Lumping-based equivalences in Markovian automata: algorithms and applications to product-form analyses

Giacomo Alzetta[b], Andrea Marin[a,*], Carla Piazza[b], Sabina Rossi[a]

[a] *Università Ca' Foscari Venezia, Italy*
[b] *Università di Udine, Italy*

Abstract

Markovian process algebras and stochastic automata are rigorous formalisms with well defined semantics that allow one to describe and verify both quantitative and qualitative properties of concurrent interacting systems. The analysis of such models is usually based on equivalence relations on the state space which are used to abstract from unwanted details and to identify those systems that exhibit the same behaviour for an external observer. In this paper we consider two relations over stochastic automata, named *lumpable bisimulation* and *exact equivalence*, that induce a strong and an exact lumping, respectively, on the underlying Markov chains. We show that an exact equivalence over the states of a non-synchronising automaton is indeed a lumpable bisimulation for the corresponding reversed automaton and then it induces a strong lumping on the time-reversed Markov chain underlying the model. This property allows us to prove that the class of quasi-reversible models is closed under exact equivalence. Quasi-reversibility is a pivotal property to study product-form models, i.e., models whose equilibrium distribution can be efficiently computed as the product of the equilibrium distribution of their sub-components opportunely parametrised. Hence, exact equivalence turns out to be a theoretical tool to prove the product-form of models by showing that they are exactly equivalent to models which are known to be quasi-reversible. Algorithms for computing both lumpable bisimulation and exact equivalence are introduced. A case study as well as performance tests are also presented.

Keywords: Stochastic Automata, Quantitative Analysis, Behavioural Equivalences, Product-forms Stochastic Models

*Corresponding author

Email addresses: alzetta.giacomo@spes.uniud.it (Giacomo Alzetta), marin@dais.unive.it (Andrea Marin), carla.piazza@uniud.it (Carla Piazza), srossi@dais.unive.it (Sabina Rossi)

1. Introduction

Stochastic models play a key role in reliability and performance analysis providing a sound framework for real improvements of software and hardware architectures, including telecommunication systems. Continuous Time Markov Chains (CTMCs) constitute the underlying semantics model of a plethora of modelling formalisms such as Stochastic Petri nets [24], Stochastic Automata Networks (SAN) [26], queueing networks [3] and a class of Markovian process algebras (MPAs), e.g., [15, 13]. The aim of these formalisms is to provide a high-level description language for complex real-time systems and automatic analysis techniques. Modularity in the model specification is an important feature of both MPAs and SANs that allows one to describe large systems in terms of interactions of simpler components. Nevertheless, one should notice that a modular specification does not lead to a modular analysis, in general. Thus, although the intrinsic compositional properties of such formalisms are extremely helpful in the specification of complex systems, in many cases carrying out an exact analysis for those models (e.g., those required by quantitative model checking) may be extremely expensive from a computational point of view.

The use of equivalence relations for quantitative models is an important formal approach that allows one to compare different systems as well as to improve the efficiency of some analysis [19, 11, 10]. To give an example, if we can prove that a model P is in some sense equivalent to Q and Q is much simpler than P, then we can carry out an analysis of the simplest component to derive the properties of the original one.

Bisimulation based relations on stochastic systems inducing the notions of ordinary (or strong) and exact lumpability for the underlying Markov chains have been studied in [6, 2, 15, 9, 28]. In this paper, we first recall the notions of *lumpable bisimulation* [16] and *exact equivalence* [8, 9] which have been both proved to be a congruence for Markovian process algebras and stochastic automata whose synchronisation semantics is defined as the master/slave synchronisation of the Stochastic Automata Networks (SAN) and comply with the ordinary and exact, respectively, lumping for Markov processes.

Interestingly, we show that an exact equivalence over a non-synchronising stochastic automaton is indeed a lumpable bisimulation on the reversed automaton (see [22] for a similar result in the context of Markov chains instead of stochastic automata) and then it induces a *strong lumping* on the time-reversed Markov chain underlying the model. This important property, allows us to prove that the class of quasi-reversible [17] stochastic networks is closed under exact equivalence. Quasi-reversibility is one of the most important and widely used characterisations of product-form models, i.e., models whose equilibrium distribution can be expressed as the product of functions depending only on the local state of each component. Informally, we can say that product-forms project the modularity in the model definition to the model analysis, thus drastically reducing the computational costs of the derivation of the quantitative indices. Basically, a composition of quasi-reversible components whose underlying chain is ergodic has a product-form solution, meaning that one can check the quasi-reversibility modularly for each component, without generating the whole state space.

In this paper we provide a new methodology to prove or disprove that a stochastic automaton is quasi-reversible: it is sufficient to show that a model is exactly equivalent to another one which is known to be (or to be not) quasi-reversible. In practice, this approach is useful because proving the quasi-reversibility of a model may be a hard task since it requires one to reverse the underlying CTMC and check some conditions on the reverse process, see, e.g., [17, 12]. Conversely, by using exact equivalence, one can prove or disprove the quasi-reversibility property by considering only the forward model, provided that it is exactly equivalent to another (simpler) quasi-reversible model known in the wide literature of product-forms. Moreover, while automatically proving quasi-reversibility is in general unfeasible, checking the exact equivalence between two automata can be done algoritmically by exploiting a partition refinement strategy, similar to that of Paige and Tarjan's algorithm for bisimulation [25].

We prove that both the notion of lumpable bisimulation and that of exact equivalence can be reduced to a labeled weighted compatibility problem and we generalize the algorithm for compatibility presented in [30] in order to deal with labels without increasing its computational complexity.

This paper is an extended version of the work published in [21]. We extended our previous work by presenting rigorous proofs for all the results stated in the paper and proposing an efficient algorithm for computing both lumpable bisimulations and exact equivalences. Moreover, we introduce a case study and show a set of performance tests for an implementation of the algorithms.

The paper is structured as follows. Section 2 introduces the notation and recalls the basic definitions on Markov chains. In Section 3 we give the definition of stochastic automata and specify their synchronisation semantics. Section 4 presents the definition of quasi-reversibility for stochastic automata. Lumpable bisimulation and exact equivalence are introduced in Section 5. In this section we prove that the class of quasi-reversible automata is closed under the exact equivalence relation. Algorithms for computing lumpable bisimulation and exact equivalence are presented in Section 6. In Section 7 we describe a case study and present some performance tests. Finally, Section 8 concludes the paper.

2. Markov chains, reversibility and lumpability

In this section we review the theoretical background on continuous-time Markov chains and the concepts of reversibility and lumpability.

2.1. Continuous-Time Markov Chains

A Continuous-Time Markov Chain (CTMC) is a stochastic process $X(t)$ for $t \in \mathbb{R}^+$ taking values into a descrete state space $\mathcal{S}$ such that (1) $X(t)$ is *stationary*, i.e., $(X(t_1), X(t_2), \ldots, X(t_n))$ has the same distribution as $(X(t_1 + \tau), X(t_2+\tau), \ldots, X(t_n+\tau))$ for all $t_1, t_2, \ldots, t_n, \tau \in \mathbb{R}^+$; (2) $X(t)$ has the *Markov* *property*, i.e., the conditional (on both past and present states) probability distribution of its future behaviour is independent of its past evolution until the present state:

$$Prob(X(t_{n+1}) = s_{n+1} \mid X(t_1) = s_1, X(t_2) = s_2, \ldots, X(t_n) = s_n) = \\ Prob(X(t_{n+1}) = s_{n+1} \mid X(t_n) = s_n).$$

A CTMC $X(t)$ is said to be *time-homogeneous* if the conditional probability $Prob(X(t+\tau) = s \mid X(t) = s')$ does not depend upon t, and is *irreducible* if every state in $\mathcal{S}$ can be reached from every other state. A state in a Markov process is called *recurrent* if the probability that the process will eventually return to the same state is one. A recurrent state is called *positive-recurrent* if the expected number of steps until the process returns to it is finite. A CTMC is *ergodic* if it is irreducible and all its states are positive-recurrent. In the case of finite Markov chains, irreducibility is sufficient for ergodicity.

An ergodic CTMC possesses an *equilibrium* (or *steady-state*) *distribution*, that is the *unique* collection of positive real numbers $\pi(s)$ with $s \in \mathcal{S}$ such that

$$\lim_{t\to\infty} Prob(X(t) = s \mid X(0) = s') = \pi(s)\,.$$

We denote by $q(s, s')$ the transition rate between two states s and s', with $s \neq s'$. The infinitesimal generator matrix $\mathbf{Q}$ of a CTMC $X(t)$ with state space $\mathcal{S}$ is the $|\mathcal{S}| \times |\mathcal{S}|$ matrix whose off-diagonal elements are the $q(s, s')$'s and whose diagonal elements are the negative sum of the extra diagonal elements of each row. Any non-trivial vector of real numbers $\boldsymbol{\mu}$ satisfying the system of global balance equations (GBEs)

$$\boldsymbol{\mu}\mathbf{Q} = \mathbf{0} \tag{1}$$

is called *invariant measure* of the CTMC. For an irreducible CTMC $X(t)$, if $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ are two invariant measures of $X(t)$, then there exists a constant $k > 0$ such that $\boldsymbol{\mu}_1 = k\boldsymbol{\mu}_2$. If the CTMC is ergodic, then there exists a unique invariant measure $\boldsymbol{\pi}$ whose components sum to unity, i.e., $\sum_{s\in\mathcal{S}} \pi(s) = 1$. In this case $\boldsymbol{\pi}$ is the *equilibrium* or *steady-state distribution* of the CTMC.

2.2. Reversibility

It is well-known that the solution of the system of global balance equations in (1) is often unfeasible when the CTMC underlying a real system model has a large number of states. However, the analysis of an ergodic CTMC in equilibrium can be greatly simplified if it satisfies the property that when the direction of time is reversed the stochastic behaviour of the process remains the same.

Given a stationary CTMC $X(t)$ with $t \in \mathbb{R}^+$, we say that $X(t)$ is *reversible* if it is stochastically identical to its reversed process, i.e., the process $(X(t_1), \ldots, X(t_n))$ has the same distribution as $(X(\tau - t_1), \ldots, X(\tau - t_n))$ for all $t_1, \ldots, t_n, \tau \in \mathbb{R}^+$ [17].

In the following we denote by $X^R(t)$ the reversed process of $X(t)$. It can be shown that if $X(t)$ is stationary then also $X^R(t)$ is stationary [17].

For a stationary Markov process there exists a necessary and sufficient condition for reversibility expressed in terms of the equilibrium distribution $\boldsymbol{\pi}$ and the transition rates (see [17]).

Proposition 2.1. *(Transition rates and probabilities of reversible processes)* A stationary CTMC with state space $\mathcal{S}$ and infinitesimal generator $\mathbf{Q}$ is reversible if there exists a vector of positive real numbers $\boldsymbol{\pi}$ summing to unity, such that for all $s, s' \in \mathcal{S}$ with $s \neq s'$,

$$\pi(s)q(s, s') = \pi(s')q(s', s)\,.$$

In this case $\boldsymbol{\pi}$ is the equilibrium distribution of the CTMC.

The *reversed process* $X^R(t)$ of a Markov process $X(t)$ can always be defined even when $X(t)$ is not reversible. In [17, 12] the authors show that $X^R(t)$ is a CTMC and its transition rates are defined according to the following proposition.

Proposition 2.2. *(Transition rates of reversed processes)* Given the stationary CTMC $X(t)$ with state space $\mathcal{S}$ and infinitesimal generator $\mathbf{Q}$, the transition rates of the reversed process $X^R(t)$, forming its infinitesimal generator $\mathbf{Q}^R$, are defined as follows: for all $s, s' \in \mathcal{S}$,

$$q^R(s', s) = \frac{\mu(s)}{\mu(s')} q(s, s')\,, \tag{2}$$

where $q^R(s', s)$ denotes the transition rate from s' to s in the reversed process and $\boldsymbol{\mu}$ is an invariant measure of $X(t)$.

The forward and the reversed processes share all the invariant measures and in particular they possess the same equilibrium distribution $\boldsymbol{\pi}$.

In the following, for a given CTMC with state space $\mathcal{S}$ and for any state $s \in \mathcal{S}$ we denote by $q(s)$ (resp., $q^R(s)$) the quantity $\sum_{s' \in \mathcal{S}, s \neq s'} q(s, s')$ (resp., $\sum_{s' \in \mathcal{S}, s \neq s'} q^R(s, s')$).

2.3. Lumpability

The notion of *lumpability* allows one to generate an aggregated Markov process that is smaller than the original one and then easier to solve. The concept of lumpability can be formalized in terms of equivalence relations over the state space of the Markov chain. Any such equivalence induces a *partition* on the state space of the Markov chain and aggregation is achieved by clustering equivalent states into macro-states, thus reducing the overall state space. In general, when a CTMC is aggregated the resulting stochastic process will not have the Markov property. However, if the partition can be shown to satisfy the so called *strong* lumpability condition [18, 1], then the Markov property is preserved and the equilibrium solution of the aggregated process may be used to derive an exact solution of the original one.

Strong lumpability has been introduced in [18] and further studied in [8, 29].

Definition 2.1. *(Strong lumpability)* Let $X(t)$ be a CTMC with state space $\mathcal{S}$ and $\sim$ be an equivalence relation over $\mathcal{S}$. We say that $X(t)$ is *strongly lumpable* with respect to $\sim$ (resp., $\sim$ is a *strong lumpability* for $X(t)$) if $\sim$ induces a partition on the state space of $X(t)$ such that for any equivalence class $S_i, S_j \in \mathcal{S}/\sim$ with $i \neq j$ and $s, s' \in S_i$,

$$\sum_{s'' \in S_j} q(s, s'') = \sum_{s'' \in S_j} q(s', s'') \,.$$

Thus, an equivalence relation over the state space of a Markov process is a strong lumpability if it induces a partition into equivalence classes such that for any two states within an equivalence class their aggregated transition rates to any other class are the same. Notice that every Markov process is strongly lumpable with respect to the identity relation, and also with respect to the trivial relation having only one equivalence class.

A probability distribution $\boldsymbol{\pi}$ is said to be *equiprobable* with respect to a partition of the state space $\mathcal{S}$ of an ergodic Markov process if for all the equivalence classes $S_i \in \mathcal{S}/\sim$ and for all $s, s' \in S_i$, $\pi(s) = \pi(s')$.

In [27] the notion of exact lumpability is introduced as a sufficient condition for a distribution to be equiprobable with respect to a partition.

Definition 2.2. *(Exact lumpability)* Let $X(t)$ be a CTMC with state space $\mathcal{S}$ and $\sim$ be an equivalence relation over $\mathcal{S}$. We say that $X(t)$ is *exactly lumpable* with respect to $\sim$ (resp., $\sim$ is an exact lumpability for $X(t)$) if $\sim$ induces a partition on the state space of $X(t)$ such that for any $S_i, S_j \in \mathcal{S}/\sim$ and $s, s' \in S_i$,

$$\sum_{s'' \in S_j} q(s'', s) = \sum_{s'' \in S_j} q(s'', s') \,.$$

An equivalence relation is an exact lumpability if it induces a partition on the state space such that for any two states within an equivalence class the aggregated transition rates into such states from any other class are the same.

Proposition 2.3. Let $X(t)$ be an ergodic CTMC with state space $\mathcal{S}$ and $\sim$ be an equivalence relation over $\mathcal{S}$. If $X(t)$ is *exactly lumpable* with respect to $\sim$ (resp., $\sim$ is an exact lumpability for $X(t)$) then for all $s, s' \in \mathcal{S}$ such that $s \sim s'$, $\mu(s) = \mu(s')$, where $\boldsymbol{\mu}$ is an invariant measure for $X(t)$.

3. Stochastic Automata

Many high-level specification languages for stochastic discrete-event systems are based on *Markovian process algebras* [15, 6, 14] that are characterized by powerful composition operators and timed actions whose delay is governed by independent random variables with a continuous-time exponential distribution. The expressivity of such languages allows the specification of both qualitative and quantitative properties in a single framework. In this paper we consider stochastic concurrent automata with an underlying continuous time Markov chain as common denominator of a wide set of Markovian stochastic process algebra. Stochastic automata are equipped with a *composition operation* by which a complex automaton can be constructed from simpler components. Our model draws a distinction between *active* and *passive* action types, and in forming the composition of automata only active/passive synchronisations are permitted. An analogue semantics is proposed for Stochastic Automata Networks in [26].

Definition 3.1. *(Stochastic Automaton (SA))* A stochastic automaton P is a tuple $(\mathcal{S}_P, \mathcal{T}_P, \rightsquigarrow_P, q_P)$ where

- $\mathcal{S}_P$ is a denumerable set of states called *state space* of P,
- $\mathcal{T}_P$ is a denumerable set of action types, partitioned into disjoint sets $\mathcal{A}_P$ of *active* types, $\mathcal{P}_P$ of *passive* types and the set $\{\tau\}$ of the *unknown* or *internal* type,
- $\rightsquigarrow_P \subseteq (\mathcal{S}_P \times \mathcal{S}_P \times \mathcal{T}_P)$ is a transition relation such that for all $s \in \mathcal{S}_P$, $(s, s, \tau) \notin \rightsquigarrow_P$,[1]
- q_P is a function from $\rightsquigarrow_P$ to $\mathbb{R}^+$ such that $\forall s_1 \in \mathcal{S}_P$ and $\forall a \in \mathcal{P}_P$, $\sum_{s_2:(s_1,s_2,a)\in\rightsquigarrow_P} q_P(s_1, s_2, a) \leq 1$.

We denote by $\rightarrow_P$ the relation containing all the tuples of the form (s_1, s_2, a, q) where $(s_1, s_2, a) \in\rightsquigarrow_P$ and $q = q_P(s_1, s_2, a)$. For $a \in \mathcal{A}_P \cup \{\tau\}$, we say that $q_P(s, s', a) \in \mathbb{R}^+$ is the *rate* of the transition from state s to s' with type a. Notice that this is indeed the apparent transition rate from s to s' relative to a. If a is passive then $q_P(s, s', a) \in (0, 1]$ denotes the *probability* that the automaton synchronises on type a with a transition from s to s'. In the following, we assume that $q_P(s, s', a) = 0$ whenever there are no transitions with type a from s to s'. If $s \in \mathcal{S}_P$, then for all $a \in \mathcal{T}_P$ we write $q_P(s, a) = \sum_{s'\in\mathcal{S}} q_P(s, s', a)$. Moreover we denote by $q_P(s, s') = \sum_{a\in\mathcal{T}_P} q_P(s, s', a)$ and $q_P(s) = \sum_{a\in\mathcal{T}_P} q_P(s, a)$. We say that P is *closed* if $\mathcal{P}_P = \emptyset$. We use the notation $s_1 \overset{a}{\rightsquigarrow}_P s_2$ to denote the tuple $(s_1, s_2, a) \in\rightsquigarrow_P$; we denote by $s_1 \xrightarrow{(a,r)}_P s_2$ (resp., $s_1 \xrightarrow{(a,p)}_P s_2$) the tuple $(s_1, s_2, a, r) \in\rightarrow_P$ (resp., $(s_1, s_2, a, p) \in\rightarrow_P$).

The CTMC underlying a closed stochastic automaton is defined as follows.

Definition 3.2. *(CTMC underlying a closed SA)* The CTMC underlying a closed stochastic automaton P, denoted $X_P(t)$, is defined as the CTMC with state space $\mathcal{S}_P$ and infinitesimal generator matrix $\mathbf{Q}$ defined as: for all $s_1 \neq s_2 \in \mathcal{S}_P$,

$$q(s_1, s_2) = \sum_{a,r:(s_1,s_2,a,r)\in\rightarrow_P} r\,.$$

[1] Notice that τ self-loops do not affect the equilibrium distribution of the CTMC underlying the automaton. Moreover, the choice of excluding τ self-loops will simplify the definition of automata synchronisation.

For ergodic chains, we denote an invariant measure and the equilibrium distribution of the CTMC underlying P by $\boldsymbol{\mu}_P$ and $\boldsymbol{\pi}_P$, respectively.

We say that an automaton is *irreducible* if for each pair of states there exists a sequence of transitions connecting them. We say that a closed automaton P is *ergodic* if its underlying CTMC is ergodic.

The synchronisation operator between two stochastic automata P and Q is defined in the style of the master/slave synchronisation of SANs [26] based on the Kronecker's algebra and the active/passive cooperation used in Markovian process algebra such as PEPA [15].

We define the synchronisation operator $\underset{L}{\otimes}$ that is an indexed family of operators, one for each possible set of action types L. Set L represents the action types on which the components must synchronise (the unknown action type, τ, may not appear in any cooperation set). We assume that each component proceeds independently with any transition whose type does not occur in L. However, any transition with action type in set L requires the simultaneous involvement of both components, one enabling the transition with active type and the other one enabling the same transition with passive type.

Definition 3.3. *(SA synchronisation)* Given two stochastic automata P and Q and a set of action types $L \subseteq \mathcal{T}_P \cup \mathcal{T}_Q$ we define the automaton $P \underset{L}{\otimes} Q = (\mathcal{S}_{P \underset{L}{\otimes} Q}, \mathcal{T}_{P \underset{L}{\otimes} Q}, \rightsquigarrow_{P \underset{L}{\otimes} Q}, q_{P \underset{L}{\otimes} Q})$ as follows:

- $\mathcal{S}_{P \underset{L}{\otimes} Q} = \mathcal{S}_P \times \mathcal{S}_Q$,
- $\mathcal{A}_{P \underset{L}{\otimes} Q} = \mathcal{A}_P \cup \mathcal{A}_Q$ and $\mathcal{P}_{P \underset{L}{\otimes} Q} = (\mathcal{P}_P \cup \mathcal{P}_Q) \setminus L$,
- $\rightsquigarrow_{P \underset{L}{\otimes} Q}$ and $q_{P \underset{L}{\otimes} Q}$ are defined according to the rules for $\rightarrow_{P \underset{L}{\otimes} Q}$ depicted in Table 1: indeed, the relation $\rightarrow_{P \underset{L}{\otimes} Q}$ contains all the tuples $((s_{p_1}, s_{q_1}),(s_{p_1}, s_{q_2}), a, q)$ such that $((s_{p_1}, s_{q_1}),(s_{p_1}, s_{q_2}), a) \in \rightsquigarrow_{P \underset{L}{\otimes} Q}$ and $q = q_{P \underset{L}{\otimes} Q}((s_{p_1}, s_{q_1}), (s_{p_1}, s_{q_2}), a)$.

Given a closed stochastic automaton P we can define its reversed automaton P^R in the style of [4], that is a stochastic automaton whose underlying CTMC $X_{P^R}(t)$ is identical to $X_P^R(t)$.

Definition 3.4. *(Reversed SA [4])* Let P be a closed stochastic automaton with an underlying irreducible CTMC and let $\boldsymbol{\mu}_P$ be an invariant measure. Then we define the stochastic automaton $P^R = (\mathcal{S}_{P^R}, \mathcal{T}_{P^R}, \rightsquigarrow_{P^R}, q_{P^R})$ reversed of P as follows:

- $\mathcal{S}_{P^R} = \{s^R \mid s \in \mathcal{S}_P\}$,
- $\mathcal{T}_{P^R} = \mathcal{A}_{P^R} \cup \{\tau\}$ with $\mathcal{A}_{P^R} = \mathcal{A}_P$ and $\mathcal{P}_{P^R} = \mathcal{P}_P = \emptyset$,
- $\rightsquigarrow_{P^R} = \{(s_1^R, s_2^R, a) : (s_2, s_1, a) \in \rightsquigarrow_P, a \in \mathcal{A}_P \cup \{\tau\}\}$,

$$\frac{s_{p_1} \xrightarrow{(a,q)}_P s_{p_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,q)}_{P \underset{L}{\otimes} Q} (s_{p_2}, s_{q_1})} \quad (a \notin L)$$

$$\frac{s_{q_1} \xrightarrow{(a,q)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,q)}_{P \underset{L}{\otimes} Q} (s_{p_1}, s_{q_2})} \quad (a \notin L)$$

$$\frac{s_{p_1} \xrightarrow{(a,r)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,p)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,pr)}_{P \underset{L}{\otimes} Q} (s_{p_2}, s_{q_2})} \quad (a \in \mathcal{A}_P \cap L,\ a \in \mathcal{P}_Q \cap L)$$

$$\frac{s_{p_1} \xrightarrow{(a,p)}_P s_{p_2} \quad s_{q_1} \xrightarrow{(a,r)}_Q s_{q_2}}{(s_{p_1}, s_{q_1}) \xrightarrow{(a,pr)}_{P \underset{L}{\otimes} Q} (s_{p_2}, s_{q_2})} \quad (a \in \mathcal{P}_P \cap L,\ a \in \mathcal{A}_Q \cap L)$$

Table 1: Operational rules for SA synchronisation

- $q_{P^R}(s_1^R, s_2^R, a) = \mu_P(s_2)/\mu_P(s_1) q_P(s_2, s_1, a)$.

It can be easily proved that for any invariant measure (including the equilibrium distribution) $\boldsymbol{\mu}_P$ for P there exists an invariant measure $\boldsymbol{\mu}_{P^R}$ for P^R such that for all $s \in \mathcal{S}_P$ it holds $\mu_P(s) = \mu_{P^R}(s^R)$, and viceversa.

4. Quasi-Reversible Automata

In this section we review the definition of quasi-reversibility given by Kelly in [17] by using the notation of stochastic automata. We first introduce a closure operation over stochastic automata that allows us to assign to all the transitions with the same passive type the same rate λ.

Definition 4.1. *(SA closure)* The closure of a stochastic automaton P with respect to a passive type $a \in \mathcal{P}_P$ and a rate $\lambda \in \mathbb{R}^+$, written $P^c = P\{a \leftarrow \lambda\}$, is the automaton defined as follows:

- $\mathcal{S}_{P^c} = \{s^c \mid s \in \mathcal{S}_P\}$,
- $\mathcal{T}_{P^c} = \mathcal{A}_{P^c} \cup \mathcal{P}_{P^c} \cup \{\tau\}$ with $\mathcal{A}_{P^c} = \mathcal{A}_P$ and $\mathcal{P}_{P^c} = \mathcal{P}_P \setminus \{a\}$,
- $\rightsquigarrow_{P^c} = \{(s_1^c, s_2^c, b) | \ (s_1, s_2, b) \in \rightsquigarrow_P,\ a \neq b\} \cup \{(s_1^c, s_2^c, \tau) | \ (s_1, s_2, a) \in \rightsquigarrow_P\}$,
- q_{P^c} is defined as:

$$q_{P^c}(s_1^c, s_2^c, b) = \begin{cases} q_P(s_1, s_2, b) & \text{if } b \neq a, \tau \\ q_P(s_1, s_2, a)\lambda + q_P(s_1, s_2, \tau) & \text{if } b = \tau \end{cases}$$

where we assume that $q_P(s_1, s_2, b) = 0$ if $(s_1, s_2, b) \notin \leadsto_P$.

Notice that for a closure P^c of a stochastic automaton P with respect to all its passive types in $\mathcal{P}_P$ we can compute the equilibrium distribution, provided that the underlying CTMC is ergodic (see Definition 3.2).

The notion of quasi-reversibility can be formalized as follows [17, 23].

Definition 4.2. *(Quasi-reversible SA)* An irreducible stochastic automaton P with $\mathcal{P}_P = \{a_1, \ldots, a_n\}$ and $\mathcal{A}_P = \{b_1, \ldots b_m\}$ is quasi-reversible if

- for all $a \in \mathcal{P}_P$ and for all $s \in \mathcal{S}_P$, $\sum_{s' \in \mathcal{S}_P} q_P(s, s', a) = 1$,
- for each closure $P^c = P\{a_1 \leftarrow \lambda_1\} \ldots \{a_n \leftarrow \lambda_n\}$ with $\lambda_1, \ldots, \lambda_n \in \mathbb{R}^+$ there exists a set of positive real numbers $\{k_{b_1}, \ldots, k_{b_m}\}$ such that for each $s \in \mathcal{S}_{P^c}$ and $1 \leq i \leq m$
$$k_{b_i} = \frac{\sum_{s' \in \mathcal{S}_{P^c}} \mu_{P^c}(s') q_{P^c}(s', s, b_i)}{\mu_{P^c}(s)}, \tag{3}$$
where $\boldsymbol{\mu}_{P^c}$ denotes any non-trivial invariant measure of the CTMC underlying P^c.

Notice that in the definition of quasi-reversibility we do not require the closure of P with respect to all its passive types to originate a stochastic automaton with an ergodic underlying CTMC because we assume $\boldsymbol{\mu}_{P^c}$ to be an invariant measure, i.e., we do not require that $\sum_{s \in \mathcal{S}_{P^c}} \mu_{P^c}(s) = 1$. However, the irreducibility of the CTMC underlying the automaton ensures that all the invariant measures differ for a multiplicative constant, hence Equation (3) is independent of the choice of the invariant measure.

Theorem 4.1 states that a network of quasi-reversible stochastic automata exhibits a product-form invariant measure and, if the joint state space is ergodic, a product-form equilibrium distribution. For the sake of simplicity, we state the theorem for two synchronising stochastic automata although the result holds for any finite set of automata which synchronise pairwise [17, 12, 23]. In the theorem below we assume that the synchronisation set contains all the passive types of the enabled activities. This property ensures that the automaton obtained by the composition does not have passive types and hence it is closed and has an underlying CTMC.

Theorem 4.1. *(Product-form solution based on quasi-reversibility)* Let P and Q be two quasi-reversible automata and L be a set of action types such that $\mathcal{P}_P \cup \mathcal{P}_Q \subseteq L$. Assume that $\mathcal{P}_P \subseteq \mathcal{A}_Q$, $\mathcal{P}_Q \subseteq \mathcal{A}_P$, and there exists a set of positive real numbers $\{k_a : a \in \mathcal{A}_P \cup \mathcal{A}_Q\}$ such that if we define the following automata $P^c = P\{a \leftarrow k_a\}$ for each $a \in \mathcal{P}_P$ and $Q^c = Q\{a \leftarrow k_a\}$ for each

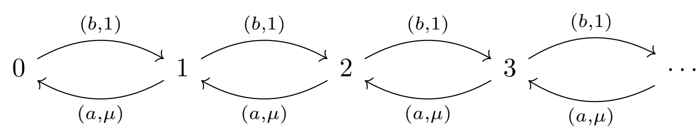

Figure 1: Stochastic automaton underlying a Jackson's queue.

$a \in \mathcal{P}_Q$ it holds:

$$
\begin{aligned}
k_a &= \frac{\sum_{s' \in \mathcal{S}_{P^c}} \mu_{P^c}(s') q_{P^c}(s', s, a)}{\mu_{P'}(s)} && \forall s \in \mathcal{S}_{P^c}, a \in \mathcal{A}_P \cap L \\
k_a &= \frac{\sum_{s' \in \mathcal{S}_{Q^c}} \mu_{Q'}(s') q_{Q^c}(s', s, a)}{\mu_{Q^c}(s)} && \forall s \in \mathcal{S}_{Q^c}, a \in \mathcal{A}_Q \cap L\,.
\end{aligned}
$$

Then, given the invariant measures $\boldsymbol{\mu}_{P^c}$ and $\boldsymbol{\mu}_{Q^c}$ it holds that

$$\mu_{P \underset{L}{\otimes} Q}(s_1, s_2) = \mu_{P^c}(s_1^c)\mu_{Q^c}(s_2^c)$$

is an invariant measure for all the positive-recurrent states $(s_1, s_2) \in \mathcal{S}_{P \underset{L}{\otimes} Q}$ where s_1^c and s_2^c are the states in $\mathcal{S}_{P^c}$ and $\mathcal{S}_{Q^c}$ corresponding to $s_1 \in \mathcal{S}_P$ and $s_2 \in \mathcal{S}_Q$ according to Definition 4.1. In this case we say that P and Q have a quasi-reversibility based product-form.

Example 4.1. *(Product-form solution of Jackson networks)* Jackson networks provide an example of models having a product-form solution. A network consists of a collection of exponential queues with state-independent probabilistic routing. Jobs arrive from the outside at each queuing station in the network according to a homogeneous Poisson process. It is well-known that the queues of Jackson networks are quasi-reversible and hence the product-form is a consequence of Theorem 4.1. Figure 1 shows the automaton underlying a Jackson's queue where a is an active type while b is a passive one. It is worth of notice that also the queues considered in [5, 20] are quasi-reversible. □

5. Lumpable Bisimulation and Exact Equivalence

In this section we introduce two coinductive definitions, named *lumpable bisimulation* and *exact equivalence*, over stochastic automata which provide a sufficient condition for strong and exact lumpability of the underlying CTMCs.

The definitions of lumpable bisimulation and exact equivalence as well as the algorithms presented in Section 6 refer to the entire class of stochastic automata, while the results we present in this section hold only for irreducible stochastic automata.

Lumpable bisimulation is developed in the style of Larsen and Skou's bisimulation [19] where transition rates are used analogously to probabilities in the probabilistic process algebra. We recall here the definition presented in [16].

Definition 5.1. *(Lumpable bisimulation)* Let P be a stochastic automaton. An equivalence relation $\mathcal{R} \subseteq \mathcal{S}_P \times \mathcal{S}_P$ is a *lumpable bisimulation* if whenever $(s, s') \in \mathcal{R}$ then for all $a \in \mathcal{T}_P$ and for all $C \in \mathcal{S}_P/\mathcal{R}$ such that

- either $a \neq \tau$,
- or $a = \tau$ and $s, s' \notin C$,

it holds $\sum_{s'' \in C} q(s, s'', a) = \sum_{s'' \in C} q(s', s'', a)$.

It is clear that the identity relation is a lumpable bisimulation. In [16] we proved that the transitive closure of a union of lumpable bisimulations is still a lumpable bisimulation. Hence, the maximal lumpable bisimulation, denoted $\sim_s$, is defined as the union of all the lumpable bisimulations and it is an equivalence relation. For any stochastic automaton P, $\sim_s$ induces a partition on the state space of the underlying Markov process that is a strong lumping (see Definition 2.1).

In order to extend the lumpable bisimulations to pairs of stochastic automata we need to consider the following definition of union between two stochastic automata.

Definition 5.2. *(Union Automaton)* Let P and Q be two stochastic automata. The *union* of P and Q is the stochastic automaton $P \cup Q$ defined as follows:

- $\mathcal{S}_{P\cup Q} = \mathcal{S}_P \uplus \mathcal{S}_Q$ is the disjoint union of the state spaces,
- $\mathcal{A}_{P\cup Q} = \mathcal{A}_P \cup \mathcal{A}_Q$ is the union of the active types,
- $\mathcal{P}_{P\cup Q} = \mathcal{P}_P \cup \mathcal{P}_Q$ is the union of the passive types,
- $s_1 \xrightarrow{(a,q)}_{P\cup Q} s_2$ if and only if either $s_1 \xrightarrow{(a,q)}_P s_2$ or $s_1 \xrightarrow{(a,q)}_Q s_2$.

Given two stochastic automata P and Q and two states $s_p \in \mathcal{S}_P$ and $s_q \in \mathcal{S}_Q$ with a slight abuse of notation we write $s_p \sim_s s_q$ to denote that s_p and s_q are lumpable bisimilar in the union automaton $P \cup Q$. Moreover, we say that P and Q are *equivalent according to the lumpable bisimulation relation*, denoted $P \sim_s Q$, if there exists $s_p \in \mathcal{S}_P$ and $s_q \in \mathcal{S}_Q$ such that $s_p \sim_s s_q$.

We now introduce the notion of exact equivalence for stochastic automata. This equivalence, mentioned as exact performance equivalence, has been introduced in [8] and studied in [9]. An equivalence relation over $\mathcal{S}_P$ is an *exact equivalence* if for any action type $a \in \mathcal{T}_P$, the total conditional transition rates from two equivalence classes to two equivalent states, via activities of this type, are the same. Moreover, for any type a, equivalent states have the same apparent conditional exit rate.

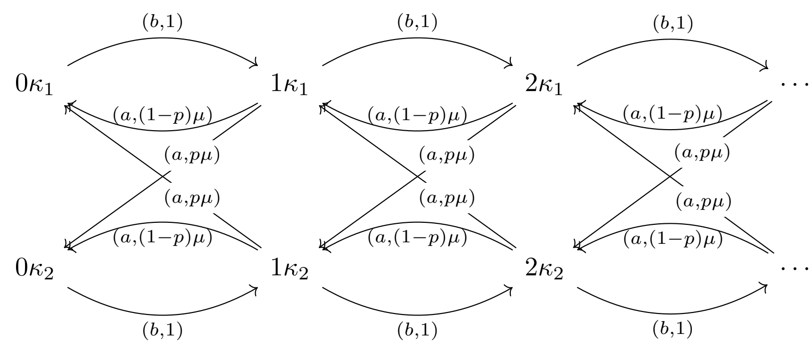

Figure 2: Queue with alternating servers.

Definition 5.3. *(Exact equivalence)* Let P be a stochastic automaton. An equivalence relation $\mathcal{R} \subseteq \mathcal{S}_P \times \mathcal{S}_P$ is an *exact equivalence* if whenever $(s, s') \in \mathcal{R}$ then for all $a \in \mathcal{T}_P$ and for all $C \in \mathcal{S}_P/\mathcal{R}$ it holds

- $q(s, a) = q(s', a)$,
- $\sum_{s'' \in C} q(s'', s, a) = \sum_{s'' \in C} q(s'', s', a)$.

The transitive closure of a union of exact equivalences is still an exact equivalence. Hence, the maximal exact equivalence, denoted $\sim_e$, is defined as the union of all exact equivalences and it is an equivalence relation.

Given two stochastic automata P and Q and two states $s_p \in \mathcal{S}_P$ and $s_q \in \mathcal{S}_Q$ with a slight abuse of notation we write $s_p \sim_e s_q$ to denote that s_p and s_q are exactly equivalent in the union automaton $P \cup Q$. Moreover, we say that P and Q are *exactly equivalent*, denoted $P \sim_e Q$, if there exists $s_p \in \mathcal{S}_P$ and $s_q \in \mathcal{S}_Q$ such that $s_p \sim_e s_q$.

Example 5.1. Let us consider a queueing model of a system with two identical processors, named κ_1 and κ_2. Each job is assigned to one of the processors which are assumed not to work in parallel. At each service completion event of processor κ_i, the next job is assigned to κ_j, for $i \neq j$, with probability p, and is assigned to processor κ_i with probability $1 - p$. Automaton P underlying this model is depicted in Figure 2 where state $n\kappa_i$, for $n > 0$ and $i = 1, 2$, denotes the state in which processor κ_i is being used and there are n customers waiting to be served. State $0\kappa_i$ denotes the empty queue. It is easy to prove that the equivalence relation $\sim$ obtained by the reflexive closure of $\{(n\kappa_1, n\kappa_2), (n\kappa_2, n\kappa_1) : n \in \mathbb{N}\}$ is an exact equivalence over the state space of P. Let us consider the automaton Q depicted in Figure 1, then it holds that the equivalence relation given by the symmetric and reflexive closure of $\sim' = \sim \cup \{(n\kappa_1, n), (n, n\kappa_2) : n \in \mathbb{N}\}$, where each n denotes a state of Q, is still an exact equivalence. $\square$

In [9] the author prove that, for any stochastic automaton P, $\sim_e$ induces an exactly lumpable partition on the state space of the Markov process underlying P.

Proposition 5.1. *(Exact lumpability)* Let P be a closed, irreducible, stochastic automaton with state space $\mathcal{S}_P$ and $X_P(t)$ its underlying Markov chain with infinitesimal generator matrix $\mathbf{Q}$. Then for any equivalence class $S_i, S_j \in \mathcal{S}_P/\sim_e$ and $s, s' \in S_i$,

$$\sum_{s'' \in S_j} q(s'', s) = \sum_{s'' \in S_j} q(s'', s')$$

i.e., $\sim_e$ is an exact lumpability for $X_P(t)$.

The next theorem plays an important role in studying the product-form of exactly equivalent automata. Informally, it states that the exact equivalence preserves the invariant measure of equivalent states. The proof follows from the results presented in [9].

Theorem 5.1. Let P and Q be two closed, irreducible, stochastic automata and $\boldsymbol{\mu}_P$ and $\boldsymbol{\mu}_Q$ be two invariant measures of P and Q, respectively. Then, there exists a positive constant K such that for each $s_1 \in \mathcal{S}_P$ and $s_2 \in \mathcal{S}_Q$ with $s_1 \sim_e s_2$ it holds that $\mu_P(s_1)/\mu_Q(s_2) = K$.

Corollary 5.1. Let P and Q be two closed, irreducible, stochastic automata and $\boldsymbol{\pi}_P$ and $\boldsymbol{\pi}_Q$ be the stationary distributions of P and Q, respectively. Then, for all $s_1, s_2 \in \mathcal{S}_P$ and $s'_1, s'_2 \in \mathcal{S}_Q$ such that $s_i \sim_e s'_i$ for $i = 1, 2$, it holds that $\pi_P(s_1)/\pi_P(s_2) = \pi_Q(s'_1)/\pi_Q(s'_2)$.

Finally, the next proposition states that both lumpable bisimulation and exact equivalence are congruences for SA synchronisation.

Proposition 5.2. *(Congruence)* Let P, P', Q, Q' be irreducible stochastic automata.

- If $P \sim_s P'$ and $Q \sim_s Q'$ then $P \underset{L}{\otimes} Q \sim_s P' \underset{L}{\otimes} Q'$ for any set of action types L.
- If $P \sim_e P'$ and $Q \sim_e Q'$ then $P \underset{L}{\otimes} Q \sim_e P' \underset{L}{\otimes} Q$ for any set of action types L.

Proof. The proof that $\sim_s$ is a congruence for SA synchronization is similar to the one in [15, 16].

The proof that $\sim_e$ is a congruence for SA synchronization can be derived from the results presented in [9]. $\square$

The next theorem provides a crucial result for our contributions. It states that any exact equivalence between two stochastic automata induces a lumpabale bisimulation between the corresponding reversed automata.

Theorem 5.2. *(Exact equivalence and lumpable bisimulation)* Let P and Q be two closed, irreducible, stochastic automata, P^R and Q^R be the corresponding reversed automata defined according to Definition 3.4 and $\sim$ be an exact equivalence over $P \cup Q$. Then $\sim' = \{(s_1^R, s_2^R) \in (\mathcal{S}_{P^R} \uplus \mathcal{S}_{Q^R}) \times (\mathcal{S}_{P^R} \uplus \mathcal{S}_{Q^R}) | \, (s_1, s_2) \in \sim\}$ is a lumpable bisimulation over $P^R \cup Q^R$.

PROOF. Let $\sim\subseteq (\mathcal{S}_P \uplus \mathcal{S}_Q) \times (\mathcal{S}_P \uplus \mathcal{S}_Q)$ be an exact equivalence, $s_1 \in \mathcal{S}_P$ and $s_2 \in \mathcal{S}_Q$ such that $s_1 \sim s_2$. We prove that $\sim' = \{(s_1^R, s_2^R) \in (\mathcal{S}_{P^R} \uplus \mathcal{S}_{Q^R}) \times (\mathcal{S}_{P^R} \uplus \mathcal{S}_{Q^R}) | \; (s_1, s_2) \in \sim\}$ is a lumpable bisimulation. For all $C \in (\mathcal{S}_P \uplus \mathcal{S}_Q)/\sim$, let $C^R = \{(s_1^R, s_2^R) | \; (s_1, s_2) \in C\} \in (\mathcal{S}_{P^R} \uplus \mathcal{S}_{Q^R})/\sim'$. We prove that for all $a \in \mathcal{T}_P \cup \mathcal{T}_Q$ and for all $C^R \in (\mathcal{S}_{P^R} \uplus \mathcal{S}_{Q^R})/\sim'$,

$$\sum_{s^R \in C^R} q_{P^R}(s_1^R, s^R, a) = \sum_{s^R \in C^R} q_{P^R}(s_2^R, s^R, a).$$

By Definition 3.4,

$$q_{P^R}(s_1^R, s^R, a) = \frac{\mu_P(s)}{\mu_P(s_1)} q_P(s, s_1, a).$$

From the fact that for all $s_1, s_2 \in C$, $\mu_P(s_1) = \mu_P(s_2)$ we have

$$\sum_{s^R \in C^R} q_{P^R}(s_1^R, s^R, a) = \frac{\mu_P(s)}{\mu_P(s_1)} \sum_{s \in C} q_P(s, s_1, a) = \frac{\mu_P(s)}{\mu_P(s_2)} \sum_{s \in C} q_P(s, s_1, a).$$

Finally, from the fact that $\sim$ is an exact equivalence and $s_1 \sim s_2$ we have

$$\sum_{s \in C} q_P(s, s_1, a) = \sum_{s \in C} q_P(s, s_2, a).$$

Hence,

$$\sum_{s^R \in C^R} q_{P^R}(s_1^R, s^R, a) = \sum_{s \in C} \frac{\mu_P(s)}{\mu_P(s_2)} q_P(s, s_2, a) = \sum_{s^R \in C^R} q_{P^R}(s_2^R, s^R, a).$$

□

As a consequence any exact equivalence over the state space of a stochastic automaton P induces a lumpable bisimulation over the state space of the reversed automaton P^R.

Corollary 5.2. Let P be a closed, irreducible, stochastic automaton and $\sim\subseteq \mathcal{S}_P \times \mathcal{S}_P$ be an exact equivalence. Then the relation $\sim' = \{(s_1^R, s_2^R) \in \mathcal{S}_{P^R} \times \mathcal{S}_{P^R} | \; (s_1, s_2) \in \sim\}$ is a lumpable bisimulation.

The following lemma provides a characterization of quasi-reversibility in terms of lumpable bisimulation. Informally, it states that an automaton is quasi-reversible if and only if for each closure its reversed is lumpable bisimilar to an automaton with a single state.

Lemma 5.3. *(Quasi-reversibility and lumpable bisimulation)* An irreducible stochastic automaton P is quasi-reversible if and only if the following properties hold for every closure P^c of P with reversed automaton P^{cR}:

- if $s^R \in \mathcal{S}_{P^{cR}}$, then $[s^R]_{\sim_s} = \mathcal{S}_{P^{cR}}$,

- if $a \in \mathcal{P}_P$ then $q_P(s,a) = 1$ for all $s \in \mathcal{S}_P$.

PROOF. ($\Rightarrow$) Let P be quasi-reversible, $\mathcal{P}_P = \{a_1, \dots, a_n\}$ and $\mathcal{A}_P = \{b_1, \dots b_m\}$. Then for all $a \in \mathcal{P}_P$ and for all $s \in \mathcal{S}_P$, $\sum_{s' \in \mathcal{S}_P} q_P(s, s', a) = q_P(s, a) = 1$. Moreover, for each closure $P^c = P\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ with $\lambda_1, \dots, \lambda_n \in \mathbb{R}^+$ there exists a set of positive real numbers $\{k_1, \dots, k_m\}$ such that for each $s \in \mathcal{S}_{P^c}$ and $1 \leq i \leq m$

$$k_{b_i} = \frac{\sum_{s' \in \mathcal{S}_{P^c}} \mu_{P^c}(s') q_{P^c}(s', s, b_i)}{\mu_{P^c}(s)} \,,$$

where μ_{P^c} denotes any non-trivial invariant measure of the CTMC underlying P^c. Now observe that for all $s \in \mathcal{S}_{P^c}$ and $1 \leq i \leq m$,

$$k_{b_i} = q_{P^{cR}}(s^R, b_i) = \sum_{s' \in \mathcal{S}_{P^{cR}}} q_{P^{cR}}(s^R, s', b_i)$$

where s^R is the state of $S_{P^{cR}}$ corresponding to s according to Definition 3.4, i.e., for all $s^R \in \mathcal{S}_{P^{cR}}$, $[s^R]_{\sim_s} = \mathcal{S}_{P^{cR}}$.

($\Leftarrow$) Assume that for every closure P^c of P and for every $s^R \in \mathcal{S}_{P^{cR}}$ it holds $[s^R]_{\sim_s} = \mathcal{S}_{P^{cR}}$. Then for every type $a \neq \tau$ there exists a constant k_a such that

$$k_a = q_{P^{cR}}(s^R, a) = \frac{\sum_{s' \in \mathcal{S}_{P^c}} \mu_{P^c}(s') q_{P^c}(s', s, a)}{\mu_{P^c}(s)} \,,$$

where μ_{P^c} denotes any non-trivial invariant measure of the CTMC underlying P^c and s^R is the state of $S_{P^{cR}}$ corresponding to s according to Definition 3.4. In particular the property holds for every type $b_i \in \mathcal{A}_P$. $\square$

The following proposition states that both lumpable bisimulations and exact equivalences are invariant with respect to the closure of automata where any closure P^c of P is defined according to Definition 4.1.

Proposition 5.3. Let P and Q be two irreducible stochastic automata with $\mathcal{A}_P = \mathcal{A}_Q$, $\mathcal{P}_P = \mathcal{P}_Q = \{a_1, \dots, a_n\}$ and $\sim$ be an exact equivalence (resp., a lumpable bisimulation) over $P \cup Q$. Then for every closure $P^c = P\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ and $Q^c = Q\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ the relation $\sim' = \{(s_1^c, s_2^c) \in (\mathcal{S}_{P^c} \uplus \mathcal{S}_{Q^c}) \times (\mathcal{S}_{P^c} \uplus \mathcal{S}_{Q^c}) |\ (s_1, s_2) \in \sim\}$ is an exact equivalence (resp., a lumpable bisimulation) over $P^c \cup Q^c$.

PROOF. Let P and Q be two stochastic automata with $\mathcal{A}_P = \mathcal{A}_Q$, $\mathcal{P}_P = \mathcal{P}_Q = \{a_1, \dots, a_n\}$ and $\sim \subseteq (\mathcal{S}_P \uplus \mathcal{S}_Q) \times (\mathcal{S}_P \uplus \mathcal{S}_Q)$ be a lumpable bisimulation. Then for every closure $P^c = P\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ and $Q^c = Q\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ the relation $\sim' = \{(s_1^c, s_2^c) \in (\mathcal{S}_{P^c} \uplus \mathcal{S}_{Q^c}) \times (\mathcal{S}_{P^c} \uplus \mathcal{S}_{Q^c}) |\ (s_1, s_2) \in \sim\}$ is also a lumpable bisimulation. Let $C^c = \{(s_1^c, s_2^c) |\ (s_1, s_2) \in C\}$ for all $C \in (\mathcal{S}_P \uplus \mathcal{S}_Q)$.

Indeed for all $a \in \mathcal{T}_P \cup \mathcal{T}_Q$, $(s_1, s_2) \in \sim$ and for all $C \in (\mathcal{S}_P \uplus \mathcal{S}_Q)/\sim$ such that either $a \neq \tau$ or $a = \tau$ and $s_1, s_2 \notin C$, it holds

$$\sum_{s \in C} q(s_1, s, a) = \sum_{s \in C} q(s_2, s, a) \,.$$

If $a \in \mathcal{A}_P = \mathcal{A}_Q$ is an active type then, by Definition 4.1, $q_P(s_1, s_2, a) = q_{P^c}(s_1^c, s_2^c, a)$ for all $s_1, s_2 \in \mathcal{S}_P$ and $q_Q(s_1, s_2, a) = q_{Q^c}(s_1^c, s_2^c, a)$ for all $s_1, s_2 \in \mathcal{S}_Q$. Hence for all $a \neq \tau$, $a \in \mathcal{T}_P \cup \mathcal{T}_Q$, $(s_1^c, s_2^c) \in \sim'$ and for all $C^c \in (\mathcal{S}_P \uplus \mathcal{S}_Q)/\sim'$ it holds

$$\sum_{s^c \in C^c} q(s_1^c, s^c, a) = \sum_{s^c \in C^c} q(s_2^c, s^c, a)\,.$$

Moreover, by Definition 4.1, $q_{P^c}(s_1^c, s_2^c, \tau) = \sum_{i=1}^{n} q_P(s_1, s_2, a_i)\lambda_i + q_P(s_1, s_2, \tau)$ for all $s_1^c, s_2^c \in \mathcal{S}_{P^c}$ and $q_{Q^c}(s_1^c, s_2^c, \tau) = \sum_{i=1}^{n} q_Q(s_1, s_2, a_i)\lambda_i + q_Q(s_1, s_2, \tau)$ for all $s_1^c, s_2^c \in \mathcal{S}_{Q^c}$ Let $s_1^c \sim' s_2^c$ and $s_1^c, s_2^c \notin C^c$. Hence $s_1, s_2 \notin C$ and $s_1 \sim s_2$. Then

$$\begin{aligned}
\sum_{s^c \in C^c} q(s_1^c, s^c, \tau) &= \sum_{s \in C}((\sum_{i=1}^{n} q(s_1, s, a_i)\lambda_i) + q(s_1, s, \tau)) \\
&= \sum_{s \in C}(\sum_{i=1}^{n} q(s_1, s, a_i)\lambda_i) + \sum_{s \in C} q(s_1, s, \tau) \\
&= \sum_{i=1}^{n}(\sum_{s \in C} q(s_1, s, a_i)\lambda_i) + \sum_{s \in C} q(s_1, s, \tau) \\
&= \sum_{i=1}^{n}(\sum_{s \in C} q(s_2, s, a_i)\lambda_i) + \sum_{s \in C} q(s_2, s, \tau) \\
&= \sum_{s \in C}(\sum_{i=1}^{n} q(s_2, s, a_i)\lambda_i) + \sum_{s \in C} q(s_2, s, \tau) \\
&= \sum_{s \in C}((\sum_{i=1}^{n} q(s_2, s, a_i)\lambda_i) + q(s_2, s, \tau)) = \sum_{s^c \in C^c} q(s_2^c, s^c, \tau).
\end{aligned}$$

Let us now assume that $\sim \subseteq (\mathcal{S}_P \uplus \mathcal{S}_Q) \times (\mathcal{S}_P \uplus \mathcal{S}_Q)$ is a exact equivalence. Hence for all $a \in \mathcal{T}_P \cup \mathcal{T}_Q$, $(s_1, s_2) \in \sim$ and for all $C \in (\mathcal{S}_P \uplus \mathcal{S}_Q)/\sim$ we have

- $q(s_1, a) = q(s_2, a)$,
- $\sum_{s \in C} q(s, s_1, a) = \sum_{s \in C} q(s, s_2, a)$.

Let $a \in \mathcal{A}_P = \mathcal{A}_Q$ be an active type. Then, for all $(s_1^c, s_2^c) \in \sim'$,

$$\begin{aligned}
q(s_1^c, a) &= \sum_{C^c \in \mathcal{S}_P/\sim'} \sum_{s^c \in C^c} q(s_1^c, s^c, a) = \sum_{C \in \mathcal{S}_P/\sim} \sum_{s \in C} q(s_1, s, a) = q(s_1, a) \\
&= q(s_2, a) = \sum_{C \in \mathcal{S}_Q/\sim} \sum_{s \in C} q(s_2, s, a) = \sum_{C^c \in \mathcal{S}_Q/\sim'} \sum_{s^c \in C^c} q(s_2^c, s^c, a) = q(s_2^c, a)
\end{aligned}$$

and

$$\sum_{s^c \in C^c} q(s^c, s_1^c, a) = \sum_{s \in C} q(s, s_1, a) = \sum_{s \in C} q(s, s_2, a) = \sum_{s^c \in C^c} q(s^c, s_2^c, a)\,.$$

Moreover, for $\mathcal{P}_P = \mathcal{P}_Q = \{a_1, \ldots, a_n\}$, it holds:

$$\begin{aligned}
q(s_1^c, \tau) = &\sum_{C^c \in \mathcal{S}_P/\sim'} \sum_{s^c \in C^c} q(s_1^c, s^c, \tau) \\
&= \sum_{C \in \mathcal{S}_P/\sim} \sum_{s \in C} ((\sum_{i=1}^{n} q(s_1, s, a_i)\lambda_i) + q(s_1, s, \tau)) \\
&= \sum_{C \in \mathcal{S}_P/\sim} \sum_{s \in C} (\sum_{i=1}^{n} q(s_1, s, a_i)\lambda_i) + \sum_{s \in C} q(s_1, s, \tau) \\
&= \sum_{C \in \mathcal{S}_P/\sim} \sum_{i=1}^{n} (\sum_{s \in C} (q(s_1, s, a_i))\lambda_i) + \sum_{s \in C} q(s_1, s, \tau) \\
&= \sum_{C \in \mathcal{S}_Q/\sim} \sum_{i=1}^{n} (\sum_{s \in C} (q(s_2, s, a_i))\lambda_i) + \sum_{s \in C} q(s_2, s, \tau) \\
&= \sum_{C \in \mathcal{S}_Q/\sim} \sum_{s \in C} (\sum_{i=1}^{n} q(s_2, s, a_i)\lambda_i) + \sum_{s \in C} q(s_2, s, \tau) \\
&= \sum_{C \in \mathcal{S}_Q/\sim} \sum_{s \in C} ((\sum_{i=1}^{n} q(s_2, s, a_i)\lambda_i) + q(s_2, s, \tau)) \\
&= \sum_{C^c \in \mathcal{S}_Q/\sim'} \sum_{s^c \in C^c} q(s_2^c, s^c, \tau) = q(s_2^c, \tau)
\end{aligned}$$

and

$$\begin{aligned}
\sum_{s^c \in C^c} q(s^c, s_1^c, \tau) &= \sum_{s \in C}((\sum_{i=1}^{n} q(s, s_1, a_i)\lambda_i) + q(s, s_1, \tau)) \\
&= \sum_{s \in C}(\sum_{i=1}^{n} q(s, s_1, a_i)\lambda_i) + \sum_{s \in C} q(s, s_1, \tau) \\
&= \sum_{i=1}^{n}((\sum_{s \in C} q(s, s_1, a_i))\lambda_i) + \sum_{s \in C} q(s, s_1, \tau) \\
&= \sum_{i=1}^{n}((\sum_{s \in C} q(s, s_2, a_i))\lambda_i) + \sum_{s \in C} q(s, s_2, \tau) \\
&= \sum_{s \in C}(\sum_{i=1}^{n} q(s, s_2, a_i)\lambda_i) + \sum_{s \in C} q(s, s_2, \tau) \\
&= \sum_{s \in C}((\sum_{i=1}^{n} q(s, s_2, a_i)\lambda_i) + q(s, s_2, \tau)) = \sum_{s^c \in C^c} q(s^c, s_2^c, \tau).
\end{aligned}$$

$\square$

We are now in position to prove that the class of quasi-reversible stochastic automata is closed under exact equivalence.

Theorem 5.4. Let P and Q be two irreducible stochastic automata such that $P \sim_e Q$. If Q is quasi-reversible then also P is quasi-reversible.

Proof. We have to prove that:

1. The outgoing transitions for each passive type $a \in \mathcal{P}_P$ sums to unity.
2. For each closure $P^c = P\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ of P with $\lambda_1, \dots, \lambda_n \in \mathbb{R}^+$ there exists a set of positive real numbers $\{k_1, \dots, k_m\}$ such that for each $s \in \mathcal{S}_{P^c}$ and $1 \leq i \leq m$, Equation (3) is satisfied.

The first claim follows immediately from the first item of Definition 5.3. Now observe that, by Definition 5.3, if $P \sim_e Q$ then $\mathcal{P}_P = \mathcal{P}_Q$ and $\mathcal{A}_P = \mathcal{A}_Q$. Let $\mathcal{P}_P = \mathcal{P}_Q = \{a_1, \dots, a_n\}$ and $\mathcal{A}_P = \mathcal{A}_Q = \{b_1, \dots, b_m\}$. By Proposition 5.3, for any closure $P^c = P\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ and $Q^c = Q\{a_1 \leftarrow \lambda_1\} \dots \{a_n \leftarrow \lambda_n\}$ the relation $\sim' = \{(s_1^c, s_2^c) \in (\mathcal{S}_{P^c} \uplus \mathcal{S}_{Q^c}) \times (\mathcal{S}_{P^c} \uplus \mathcal{S}_{Q^c}) |\ (s_1, s_2) \in \sim \text{ and } (s_1, s_2) \in (\mathcal{S}_P \uplus \mathcal{S}_Q) \times (\mathcal{S}_P \uplus \mathcal{S}_Q)\}$ is an exact equivalence. By Theorem 5.2, the relation $\sim'' = \{(s_1^{cR}, s_2^{cR}) \in (\mathcal{S}_{P^{cR}} \uplus S_{Q^{cR}}) \times (\mathcal{S}_{P^{cR}} \uplus S_{Q^{cR}}) |\ (s_1, s_2) \in \sim'\}$ is a lumpable bisimulation. By Lemma 5.3 since Q is quasi-reversible then for all $s^R \in \mathcal{S}_{Q^{cR}}$ it holds $[s^R]_{\sim_s} = \mathcal{S}_{Q^{cR}}$, i.e., there exists a set of positive real numbers $\{k_{b_1}, \dots, k_{b_m}\}$ such that for each $s^R \in \mathcal{S}_{Q^{cR}}$ and $1 \leq i \leq m$

$$k_{b_i} = \sum_{s' \in \mathcal{S}_{Q^{cR}}} q_{Q^{cR}}(s^R, s', b_i) = \frac{\sum_{s' \in \mathcal{S}_{Q^c}} \mu_{Q^c}(s') q_{Q^c}(s', s, b_i)}{\mu_{Q^c}(s)},$$

which can be written as

$$k_{b_i} = \frac{\sum_{C \in \mathcal{S}_{Q^c}/\sim_e} \sum_{s' \in C} \mu_{Q^c}(s') q_{Q^c}(s', s, b_i)}{\mu_{Q^c}(s)} .$$

By Proposition 5.1, $\sim_e$ induces an exact lumping on the CTMC underlying Q^c and, by Proposition 2.3, for all s and s' in the same equivalence class $\mu_{Q^c}(s) = \mu_{Q^c}(s')$. Hence we can write

$$k_{b_i} = \frac{\sum_{C \in \mathcal{S}_{Q^c}/\sim_e} \mu_{Q^c}(C) \sum_{s' \in C} q_{Q^c}(s', s, b_i)}{\mu_{Q^c}(s)} .$$

where $\mu_{Q^c}(C)$ denotes $\mu_{Q^c}(s)$ for an arbitrary state $s \in C$. Now from the fact that $P^c \sim_e Q^c$, we have that for each class $C \in \mathcal{S}_{Q^c}/\sim_e$ there exists a class $C' \in \mathcal{S}_{P^c}/\sim_e$ such that all the states $s \in C$ are equivalent to the states in C'. Moreover, by Definition 5.3, we have $\sum_{s' \in C} q_{Q^c}(s', s_1, b_i) = \sum_{s' \in C'} q_{P^c}(s', s_2, b_i)$ for every state $s_1 \sim_e s_2$ with $s_1 \in Q^c$ and $s_2 \in P^c$. Therefore, we can write:

$$\begin{aligned} k_{b_i} &= \frac{\sum_{C \in \mathcal{S}_{Q^c}/\sim_e} \mu_{Q^c}(C) \sum_{s' \in C} q_{Q^c}(s', s_1, b_i)}{\mu_{Q^c}(s_1)} \\ &= \frac{\sum_{C \in \mathcal{S}_{Q^c}/\sim_e} \mu_{Q^c}(C) \sum_{s' \in C} q_{P^c}(s', s_2, b_i)}{\mu_{Q^c}(s_1)} \\ &= \frac{\sum_{C' \in \mathcal{S}_{P^c}/\sim_e} K\mu_{P^c}(C') \sum_{s' \in C'} q_{P^c}(s', s_2, b_i)}{K\mu_{P^c}(s_2)} \\ &= \frac{\sum_{s' \in \mathcal{S}_{P^c}} \mu_{P^c}(s') q_{P^c}(s', s_2, b_i)}{\mu_{P^c}(s_2)} , \end{aligned}$$

where K is the positive constant given by Theorem 5.1. Summing up, since every closure Q^c of Q corresponds to a closure P^c for P and Q^c satisfies Equation (3) for all states s and active types b_i, then the set of positive rates $\{k_{b_i}\}$ defined for Q^c are the same that satisfy Equation (3) for P^c. Therefore, P is also quasi-reversible. □

Example 5.2. Let us consider the automata Q and P depicted in Figure 1 and Figure 2, respectively. We already observed in Example 5.1 that there exists an exact equivalence $\sim'$ such that $n, n\kappa_1$ and $n\kappa_2$ belong to the same equivalence class, where n is a state of Q and $n\kappa_i$ belongs to the state space of P. Then, since Q is well-known to be quasi-reversible, by Theorem 5.1 also P is quasi-reversible. As a consquence, the queueing station modelled by P can be embdded in quasi-reversible product-form queueing networks maintining the property that the equilibrium distribution is separable. □

The next example shows that, differently from exact equivalence, lumpable bisimulation does not preserve quasi-reversibility.

Example 5.3. Consider the automaton R depicted in Figure 3. It is easy to

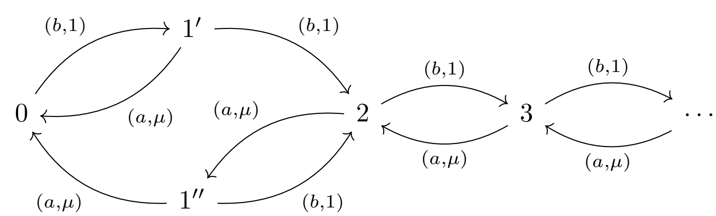

Figure 3: Stochastic automaton strongly equivalent to a Jackson queue.

prove that R is lumpable bisimilar to Jackson's queue Q depicted in Figure 1. However, R is not quasi-reversible, i.e., the corresponding reversed automaton is not lumpable bisimilar to a single-state automaton. More precisely, one can observe that in the reversed automaton there is one type a transition exiting from state 0^R but there is no type a transition from state $1'^R$. This is sufficient to claim that states 0^R and $1'^R$ cannot belong to the same equivalence class. $\square$

The following result is an immediate consequence of Theorems 4.1 and 5.4.

Corollary 5.3. Let P, P', Q, Q' be irreducible stochastic automata such that $P \sim_e P'$ and $Q \sim_e Q'$. If P and Q have a quasi-reversibility based product-form then also P' and Q' are in product-form.

6. Algorithms for Lumpable Bisimulation and Exact Equivalence

In [30], Valmari and Franceschinis proposed an algorithm for computing a lumpability over directed weighted graphs. In particular, the algorithm exploits a partition refinement strategy, similar to that of Paige-Tarjan's algorithm for bisimulation [25], enriched with a sorting of classes. In this section we exploit Valmari and Franceschinis' algorithm to design procedures for computing lumpable bisimulations and exact equivalences.

6.1. Compatible Relations over Labeled Weighted Graphs

We start by considering a generalization of the compatibility problem considered in [30].

We recall some basic definitions over graphs. A labeled graph $G = (S, T, E)$ consists of a finite set of nodes S, a finite set of labels T and a finite set of edges $E \subseteq S \times T \times S$. A weighted graph $G = (S, E, W)$ is a graph (S, E) together with a weighting function $W : E \to \mathbb{R}$ that associates to each edge a real number. A labeled weighted graph $G = (S, T, E, W)$ is a labeled graph, i.e., $E \subseteq S \times T \times S$, together with a weighting function W. Given $S' \subseteq S$, we denote by $W(s, t, S')$ the sum of the weights of the edges from s to S' having label t. Notice that given a labeled weighted graph $G = (S, T, E, W)$ and a label $t \in T$ it is possible to consider the weighted graph $G \restriction t = (S, E \restriction t, W \restriction t)$, where only the edges with label t together with their weights are considered. We use the notation

$(E \restriction t)^{-1}(s)$ to denote the set of nodes that reach s in $G \restriction t$ (i.e., the pre-image of s in $G \restriction t$).

The following definition of compatibility extends that of [30] to weighted labeled graphs.

Definition 6.1. *(Compatibility)* Let $G = (S, T, E, W)$ be a labeled weighted graph and $R \subseteq V \times V$ be an equivalence relation. R is said to be *compatible* with G if for each $t \in T$, for each $C, C' \in S/R$, and for each $s, s' \in C$ it holds that $W(s, t, C') = W(s', t, C')$.

Lemma 6.1. Let G be a labeled weighted graph and $R \subseteq S \times S$ be an equivalence relation. There exists a unique largest equivalence relation included in R and compatible with G.

Proof. The identity relation is always compatible and it is included in R, hence there exists at least one compatible relation included in R.

It is easy to prove that if $R_1, R_2 \subseteq R$ are two equivalence relations included in R and compatible with G, then $R_1 \sqcup R_2$, the smallest equivalence relation such that $R_1, R_2 \subseteq R_1 \sqcup R_2$, is included in R and compatible with G.

□

As a consequence of the above lemma, if we consider the total relation $R = S \times S$, we get that there exists a unique largest equivalence relation compatible with G.

Notice that a relation is compatible with a labeled weighted graph G if and only if for each t in T it is compatible with the weighted graph $G \restriction t$ with respect to the definition in [30].

Let $G = (S, T, E, W)$ be a labeled weighted graph and $R \subseteq S \times S$ be an equivalence relation the *labeled weighted compatibility problem* requires to compute the largest equivalence relation included in R compatible with G.

In Section 6.4 we will present an algorithm to efficiently solve the labeled weighted compatibility problem. Before presenting the algorithm, in Section 6.2 (Section 6.3) we will see how the problem of computing the lumpable bisimulation (exact equivalence, respectively) can be reduced to the labeled weighted compatibility problem.

6.2. Lumpable Bisimulation as Labeled Weighted Compatibility

Given a stochastic automaton P, in the following we denote by $q[s, S, a]$ the term $\sum_{s'' \in S} q(s, s'', a)$, with $S \subseteq \mathcal{S}_P$.

Definition 6.2. *(Lumping Graph)* Let P be a stochastic automaton. The *lumping graph* of P is the labelled weighted graph $\mathcal{L}_P = (\mathcal{S}_P, \rightsquigarrow_P, WL_P)$, where WL_P is the function which associates to each tuple in $\rightsquigarrow_P$ the following value:

$$WL_P(s_1, a, s_2) = \begin{cases} q(s_1, s_2, a) & \text{if } a \neq \tau \vee s_1 \neq s_2 \\ -q[s_1, \mathcal{S}_P \setminus \{s_1\}, a] & \text{otherwise} \end{cases}$$

The following result shows how to reduce the problem of computing the lumpable bisimulation $\sim_s$ over a stochastic automaton P to a labeled weighted compatibility problem.

Theorem 6.2. *(Lumpable bisimulation as compatibility)* Let P be a stochastic automaton and $\mathcal{L}_P$ be its lumping graph. The largest relation compatible with $\mathcal{L}_P$ is $\sim_s$.

Proof. Let R be the largest relation compatible with $\mathcal{L}_P$ we prove that $R = \sim_s$ by proving that $R \subseteq \sim_s$ and $\sim_s \subseteq R$.

In order to prove that $R \subseteq \sim_s$ it is sufficient to prove that R is a lumpable bisimulation. Let $(s_1, s_2) \in R$ and $C \in \mathcal{S}_P/R$. If $a \neq \tau$, then $\sum_{s' \in C} q(s_1, s', a) = q[s_1, C, a] = WL_P(s_1, a, C) = WL_P(s_2, a, C) = q[s_2, C, a] = \sum_{s' \in C} q(s_1, s', a)$. Similarly, if $a = \tau$ and $s_1, s_2 \notin C$ we get the thesis, since $s_i \notin C$ implies $q[s_i, C, \tau] = WL_P(s_i, \tau, C)$, for $i = 1, 2$.

In order to prove that $\sim_s \subseteq R$ it is sufficient to prove that $\sim_s$ is compatible. Let $s_1 \sim_s s_2$, and $C \in \mathcal{S}_P/\sim_s$, we have to prove that $WL_P(s_1, a, C) = WL_P(s_2, a, C)$, for each $a \in \mathcal{T}_P$. The only interesting case is the case $a = \tau$ and $s_1, s_2 \in C$. In this case $WL_P(s_1, \tau, C) = \sum_{s' \in C, s' \neq s_1} q(s_1, s', \tau) - q[s_1, \mathcal{S}_P \setminus \{s\}, \tau] = -\sum_{s' \notin C} q(s_1, s', \tau) = -\sum_{C' \neq C} WL_P(s_1, \tau, C')$. Since, neither s_1 nor s_2 belongs to any of the classes C' involved in this last sum, from the other cases we get $-\sum_{C' \neq C} WL_P(s_1, \tau, C') = -\sum_{C' \neq C} WL_P(s_2, \tau, C')$ and now reasoning on s_2 this last is $WL_P(s_2, \tau, C)$. $\square$

6.3. Exact Equivalence as Labeled Weighted Compatibility

Definition 6.3. *(Inverse Graph)* Let P be a stochastic automaton. The *inverse graph* of P is the labelled weighted graph $\mathcal{I}_P = (\mathcal{S}_P, \leftsquigarrow_P, WI_P)$, where

$$(s_1, a, s_2) \in \leftsquigarrow_P \quad \text{iff} \quad (s_2, a, s_1) \in \rightsquigarrow_P$$

and WI_P is the function which associates to each tuple in $\leftsquigarrow_P$ the following value:

$$WI_P(s_1, a, s_2) = q(s_2, s_1, a)$$

Definition 6.4. *(Initial Equivalence)* Let P be a stochastic automaton. The *initial equivalence* of P is the equivalence relation $IE_P \subseteq \mathcal{S}_P \times \mathcal{S}_P$ defined as follows:

$$(s_1, s_2) \in IE_P \quad \text{iff} \quad q(s_1, a) = q(s_2, a) \text{ for each } a \in \mathcal{T}_P$$

The following result shows how to reduce the problem of computing the exact equivalence $\sim_e$ over a stochastic automaton P to a labeled weighted compatibility problem.

Theorem 6.3. *(Exact equivalence as compatibility)* Let P be a stochastic automaton, $\mathcal{I}_P$ be its inverse graph, and IE_P be its initial equivalence. The largest relation included in IE_P compatible with $\mathcal{I}_P$ is $\sim_e$.

PROOF. Let R be the largest relation included in IE_P compatible with $\mathcal{I}_P$. We prove that $R = \sim_e$ by proving that $R \subseteq \sim_e$ and $\sim_e \subseteq R$.

In order to prove that $R \subseteq \sim_e$ it is sufficient to prove that R is an exact equivalence. Let $(s_1, s_2) \in R$ and $C \in \mathcal{S}_P/R$. Since $R \subseteq IE_P$ we have that $q(s_1, a) = q(s_2, a)$, for each $a \in \mathcal{T}_P$. Moreover, $\sum_{s' \in C} q(s', s_1, a) = WI_P(s_1, a, C) = WI_P(s_2, a, C) = \sum_{s' \in C} q(s', s_1, a)$.

In order to prove that $\sim_e \subseteq R$ it is sufficient to prove that $\sim_e$ is included in IE_P and compatible. Let $s_1 \sim_e s_2$. It holds that for each $a \in \mathcal{T}_P$ $q(s_1, a) = q(s_2, a)$, hence $(s_1, s_2) \in IE_P$.

Moreover, let $C \in \mathcal{S}_P/\sim_e$, we prove that $WI_P(s_1, a, C) = WI_P(s_2, a, C)$, for each $a \in \mathcal{T}_P$. Indeed $WI_P(s_1, a, C) = \sum_{s' \in C} q(s', s_1, a) = \sum_{s' \in C} q(s', s_2, a) = WI_P(s_1, a, C)$. □

6.4. Efficient Labeled Weighted Compatibility Algorithm

As it is common in bisimulation and lumpability algorithms, instead of working with equivalence relations, we work on the corresponding partitions over S, i.e., $\mathcal{R}$ is the initial partition and we are looking for the coarsest refinement of $\mathcal{R}$ compatible with G.

Our algorithm works exactly as the one in [30] except that we need to deal with labels on edges. The basic idea in Algorithm 1 is that we start with an initial partition $\mathcal{P}$ and each iteration of the main while loop at line 7 refines $\mathcal{P}$ by splitting blocks. At the end of the computation $\mathcal{P}$ is the desired partition. UB stands for Unused Blocks and contains the blocks of $\mathcal{P}$ which have not yet been used as splitters. TB stands for Touched Blocks and contains the blocks of $\mathcal{P}$ that could be split in the current iteration. The array w records for each $s \in S$ the total weight of the t-edges from s to the current splitter block C. TS stands for Touched States and contains the states of S that reach through t-edges the splitter C. Hence, at line 19 $w[s]$ has a numerical value if and only if s is in TS. For each s in the splitter C, the for loop at lines 10-11 stores in $E[s, t]$ the pre-image of s with respect to t-edges. This is necessary since we process many labels. The splitter itself could get split with respect to a label t. In this case we need to ensure that for all the labels processed after t we still refer to the entire pre-image of C. This pre-computation will guarantee the correctness of the "exclude-the-largest" policy at line 36. The for loop at lines 12-37 takes care of using C as splitter with respect to each label. The operations performed inside this loop coincide with that of the algorithm in [30]. In particular, the splits are performed inside the while loop at lines 23-36. The key ingredient for obtaining a time performant algorithm is line 36: if block B, which has just been split, has already been used as splitter, then the largest of the sub-blocks of B is not included in the list UB of future splitters. The notation $[B,]^?$ means that B is considered only if it is different from B_1. The commented instructions at lines 3 and 9 take care of storing a partition $\mathcal{S}$ which is always coarser than $\mathcal{P}$. The block S_C of $\mathcal{S}$ is the one that includes the block C of $\mathcal{P}$. Intuitively, this is the partition of super-blocks. At each iteration the partition $\mathcal{P}$ will be "stable" with respect to $\mathcal{S}$ in the following sense:

$$\forall t \in T \, \forall B \in \mathcal{P} \, \forall C \in \mathcal{S} \, \forall s_1, s_2 \in B \, (W(s_1, t, C) = W(s_2, t, C))$$

It is quite easy to prove that Algorithm 1 always terminates.

Lemma 6.4. *(Termination)* LWC$(G, \mathcal{R})$ terminates.

Proof. We only have to prove that the while loop at line 7 terminates. Let UB_i ($\mathcal{P}_i$) be the value of UB ($\mathcal{P}$, respectively) at the i-th iteration of the while loop. We have that for each i it holds that $|\mathsf{UB}_i|$ is always finite and $|\mathcal{P}_i| \leq |S|$.

We prove that if $|\mathsf{UB}_{i+1}| \geq |\mathsf{UB}_i|$, then $|\mathcal{P}_{i+1}| > |\mathcal{P}_i|$. We extract one block from UB at line 8, hence to get $|\mathsf{UB}_{i+1}| \geq |\mathsf{UB}_i|$ we should insert at least one block in it. This implies that at least one split has been performed, i.e., $|\mathcal{P}_{i+1}| > |\mathcal{P}_i|$.

Hence, since for each i it holds that $|\mathcal{P}_i| \leq |S|$, there exists j such that for each $k > j$ we have $|\mathcal{P}_{k+1}| = |\mathcal{P}_k|$. So by contraposition, we get that for each $k > j$ it holds that $|\mathsf{UB}_{k+1}| < |\mathsf{UB}_k|$. As a consequence the while loop terminates. □

We claim that Algorithm 1 solves the labeled weighted compatibility problem. Our proof of correctness extends to the labeled case the one in [30].

First we prove that we can safely modify the input without substantially changing the result. In particular, we move from $G = (S, T, E, W)$ to $G' = (S', T, E', W')$, where

- $S' = S \cup \{s_\perp\}$;
- $E' = E \cup \{(s, t, s_\perp) \mid s \in S, t \in T\}$;
- $W'(s, t, s') = W(s, t, s')$ for each $s' \in S$, while $W'(s, t, s_\perp) = -W(s, t, S)$.

Hence, for each label t we have $W'(s, t, S') = 0$. Given a partition $\mathcal{X}$ over S we denote by $\mathcal{X}'$ the partition $\mathcal{X} \cup \{\{s_\perp\}\}$ over S'. It is immediate to prove that $\mathcal{P}$ is a refinement of $\mathcal{R}$ compatible with G if and only if $\mathcal{P}'$ is a refinement of $\mathcal{R}'$ compatible with G'.

Moreover, we can safely uncomment line 3 and line 9 without changing the behaviour of the algorithm, i.e., the common variables have the same values at each step.

Lemma 6.5. *(Algorithm enrichment)* The behaviour of LWC$(G, \mathcal{R})$ does not change if we replace the skip instructions at line 3 and line 9 with the commented instructions.

We denote by LWC$'(_, _)$ the variant of LWC$(_, _)$ with the skips replaced by the comments. Hence, we get the following result.

Lemma 6.6. *(Graph enrichment)* It holds that LWC$(G, \mathcal{R})$ is correct if and only if LWC$(G', \mathcal{R}')$ is correct.

Hence, by Lemma 6.6 and Lemma 6.5 it is sufficient for us to prove that $\text{LWC}'(G', \mathcal{R}')$ is correct. To avoid confusion in $\text{LWC}'(G', \mathcal{R}')$ we replace the variable $\mathcal{P}$ with $\mathcal{P}'$.

We first prove that the output of the algorithm is coarser than the coarsest refinement of $\mathcal{R}'$ compatible with G'. Notice that at this stage we do not prove that the output is compatible with G'.

Lemma 6.7. *(Necessity of spits)* Let $\mathcal{P}'$ be the output of $\text{LWC}'(G', \mathcal{R}')$. If s_1 and s_2 belongs to different blocks in $\mathcal{P}'$, then in each compatible refinement of $\mathcal{R}'$ the elements s_1 and s_2 belongs to different blocks.

Proof. It is equivalent to prove that s_1 and s_2 belongs to different blocks in $\mathcal{Q}'$ the coarsest refinement of $\mathcal{R}'$ compatible with G'. Let $\mathcal{P}'_i$ be the partition computed by $\text{LWC}'(G', \mathcal{R}')$ before the $i+1$-th repetition of the main while loop (line 7). We prove by induction on i that $\mathcal{Q}'$ is always a refinement of $\mathcal{P}'_i$.

Base: $i = 0$. We have $\mathcal{P}'_0 = \mathcal{R}'$ hence the thesis hold since $\mathcal{Q}'$ is by definition a refinement of $\mathcal{R}'$.

Inductive step: we assume that the thesis holds for $i < j$ and we prove that $\mathcal{Q}'$ is a refinement of $\mathcal{P}'_j$. By inductive hypothesis $\mathcal{Q}'$ is a refinement of $\mathcal{P}'_{j-1}$. Let s_1 and s_2 be such that $s_1 \in B_1^j \in \mathcal{P}'_j$ and $s_2 \in B_2^j \in \mathcal{P}'_j$ with $B_1^j \neq B_2^j$, while $s_1, s_2 \in B^{j-1} \in \mathcal{P}'_{j-1}$. We have to prove that s_1 and s_2 belongs to different blocks also in $\mathcal{Q}'$. By the hypothesis we get that there exists $t \in T$ and $B' \in \mathcal{P}'_{j-1}$ such that $W(s_1, t, B') \neq W(s_2, t, B')$. Since $B' \in \mathcal{P}'_{j-1}$ and $\mathcal{Q}'$ is a refinement of $\mathcal{P}'_{j-1}$ we have that there exists $Q_1, \ldots, Q_k \in \mathcal{Q}'$ such that $B' = Q_1 \cup \cdots \cup Q_k$. Hence

$$\sum_{h=1}^{k} W(s_1, t, Q_h) = W(s_1, t, B') \neq W(s_2, t, B') = \sum_{h=1}^{k} W(s_1, t, Q_h)$$

This implies that there exists k' such that $W(s_1, t, Q_{k1}) \neq W(s_2, t, Q_{k'})$. Hence, since $\mathcal{Q}'$ is compatible with G', s_1 and s_2 cannot belong to the same block of $\mathcal{Q}'$. $\square$

It is immediate to see that the output of the algorithm is a refinement of $\mathcal{R}'$. Hence, it only remains to prove that the output of the algorithm is compatible with G'.

We now prove that at each iteration the current partition is "stable" with respect to the super-blocks partition. This is the only result in which we really exploit both the super-blocks partition and the new element $s_\perp$.

Lemma 6.8. *(Stability invariant)* During the execution of $\text{LWC}'(G', \mathcal{R}')$ it is always true that for each $t \in T$, for each $B \in \mathcal{P}'$, for each $S \in \mathcal{S}$, for each $s_1, s_2 \in B$ it holds that $W(s_1, t, S) = W(s_2, t, S)$.

Proof. We recall that in the algorithm S_C is the block of $\mathcal{S}$ which includes the block C of $\mathcal{P}'$. It is immediate to see that $\mathcal{S}$ is always coarser than $\mathcal{P}'$. Hence S_C is correctly defined.

Let $\mathcal{P}'_i$ be the partition computed by LWC$'(G', \mathcal{R}')$ before the $i+1$-th repetition of the main while loop (line 7) and similarly $\mathcal{S}_i$ be the partition computed by LWC$'(G', \mathcal{R}')$ before the $i+1$-th repetition of the same loop. We prove the thesis by induction on i.

Base: $i = 0$. The thesis is true since for each $t \in T$ it holds that $W'(s, t, S \cup \{s_\perp\}) = 0$.

Inductive step: we assume that the thesis holds for $i < j$ and we prove it for j. If we consider a block in $\mathcal{S}_j$ that was also in $\mathcal{S}_{j-1}$, then the thesis trivially holds by inductive hypothesis. Let C and $S_C \setminus C$ be the two new blocks in $\mathcal{S}_j$ not in $\mathcal{S}_{j-1}$. These have been added to $\mathcal{S}_j$ during the $j-1$th execution of the while loop. During this step all the blocks of $\mathcal{P}'_{j-1}$ have been split with respect to C leading to $\mathcal{P}'_j$. Hence, for each $t \in T$, for each $B \in \mathcal{P}'_j$, and for each $s_1, s_2 \in B$ it holds that $W'(s_1, t, C) = W'(s_2, t, C)$. Moreover, as far as $S_C \setminus C$ is concerned $W'(s_1, t, S_C \setminus C) = W'(s_1, t, S_C) - W'(s_1, t, C)$. This last by inductive hypothesis and since $W'(s_1, t, C) = W'(s_2, t, C)$, is equal to $W'(s_2, t, S_C) - W'(s_2, t, C) = W'(s_2, t, S_C \setminus C)$. □

Another invariant on super-blocks is useful to prove correctness.

Lemma 6.9. *(Cardinality invariant)* During the execution of LWC$'(G', \mathcal{R}')$ it is always true that for each $S \in \mathcal{S}$, if S is the union of k blocks of $\mathcal{P}$, then the list UB contains at least $k-1$ of such blocks.

Proof. Let $\mathcal{P}'_i$, $\mathcal{S}_i$, and UB_i be the variables before the $i+1$-th repetition of the main while loop (line 7). We prove the thesis by induction on i.

Base: $i = 0$. We have that UB_0 contains all the blocks that are in $\mathcal{S}_0$.

Inductive step: we assume that the thesis holds for $i < j$ and we prove it for j. For all the blocks that are included in a block S of $\mathcal{S}_{j-1}$ and are in UB_{j-1} all their sub-blocks are added to UB_j. For all the blocks that are included in a block S of $\mathcal{S}_{j-1}$ and are not in UB_{j-1} all their sub-blocks but one are added to UB_j. Hence, the thesis holds. □

We are now ready to prove the correctness of our algorithm.

Theorem 6.10. *(Correctness)* LWC$'(G', \mathcal{R}')$ returns the coarsest refinement of $\mathcal{R}'$ compatible with G'.

Proof. By Lemma 6.7 we have that LWC$'(G', \mathcal{R}')$ returns a partition which is coarser than any refinement of $\mathcal{R}'$ compatible with G'.

By Lemma 6.9 we get that when LWC$'(G', \mathcal{R}')$ terminates, since UB_j is empty it must be $\mathcal{P}' = \mathcal{S}$. Hence, by Lemma 6.8 $\mathcal{P}'$ is compatible with G'. □

As far as the complexity is concerned we can prove that our algorithm has the same time and space complexities of the one in [30].

Lemma 6.11. LWC$(G, \mathcal{R})$ requires time $O(|S| + |E| \log |S|)$ without considering the cost of the sorting operations at line 33.

Proof. We start proving the thesis in the case in which G is connected, i.e., $|S| = O(|E|)$.

The initialization instructions before line 7 requires time $\Theta(|S|)$.

Let us consider one iteration of the main loop at line 7. The for loop at line 10 requires time $\Theta(|C| + |E^{-1}(C)|)$, where $E^{-1}(C)$ is the set of edges reaching C with any possible label. For a given label t, the for loop at line 12 without considering the sorting operations requires time $\Theta(|(E \upharpoonright t)^{-1}(C)|)$. Hence, the main loop without considering the sorting operations costs $\Theta(|C| + |E^{-1}(C)|)$. Notice that in order to obtain this result which does not depend on $|T|$ one should compute just before line 10 the set T_C of labels involved in the preimage of C. This can be done again at a cost of $\Theta(\Theta(|C| + |E^{-1}(C)|))$. Then the for loops at lines 10 and 12 should range over T_C instead of T.

The complexity $\Theta(|C| + |E^{-1}(C)|)$ can be distributed over the elements of C considering a cost of $\Theta(1 + |E^{-1}(s')|)$ for each $s' \in C$.

Globally over all the executions of the main while loop a node s' can belong to a splitter block C at most $\Theta(log|S|)$ times since once the block in which s' originally belongs is removed from UB, the largest sub-block is never reinserted in UB. Hence, every time s' belongs to a splitter the size of the splitter is at least half of the previous time that s' was in a splitter. So, s' is in a splitter $O(\log|S|)$ times, which gives us a complexity of $O(\sum_{s' \in S}((\log|S|)(1 + |E^{-1}(s')|))) = O((|S| + |E|)\log|S|) = O(|E|\log|S|)$.

In the general case, i.e., if G is not connected, we can split the complexity over its connected components obtaining a cost of $O(|S| + |E|\log|S|)$, where the additional $O(|S|)$ takes into consideration all the initialization steps that are present even if $|E|\log|S| < |S|$. □

Lemma 6.12. The sorting operations in LWC$(G, \mathcal{R})$ runs in $O(|E|\log|S|)$.

Proof. The proof proceeds exactly as in [30]. □

As a direct consequence of Lemma 6.11 and Lemma 6.12 we get the following result.

Theorem 6.13. LCW$(G, \mathcal{R})$ requires time $O(|S|+|E|\log|S|)$ and space $O(|S|+|E|)$.

Notice that the models considered in this paper generate strongly connected graphs with $|S| = O(|E|)$, hence the time complexity of our algorithm can be simplified into $O(|E|\log|S|)$, while the space complexity becomes $O(|E|)$.

7. Case Study and Performance Tests

The algorithms for lumpable bisimulation and exact equivalence described in Section 6 have been integrated in the PEPA Eclipse plug-in which is available at `https://github.com/Bakuriu/PEPA-Eclipse-Plug-in`.

As discussed in Section 3, the synchronisation semantics of stochastic automata can be seen as the active/passive synchronisation in PEPA. In particular, stochastic automata can be easily translated into PEPA components having the same semantics.

In this section we consider a case study, taken from [7], consisting of a web server cluster interacting with a set of clients. Specifically, the server cluster provides contents to users and allows content creators to add new contents on the servers. The system is meant to satisfy certain quality of service requirements regarding availability and response-time. In order to meet such requirements the system skews the prioritisation for fast reads over writes, so that writes are buffered and only processed at a time when there are only few reads. The consequence is that a reader may not be able to read the latest updates, although high availability is maintained.

Each server can fail, and be repaired, independently. If all the servers fail a special recovery mechanism is triggered which recovers the whole cluster.

7.1. The server model

Each server receives read requests, which, before being satisfied, cause a read lookup by the server. In order to successfully complete a write request it is necessary that no server is performing a read lookup (i.e., no server should be in the state $\mathsf{ServerRead}$ below), so that all servers can perform the write operation simultaneously.

The server can fail, and during the failure time no read request can be accepted. We assume that write actions occuring during the server failure time are ignored; when the server will be restarted it will be re-synchronised with the other servers in the cluster.

The SA representing a single server is depicted in Figure 4.

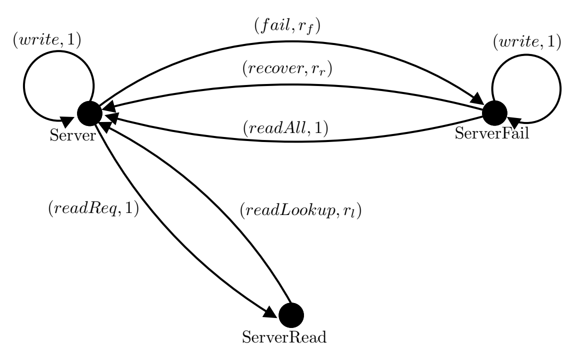

Figure 4: The SA for a server.

7.2. The server cluster model

The cluster of servers is able to witness the failures of the server and trigger their recovery. When all servers have failed the whole cluster is restarted, recovering all failures simultaneously.

The SA for the cluster of servers is represented in Figure 5.

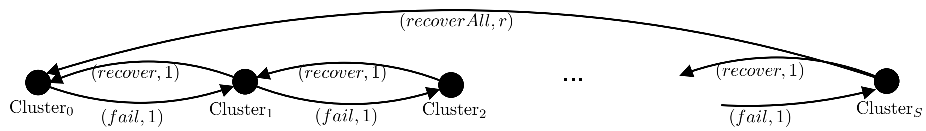

Figure 5: The SA for the cluster.

With this definition we can represent the cluster of servers as:

$$\mathsf{Servers} \stackrel{\text{def}}{=} \left(\mathsf{Server} \underset{L}{\otimes} \mathsf{Server} \underset{L}{\otimes} \ldots \underset{L}{\otimes} \mathsf{Server}\right) \underset{L'}{\otimes} \mathsf{Cluster}_0$$

with $L = \{write, recoverAll\}$ and $L' = L \cup \{recover\}$.

7.3. The model for buffered writes

The write requests that the system receives are buffered and performed only when no server is reading its database. The read buffer can hold up to B buffered writes. Whenever all the servers are available to perform a write operation we assume that all of the buffered writes are performed.

The SA for the buffered write is represented in Figure 6.

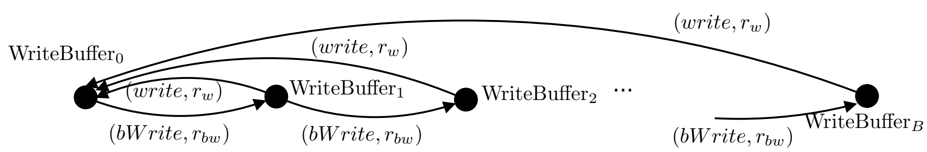

Figure 6: The SA for the buffered write.

7.4. The model for the users

The system has two different kind of users: authors, who publish content to the servers by triggering the write requests, and readers, who trigger the read requests from the servers. We assume that the number of readers is higher than the number of writers, and as such we have prioritized reads over writes.

We consider a fixed time period in which W writes and R read requests occur. At the end of this time period the writers and readers are reset and an other W write and R read requests occur. This assumption is reasonable for a stable system, for example an online newspaper which publishes everyday a specific number of articles and has a mostly constant number of views in a month.

The authors of content cannot trigger a server *write* operation, but can only write to the buffer as represented in Figure 7.

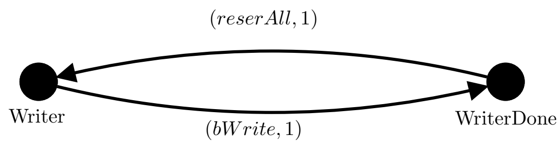

Figure 7: The SA for a writer.

The Writers SA is the composition of W content authors of the system:

$$\mathsf{Writers} \stackrel{\text{def}}{=} \mathsf{Writer} \underset{K}{\otimes} \mathsf{Writer} \underset{K}{\otimes} \dots \underset{K}{\otimes} \mathsf{Writer}$$

where $K = \{resetAll\}$.

The readers send read requests and wait for the lookup by the servers as illustrated in Figure 8.

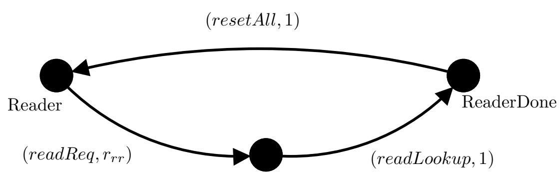

Figure 8: The SA for a reader.

Again the Readers SA is the synchronization of R readers of the system:

$$\mathsf{Readers} \stackrel{\text{def}}{=} \mathsf{Reader} \underset{K}{\otimes} \mathsf{Reader} \underset{K}{\otimes} \dots \underset{K}{\otimes} \mathsf{Reader}$$

The system will always contain a component dedicated to triggering the reset of the readers and writers, as represented in Figure 9.

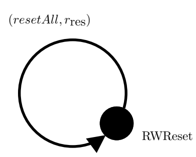

Figure 9: The SA for the RWreset.

7.5. The complete system

We are finally able to define the SA that represents the whole system.

$$\begin{aligned}\mathsf{Environment} &\stackrel{\text{def}}{=} \mathsf{Writers} \underset{K}{\otimes} \mathsf{Readers} \underset{K}{\otimes} \mathsf{RWReset}\\ \mathsf{WebCluster} &\stackrel{\text{def}}{=} \mathsf{Servers} \underset{\{write\}}{\otimes} \mathsf{Buffer}_0\\ \mathsf{System} &\stackrel{\text{def}}{=} \mathsf{Environment} \underset{N}{\otimes} \mathsf{WebCluster}\end{aligned}$$

where $N = \{bWrite, readReq, readLookup\}$.

7.6. Performance tests

We considered the performance of our algorithms for different values of parameters S, B, R, W of the system. The tests were run on a Acer Aspire 5742G machine, which is equipped with a quad core Intel Core i5 M 480 2.67GHz, 4GB of RAM memory.

Table 2 contains the results obtained without using any aggregation technique, and it shows that it is not possible to perform the steady-state analysis even for moderate values of the parameters due to memory errors.

Table 3 shows the results obtained using contextual lumpability. In this case we were able to compute the steady-state distribution for the aggregated state space in all cases.

Table 4 contains the results obtained using exact equivalence. In this case study, exact equivalence and contextual lumpability do not coincide, as it is easily verified comparing the aggregated state space sizes. The exact equivalence produces a finer aggregation, which is to be expected since it enforces that all states lumped together have the same steady-state probability. The reduction in the state space sizes is thus significantly worse than the contextual lumpability case, achieving a 13-fold reduction in the number of both states and transitions, and our machine failed to compute the steady-state distribution over the aggregated state space for the parameters $4, 3, 3, 4$ and $3, 3, 3, 6$. On the other hand, if one desires to derive the steady-state distribution of the non aggregated automaton given that of the aggregated, we have to consider that this is much more computationally efficient in the case of an aggregation based on exact equivalence with respect to an aggregation based on contextual lumpability.

An important aspect of this case study is that the aggregations we have achieved are not possible to achieve using only the aggregating arrays or other syntactical means, since the repeated components are not in a plain parallel composition, but cooperate over some action types.

8. Conclusion

In this paper we have studied the relations between exact equivalence [8, 9] and the notion of lumpable bisimulation that has been introduced in [16]. We proved that any exact equivalence over a non-synchronising stochastic automaton is indeed a lumpable bisimulation on the corresponding reversed automaton

S, B, W, R	Original State-space # states	# edges	Timings ms (stdev) Derivation		Solving	
2, 2, 2, 2	396	1412	58.7	(11.80)	184.40	(9.61)
2, 2, 2, 3	1032	3932	73.6	(7.04)	434.40	(63.61)
3, 3, 2, 2	1376	6792	137.3	(60.73)	750.00	(92.08)
2, 2, 3, 3	2064	8540	139.8	(51.09)	686.80	(32.51)
2, 2, 2, 4	2592	10508	125.7	(42.55)	1648.75	(50.51)
3, 3, 2, 3	3920	21244	174.3	(72.37)	7689.60	(60.76)
2, 2, 3, 4	5184	22732	185.0	(68.82)	3450.20	(25.49)
2, 2, 2, 5	6336	27148	215.5	(71.26)	23191.20	(1106.68)
3, 3, 3, 3	7840	45396	239.3	(71.90)	33944.00	(3452.24)
3, 3, 2, 4	10624	62784	364.0	(126.31)	N/A	(N/A)
2, 2, 3, 5	12672	58508	370.9	(143.54)	N/A	(N/A)
2, 2, 2, 6	15168	68236	420.3	(183.38)	N/A	(N/A)
3, 3, 3, 4	21248	133504	471.2	(131.02)	N/A	(N/A)
4, 3, 3, 3	21248	155040	910.1	(224.37)	N/A	(N/A)
3, 3, 2, 5	27776	177280	445.3	(55.69)	N/A	(N/A)
2, 2, 3, 6	30336	146572	490.9	(158.30)	N/A	(N/A)
3, 3, 3, 5	55552	375360	591.4	(60.65)	N/A	(N/A)
4, 3, 3, 4	61472	494692	2274.9	(391.05)	N/A	(N/A)
3, 3, 2, 6	70656	482592	799.30	(163.32)	N/A	(N/A)
3, 3, 3, 6	141312	1018144	1566.6	(531.87)	N/A	(N/A)

Table 2: Profiling results for the derivation of the state space and the steady-state analysis for the web server model, without aggregation. The cells containing "N/A" could not be filled due to memory errors.

and then it induces a strong lumping on the underlying time-reversed CTMC. We show that this fact has important implications not only from a theoretical point of view but also in reducing the computational complexity of the analysis of cooperating models in equilibrium. Indeed, the class of quasi-reversible automata, whose composition is known to be in product-form and hence analysable efficiently, is closed under the exact equivalence. This leads to a new approach for proving the quasi-reversibility of a stochastic component which does not require to study the time-reversed underlying CTMC but to find a model exactly equivalent to the considered one that is already known to be (or to be not) quasi-reversible, or whose quasi-reversibility can be decided easier.

We considered the labeled weighted compatibility problem and proved that both lumpable bisimulation and exact equivalence can be efficiently reduced to it. Then, we proposed an algorithm for the computation of the labeled weighted compatibility problem, that generalizes the one presented in [30] to the many labels case without increasing its time and space complexities. This immediately provides us algorithms for lumpable bisimulation and exact equivalence having the same complexities of the one in [30]. While the case of exact equivalence was not considered in [16], in the case of lumpable bisimulation the new algorithm improves the time complexity of the one in [16].

	Lumped State-space		Timings ms (stdev)			
S, B, W, R	# states	# edges	Derivation		Solving	
2, 2, 2, 2	126	353	254.40	(50.41)	71.80	(12.07)
2, 2, 2, 3	180	503	440.50	(52.18)	79.00	(6.52)
3, 3, 2, 2	240	772	516.10	(31.27)	100.00	(4.47)
2, 2, 3, 3	240	681	681.60	(55.19)	100.60	(6.31)
2, 2, 2, 4	234	653	849.20	(114.30)	117.20	(33.36)
3, 3, 2, 3	360	1156	1226.80	(82.15)	164.80	(13.72)
2, 2, 3, 4	312	885	1472.80	(131.54)	130.80	(12.70)
2, 2, 2, 5	288	803	2020.50	(195.94)	130.40	(19.76)
3, 3, 3, 3	480	1566	2888.70	(349.94)	168.00	(4.90)
3, 3, 2, 4	480	1540	4760.10	(619.80)	272.20	(149.77)
2, 2, 3, 5	384	1089	5681.70	(590.86)	221.40	(120.44)
2, 2, 2, 6	342	953	7519.80	(436.23)	252.60	(215.19)
3, 3, 3, 4	640	2088	16695.10	(309.99)	254.60	(59.81)
4, 3, 3, 3	640	2220	37002.90	(8644.45)	892.20	(255.13)
3, 3, 2, 5	600	1924	25533.20	(2683.20)	210.40	(71.95)
2, 2, 3, 6	456	1293	27366.90	(1332.81)	298.00	(295.22)
3, 3, 3, 5	800	2610	101569.30	(6801.87)	652.20	(853.76)
4, 3, 3, 4	880	3055	310708.00	(25414.64)	779.20	(151.08)
3, 3, 2, 6	720	2308	165280.00	(9526.09)	230.40	(66.10)
3, 3, 3, 6	960	3132	1116392.90	(457681.16)	781.60	(126.67)

Table 3: Profiling results for the derivation of the state space and the steady-state analysis for the web server model, using the contextual-lumpability for aggregation.

References

[1] S. Baarir, M. Beccuti, C. Dutheillet, G. Franceschinis, and S. Haddad. Lumping partially symmetrical stochastic models. *Performance Evaluation*, 68(1):21–44, 2011.

[2] C. Baier, J.-P. Katoen, and H. Hermanns. Comparative branching-time semantics for Markov chains. *Information and Computation*, 200(2):149–214, 2005.

[3] S. Balsamo and A. Marin. *Queueing Networks in Formal methods for performance evaluation*, chapter 2, pages 34–82. M. Bernardo and J. Hillston (Eds), LNCS, Springer, 2007.

[4] S. Balsamo, G. Dei Rossi, and A. Marin. Lumping and reversed processes in cooperating automata. *Annals of Operations Research*, 2014.

[5] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *Journal of the ACM*, 22(2):248–260, 1975.

S, B, W, R	Lumped State-space # states	# edges	Timings ms (stdev) Derivation		Solving	
2, 2, 2, 2	189	609	331.10	(51.07)	111.00	(16.14)
2, 2, 2, 3	278	912	612.80	(60.65)	142.00	(21.01)
3, 3, 2, 2	475	1921	703.30	(60.06)	161.00	(8.43)
2, 2, 3, 3	426	1488	1063.20	(70.94)	155.40	(5.13)
2, 2, 2, 4	385	1304	1505.20	(88.21)	147.60	(15.09)
3, 3, 2, 3	895	3965	2492.00	(85.74)	373.00	(57.76)
2, 2, 3, 4	614	2252	3606.20	(215.40)	257.00	(34.84)
2, 2, 2, 5	528	1870	5158.30	(266.94)	236.00	(37.50)
3, 3, 3, 3	1328	6196	7592.70	(238.72)	584.00	(71.70)
3, 3, 2, 4	1576	7757	12847.70	(179.44)	773.40	(171.75)
2, 2, 3, 5	859	3314	17722.40	(759.65)	382.40	(77.91)
2, 2, 2, 6	713	2664	23980.30	(1701.14)	284.80	(46.90)
3, 3, 3, 4	2441	12737	50078.70	(2979.98)	1835.40	(59.37)
4, 3, 3, 3	6592	45706	113007.20	(21627.30)	102074.00	(7518.95)
3, 3, 2, 5	2629	14241	82905.90	(2790.25)	2527.00	(97.07)
2, 2, 3, 6	1164	4733	94053.60	(3628.03)	388.40	(47.40)
3, 3, 3, 5	6420	42100	329729.90	(22042.39)	35038.40	(340.39)
4, 3, 3, 4	14772	118834	941951.70	(195280.63)	N/A	(N/A)
3, 3, 2, 6	4192	24793	514564.00	(16181.00)	7027.20	(110.80)
3, 3, 3, 6	10849	79667	3634712.90	(1398157.48)	N/A	(N/A)

Table 4: Profiling results for the derivation of the state space and the steady-state analysis for the web server model, using exact-equivalence for aggregation. The cells containing "N/A" could not be filled due to memory errors.

[6] M. Bernardo and R. Gorrieri. A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.

[7] J.T. Bradley, N.J. Dingle, S.T. Gilmore, and W.J. Knottenbelt. Derivation of passage-time densities in PEPA models using ipc: the Imperial PEPA compiler. In *Proc. of the 11th International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'03)*, pages 344–351, 2003.

[8] P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31:59–75, 1994.

[9] P. Buchholz. Exact performance equivalence: An equivalence relation for stochastic automata. *Theoretical Computer Science*, 215(1-2):263–287, 1999.

[10] M. Bugliesi, L. Gallina, S. Hamadou, A. Marin, and S. Rossi. Behavioural equivalences and interference metrics for mobile ad-hoc networks. *Performance Evaluation*, 73:41–72, 2014.

[11] E. Gelenbe and J. M. Fourneau. Flow equivalence and stochastic equivalence in G-Networks. *Computational Management Science*, 1(2):179–192, 2004.

[12] P. G. Harrison. Turning back time in Markovian process algebra. *Theoretical Computer Science*, 290(3):1947–1986, 2003.

[13] H. Hermanns. *Interactive Markov Chains.* Springer, 2002.

[14] H. Hermanns, U. Herzog, and J. P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.

[15] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge Press, 1996.

[16] J. Hillston, A. Marin, C. Piazza, and S. Rossi. Contextual lumpability. In *Proc. of the 7th International Conference on Performance Evaluation Methodologies and Tools (Valuetools'13)*, pages 194–203. ACM Press, 2013.

[17] F. Kelly. *Reversibility and stochastic networks.* Wiley, New York, 1979.

[18] J. G. Kemeny and J. L. Snell. *Finite Markov Chains.* Springer, 1976.

[19] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.

[20] J. Y. Le Boudec. A BCMP extension to multiserver stations with concurrent classes of customers. In *Proc. of the 1986 International Conference on Computer performance modelling, measurement and evaluation (ACM SIGMETRICS'86)*, pages 78–91. ACM Press, 1986.

[21] A. Marin and S. Rossi. Lumping-based equivalences in Markovian automata and applications to product-form analyses. In *Proc. of the 12th International Conference on Quantitative Evaluation of SysTems (QEST'15)*, pages 160–175. Springer, 2015.

[22] A. Marin and S. Rossi. On the relations between lumpability and reversibility. *Acta Informatica*, 2016.

[23] A. Marin and M. G. Vigliotti. A general result for deriving product-form solutions of Markovian models. In *Proc. of First Joint WOSP/SIPEW International Conference on Performance Engineering*, pages 165–176. ACM, 2010.

[24] M. K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transactions on Computers*, 31(9):913–917, 1982.

[25] R. Paige and R. E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[26] B. Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. *SIGMETRICS Performance Evaluation Review*, 13(2):147–154, 1985.

[27] P. Schweitzer. Aggregation methods for large Markov chains. *Mathematical Computer Performance and Reliability*, 1984.

[28] J. Sproston and S. Donatelli. Backward bisimulation in Markov chain model checking. *IEEE Transactions on Software Engineering*, 32(8):531–546, 2006.

[29] U. Sumita and M. Reiders. Lumpability and time-reversibility in the aggregation-disaggregation method for large Markov chains. *Communications in Statistics - Stochastic Models*, 5:63–81, 1989.

[30] A. Valmari and G. Franceschinis. Simple O(m logn) Time Markov Chain Lumping. In *Proc. of 16th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS'10)*, pages 38–52, 2010.

Algorithm 1 Algorithm for Labeled Weighted Compatibility

```
function LWC(G = (S, T, E, W), R)
    P ← R
    skip                                          ▷ 𝒮 ← {S ∪ {s⊥}}
    UB ← P
    TB ← ∅
    w[s] ← unused for every s ∈ S
    while UB ≠ ∅ do
        C ← Pop(UB)
        skip                                      ▷ 𝒮 ← (𝒮 \ {S_C}) ∪ {C, S_C \ C}
        for t ∈ T for s' ∈ C do
            E[s', t] ← (E ↾ t)^{-1}(s')
        for t ∈ T do
            TS ← ∅
            for s' ∈ C for s ∈ E[s', t] do
                if w[s] = unused then
                    TS ← TS ∪ {s}
                    w[s] ← W(s, t, s')
                else w[s] ← w[s] + W(s, t, s')
            for s ∈ TS if w[s] ≠ 0 do
                B ← GetBlockOf(s)
                if B contains 0 marked states then TB ← TB ∪ {B}
                mark s in B
            while TB ≠ ∅ do
                B ← Pop(TB)
                B_1 ← marked states in B
                B ← remaining states in B
                if B = ∅ then give the identity of B to B_1 in P
                else make B_1 a new block in P
                y ← pmc(w[s]) for s ∈ B_1
                B_2 ← {s ∈ B_1 | w[s] ≠ y}
                B_1 ← B_1 \ B_2
                if B_2 = ∅ then ℓ ← 1
                else sort and partition B_2 according to w[s], yielding B_2, ..., B_ℓ
                    make each of B_2, ..., B_ℓ a new block in P
                if B ∈ UB then add B_1, ..., B_ℓ except B in UB
                else add [B,]^? B_1, ..., B_ℓ except largest in UB
            for s ∈ TS do w[s] ← unused
    return P
```