $\forall\, n_3 \in N_3 : r_{n_2,n_3}(s_2, C_2) = r_{n_2,n_3}(\hat{s}_2, C_2)$. A similar formulation holds for $\mathbf{R}_{n_2}(C_2, s_2)$. The condition is only sufficient since (1) a weighted sum of matrices may be equal even if the individual terms differ, and (2) $\mathbf{R}_{n_i} = \mathbf{R}_{n_i'} \Leftrightarrow n_i = n_i'$ does not necessarily hold for an arbitrary MD. Canonical MDs [15] are a particular subclass of MDs in which the expression is true; $\mathbf{R}_{n_i}$ is uniquely represented by n_i. Nevertheless, efficiency of MDs is based on sharing of equal nodes, so one can expect that having two nodes R_{n_i} and $R_{n_i'}$ that represent the same matrix $\mathbf{R}_{n_i} = \mathbf{R}_{n_i'}$ is uncommon in MDs.

Based on the above observation, we can localize the comparison at level 2 and set $K(\mathbf{R}_{n_2}, s_2, C_2)$

$$= \begin{cases} \{(r_{n_2,n_3}(s_2, C_2), n_3) |\ n_3 \in N_3\} & \text{ordinary lumping} \\ \{(r_{n_2,n_3}(C_2, s_2), n_3) |\ n_3 \in N_3\} & \text{exact lumping} \end{cases}$$

which is a set representation of the formal sum $\sum_{n_3 \in N_3} r_{n_2,n_3}(s_2, C_2) \cdot R_{n_3}$ or $\sum_{n_3 \in N_3} r_{n_2,n_3}(C_2, s_2) \cdot R_{n_3}$ with references to nodes R_{n_3}, and not to matrices $\mathbf{R}_{n_3}$. Two formal sums are equal if their corresponding sets are equal. That means that the algorithm is applied locally only at nodes in N_2 of size $|\mathcal{S}_2| \times |\mathcal{S}_2|$ and not at matrices of size at most $|\mathcal{S}_3| \times |\mathcal{S}_3|$.

Using local lumpability conditions for a level that are only sufficient (instead of both sufficient and necessary) leads to an improved time complexity for the algorithm, but also prevents the algorithm from generating the coarsest possible lumpable partition for that level. That means there is a trade-off between time complexity and coarseness of the computed partition when the algorithm is used on an L-level MD. We have so far shown how to compute a partition

that satisfies Eqs. (1) and (4) of Definition 3 for one node of the matrix. The last step in computing $\mathcal{P}_{lo}$ and $\mathcal{P}_{le}$ is to find a partition that satisfies those equations for all nodes of level 2. COMPLUMPINGLEVEL in Figure 3(a) computes such a partition by fixed-point iteration. More specifically, it applies the COMPLUMPING algorithm repeatedly to all nodes in level i until they are all lumpable with respect to the computed partition $\mathcal{P}$.

```
COMPLUMPINGLEVEL(P^ini, i)
1  P := P^ini
2  repeat
3     P' := P
4     for each n_i ∈ N_i
5        P' := COMPLUMPING(R_{n_i}, S_i, P')
6  until P = P'
7  return P
```

(a) Computes lumpable partition for level i

```
COMPOSITIONALLUMP
1  for i := 1 to L
2     Compute P_i^ini
3     P_i = COMPLUMPINGLEVEL(P_i^ini, i)
4     for each n_i ∈ N_i
5        R̃_{n_i} := LUMP(R_{n_i}, S_i, P_i)
6        Replace R_{n_i} with R̃_{n_i} in MD
7     Compute lumped version of rewards
      and initial probabilities at level i
```

(b) Main algorithm

Figure 3. Compositional lumping algorithm for an MD

Overall Algorithm. The final step of our algorithm for lumping an MD is to use COMPLUMPINGLEVEL to compute lumpable partitions for each level starting from an initial partition (lines 2-3 of Figure3(b)), lump every node with respect to the partition corresponding to its level (line 5), replace each node with its lumped version (line 6), and finally, compute $\widetilde{\boldsymbol{\pi}}^{\text{ini}}$ and $\widetilde{\mathbf{r}}$ (line 7).

Line 2 computes the initial partition for level i based on whether we are computing ordinary lumping or exact lumping. In fact, $\mathcal{P}_i^{\text{ini}}$ is the coarsest partition on $\mathcal{S}_i$ such that $\forall\, C \in \mathcal{P}_i^{\text{ini}}, s_i, s_i' \in C, n_i \in N_i, n_{i+1} \in N_{i+1}$:

ordinary lumping: $f_i(s_i) = f_i(s_i')$

exact lumping: $f_{\pi,i}(s_i) = f_{\pi,i}(s_i')$ and $r_{n_i,n_{i+1}}(s_i, \mathcal{S}_i) = r_{n_i,n_{i+1}}(s_i', \mathcal{S}_i)$

The overall algorithm is implemented in the Möbius tool [7] and the implementation is integrated in the symbolic state-space generator (SSG) [10].

5 Example

We consider a tandem multi-processor system with load-balancing and failure and repair operations that consists of two subsystems: MSMQ and hypercube. Each subsystem has a number of servers, an input pool of jobs that are waiting to be serviced, and an output pool of jobs that have already been serviced by a server in the subsystem. Each subsystem takes jobs from its input pool, processes them using its servers and passes them to its output pool. Each of the two subsystems interacts with the other by sharing its output pool with the input pool of the other. In other words, jobs in one subsystem enter the input pool of the other upon completion of service. The system is closed in the sense that there is always a constant number J of jobs in it.

The first subsystem is an MSMQ (Multi-Server Multi-Queue) polling-based queuing system with 3 identical servers and 4 identical queues, as described in [14]. Fig. 4 shows a high-level view of the cyclic arrangement of the queues and also shows how each of the 3 servers moves from one queue to the next after a waiting time that is an exponentially distributed random variable with a constant rate. Upon entering a queue, a server polls the queue. If there is no job waiting for service, it goes to the next queue after some waiting time. Otherwise, it gives service to one job in the queue and waits to be transferred to the next one. The MSMQ subsystem distributes the jobs from its input pool to each of the 4 queues with equal probability. After being served, each job is transferred to the hypercube subsystem's input pool. More details on the MSMQ subsystem can be found in [14].

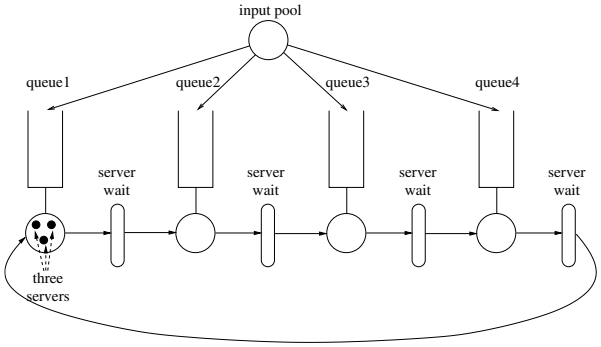

Figure 4. MSMQ subsystem

The hypercube subsystem consists of 8 cube-connected servers (Fig. 5). All the servers have the same behavior (given they are in the same state) except for two servers A and A'. A and A' have the same behavior as each other but are special (with respect to the other hypercube servers) in that they receive jobs from the input pool as described later. Servers in the hypercube can fail, and are repaired by a single repair facility that picks servers to repair uniformly from the pool of failed servers[3]. A failed server keeps its jobs in its queue unless they are transferred to a neighbor server by the load-balancing scheme described below. The subsys-

[3] Failure and repair behaviors of the subsystem are not shown in Fig. 5.

tem is considered unavailable when two or more servers are down.

Each server has a queue with capacity J. Jobs enter the servers' queues by a dispatcher that picks a job from the subsystem's input pool and assigns it to either server A or server A' with a probability distribution that favors the server that has fewer jobs in its queue. Another load-balancing scheme that governs the distribution of jobs among the servers is that whenever a server has > 1 more jobs than any of its neighbors, it sends one of its jobs to one of those neighbors, again assigning higher probability to servers with fewer jobs. That is how jobs are distributed among all servers. Finally, if a server fails, it transfers jobs in its queue one by one, with an exponentially distributed delay, to a neighbor server that is not down.

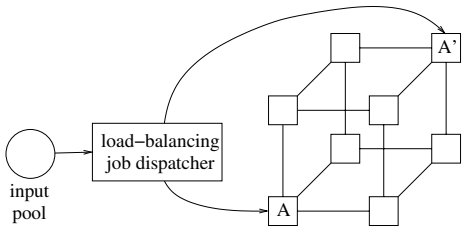

Figure 5. Hypercube subsystem

We used the Möbius tool [7] to model each of the subsystems using the stochastic activity network formalism [19]. Then, we composed the models by sharing their input and output pools via the Join operator in the Rep/Join composed model editor. Our implementation of the symbolic SSG [10] automatically partitions the set of places of the complete model and assigns each block of the partition to one level of the MD as follows: (1) common places of the two submodels, i.e., input and output pools, (2) places of the hypercube submodel minus those in level 1, and (3) places of the MSMQ submodel minus those in level 1.

Performance Results. All experiments were conducted using an Athlon XP2400 machine with 1.5 GB of main memory running Linux OS. The implementation was based on Möbius version 1.6.0 and was compiled with the gcc 3.3 compiler with the -O3 optimization option.

Table 1 shows information about the MD representation of the original (i.e., unlumped) and lumped CTMC of the tandem multi-processor system for different values of J. More specifically, the upper table shows the state-space size for each level and for the complete model and the number of MD nodes in each level. The middle table shows the lumped state-space size for each level and for the complete model and the state-space reduction we gain from the compositional lumping algorithm for each level and for the complete model. Notice that the number of nodes in each level of the lumped and unlumped MC is the same because the compositional lumping algorithm only replaces each MD node

J	unlumped SS sizes				# of MD nodes		
	overall	$\mathcal{S}_1$	$\mathcal{S}_2$	$\mathcal{S}_3$	N_1	N_2	N_3
1	22100	2	650	160	1	3	3
2	197600	3	3575	700	1	5	4
3	1236300	4	14300	2220	1	7	5

J	lumped SS sizes				reduction in SS		
	overall	$\widetilde{\mathcal{S}}_1$	$\widetilde{\mathcal{S}}_2$	$\widetilde{\mathcal{S}}_3$	overall	l_2	l_3
1	395	2	30	40	55.9	21.7	4
2	4075	3	178	175	48.4	20.4	4
3	28090	4	803	555	44	17.8	4

J	unlumped SS		lumped SS	
	gen time	MD space	lump time	MD space
1	0.05 s	53.9 KB	0.04 s	4.7 KB
2	0.80 s	421.0 KB	0.26 s	36.0 KB
3	12.10 s	2230.0 KB	1.80 s	201.0 KB

Table 1. Specifications of MD representation of tandem system's CTMC

with a possibly smaller one and does not create or delete any node. In the middle part of the table, l_2 and l_3 refer to the second and third levels of the MD. The state-space reduction for level 1 is not shown in the table because it is always 1. Finally, the lower table gives computation times in seconds for the symbolic state-space generation and lumping algorithm as well as memory use of the MDs of the unlumped and lumped MC in kilobytes.

We observe in Table 1 that the compositional lumping algorithm reduces the state-space size of the overall model to roughly $1/40$ to $1/50$ of its original value, and that is roughly equal to the product of the reductions in state space for all the levels. The equivalently behaving sets of servers, that is, (1) the three servers of the MSMQ subsystem, (2) servers A and A' in the hypercube, and (3) the other 6 servers in the hypercube, are the source of the lumpability found by our compositional lumping algorithm.

As mentioned before, our algorithm does not necessarily generate the smallest possible lumped CTMC (or its MD representation), because it is applied locally at each level of the MD and does not have a global view of the CTMC represented by the MD. In other words, the resulting lumped CTMC could possibly be lumped to a smaller CTMC by a state-level lumping algorithm that has a flat (i.e., global) view of the CTMC. Obviously, in the worst case, none of the levels of the MD satisfy the lumpability conditions for any non-trivial partition (partitions with more than one class), so that our lumping algorithm cannot reduce the size of the state space. For the given example, we verified that our compositional algorithm generates the smallest lumped CTMC possible. We did that by running the compositional

algorithm result through our implementation of the state-level lumping algorithm [9].

Generally, the reduction in state-space size has two major effects on the efficiency of iterative numerical solution algorithms that compute measures of CTMCs: it reduces both the space and time requirements for such algorithms. Reduction in the size of the state space affects space requirements in two ways. First, it makes the MD representation of the CTMC smaller. In our example, the memory requirement for the MD has been reduced by around an order of magnitude for all values of J. Second, and more importantly, it reduces the size of the solution vector; in our example, the vector was reduced to no more than $1/40$ of its original size. Therefore, the advantage of using our compositional lumping algorithm is that we can solve larger models than would be possible using only symbolic techniques; for our example, we solved models that are one or two orders of magnitude larger. Reduction in the state space also results in a roughly proportionate reduction in the amount of time spent for each iteration of the numerical solution algorithm. It is important to realize that all the benefits in terms of time and space requirements described above are achieved through an efficient algorithm in an amount of time that is negligible compared to the time needed for numerical analysis, and that is, for our example, considerably less than the time needed for state-space generation.

6 Conclusion

In this paper, we presented theoretical results, algorithms, implementation, and results from exercising an application example for the compositional lumping of CTMCs that are represented as an MD. The main point is that nodes of each level are reduced separately from those of other levels, i.e., based on local conditions. Unlike previous compositional lumping algorithms that were formalism-dependent, our algorithm is applicable on any MD, and thus, on any formalism that uses MDs for its CTMC representation. We consider ordinary as well as exact lumpability.

Acknowledgments We thank Jenny Applequist and several referees for their valuable remarks.

References

[1] A. Benoit, L. Brenner, P. Fernandes, and B. Plateau. Aggregation of stochastic automata networks with replicas. In *Linear Algebra and Its Applications*, 386:111–136, 2004.

[2] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *J. of App. Prob.*, 31:59–74, 1994.

[3] P. Buchholz. Equivalence relations for stochastic automata networks. In W. J. Stewart, editor, *Computation with Markov Chains*, Kluwer, 1995.

[4] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed colored nets and symmetric modeling applications. *IEEE Trans. on Computers*, 42(11):1343–1360, November 1993.

[5] G. Ciardo, R. Marmorstein, and R. Siminiceanu. Saturation unbound. In *Proc. of TACAS*, *LNCS* 2619, pages 379–393, Springer, 2003.

[6] G. Ciardo and A. Miner. A data structure for the efficient Kronecker solution of GSPNs. In *Proc. of PNPM*, pages 22–31, IEEE CS, 1999.

[7] D. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. G. Webster. The Möbius framework and its implementation. *IEEE Trans. on Soft. Eng.*, 28(10):956–969, October 2002.

[8] C. Delamare, Y. Gardan, and P. Moreaux. Performance evaluation with asynchronously decomposable SWN: implementation and case study. In *Proc. of PNPM*, pages 20–29, IEEE CS, 2003.

[9] S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal state-space lumping in Markov chains. *Inf. Proc. Letters*, 87(6):309–315, September 2003.

[10] S. Derisavi, P. Kemper, and W. H. Sanders. Symbolic state-space exploration and numerical analysis of state-sharing composed models. *Linear Algebra and Its Applications*, 386:137–166, July 2004.

[11] O. Gusak, T. Dayar, and J.-M. Fourneau. Lumpable continuous-time stochastic automata networks. *European J. of Operational Research*, 148:436–451, 2003.

[12] H. Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality*, *LNCS* 2428, Springer 2002.

[13] J. G. Kemeney and J. L. Snell. *Finite Markov Chains*. D. Van Nostrand Company, Inc., 1960.

[14] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling With Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[15] A. Miner. Efficient solution of GSPNs using canonical matrix diagams. In *Proc. of PNPM*, pages 101–110, IEEE CS, 2001.

[16] A. Miner and D. Parker. Symbolic representations and analysis of large probabilistic systems. In *Validation of Stochastic Systems: A Guide to Current Research*, *LNCS* 2925, Springer, 2004.

[17] B. Plateau and K. Atif. Stochastic automata network for modeling parallel systems. *IEEE Trans. in Soft. Eng.*, 17(10):1093–1108, October 1991.

[18] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE J. on Selected Areas in Communications*, 9(1):25–36, January 1991.

[19] W. H. Sanders and J. F. Meyer. Stochastic activity networks: Formal definitions and concepts. In *Lectures on Formal Methods and Performance Analysis*, *LNCS* 2090, Springer 2000.