Towards State Space Reduction Based on T-Lumpability-Consistent Relations

Marco Bernardo

Università di Urbino "Carlo Bo" – Italy
Istituto di Scienze e Tecnologie dell'Informazione

Abstract. Markovian behavioral equivalences can be exploited for state space reduction before performance evaluation takes place. It is known that Markovian bisimilarity corresponds to ordinary lumpability and that Markovian testing and trace equivalences correspond to a coarser exact relation we call T-lumpability. While there exists an ordinary-lumpability-consistent aggregation algorithm, this is not the case with T-lumpability. Based on the axiomatization of Markovian testing and trace equivalences, we provide a sufficient condition for T-lumpability that can easily be embedded in the aggregation algorithm for ordinary lumpability, thus enhancing the potential for exact state space reduction. We also identify a class of systems – those providing incremental services – for which the resulting aggregation algorithm turns out to be useful.

1 Introduction

Markovian behavioral equivalences [2] are a formal means to establish whether different models represent systems that behave the same from the functional and performance point of view. For instance, Markovian bisimilarity [6] considers two models to be equivalent whenever they are able to mimic each other's functional and performance behavior step by step. Instead, Markovian testing equivalence [1] considers two models to be equivalent whenever an external observer interacting with them by means of tests is not able to distinguish between them from the functional or performance viewpoint. Finally, Markovian trace equivalence [8] considers two models to be equivalent whenever they are able to execute computations with the same functional and performance characteristics.

These equivalences can be exploited in practice for reducing the state space underlying a model before functional verification and performance evaluation take place. In our Markovian framework, the state space underlying a model represents a continuous-time Markov chain (CTMC). Useful CTMC-level aggregations are those that are exact. Given two CTMCs such that the second one is an exact aggregation of the first one, the transient/stationary probability of being in a macrostate of the second CTMC is the sum of the transient/stationary probabilities of being in one of the constituent microstates of the first CTMC. This means that, when going from the first CTMC to the second CTMC, all the performance characteristics are preserved.

Markovian bisimilarity is consistent with an exact CTMC-level relation that is known under the name of ordinary lumpability [6, 3], whereas Markovian testing and trace equivalences induce a coarser exact CTMC-level relation that we call T-lumpability [1]. Therefore, all the three Markovian behavioral equivalences are equally useful in principle for state space reduction purposes.

However, while there exists a polynomial-time aggregation algorithm for ordinary-lumpability [5], this is not the case with T-lumpability. In fact, while in the case of ordinary-lumpability-based state space reduction the states that can be aggregated are precisely those equivalent to each other, two T-lumpable states that are not ordinarily lumpable cannot be aggregated. This can be seen by looking at the axiomatization of Markovian testing and trace equivalences on a typical Markovian process calculus. Their characterizing laws show that the states that can be aggregated are not the two we are considering, but the ones reachable from them in one transition [1].

The contribution of this paper is to derive a sufficient condition for T-lumpability from the characterizing laws of Markovian testing and trace equivalences, which can be smoothly embedded as a state space preprocessing step in the aggregation algorithm for ordinary lumpability. Besides enhancing the potential for exact state space reduction in practice, we also identify a class of systems for which the resulting aggregation algorithm turns out to be useful. This is the class of systems providing incremental services, i.e. services consisting of sequences of phases such that each service is an extension of another.

This paper is organized as follows. In Sect. 2 we present a motivating example for our work, in which we introduce incremental services. In Sect. 3 we recall the definitions of Markovian bisimilarity, Markovian testing equivalence, and Markovian trace equivalence in a process algebraic framework, so that we can later on exhibit their characterizing equational laws. In Sect. 4 we recall the corresponding exact CTMC-level relations, i.e. ordinary lumpability and T-lumpability. In Sect. 5 we discuss the state space reduction issue by illustrating the characterizing laws of the three Markovian behavioral equivalences. In Sect. 6 we show how to modify the aggregation algorithm for ordinary lumpability in a way that takes into account the characterizing laws of the T-lumpability-consistent Markovian behavioral equivalences. Finally, Sect. 7 contains some concluding remarks.

2 Motivating Example: Incremental Services

Consider a set of services, each of which can be described as a sequence of phases. Then we can define the length of each of those services as the number of its phases and we can say that one of them is a prefix of another one if the phases of the former occur at the beginning of the latter exactly in the same order. We call those services incremental if each of them is a prefix of another one, except for the longest one.

We can distinguish between two ways of using incremental services, which we call eager and lazy, respectively. In the eager case, the client preselects the entire sequence of phases, hence the server knows in advance the specific service to be

provided. By contrast, in the lazy case, the client decides what to do after each phase, so the specific service to be delivered is known only at the end. Whether the behavior of the client is eager or lazy has a remarkable impact on the size of the state space of the incremental service model.

In order to formalize this behavior, we can use a Markovian process calculus consisting of a set $\mathcal{P}$ of process terms built from a set $Name \times \mathbb{R}_{>0}$ of exponentially timed actions and a set of typical operators like the inactive process $\underline{0}$, the action prefix operator $<a, \lambda>.P$, the alternative composition operator governed by the race policy $P_1 + P_2$, the parallel composition operator $P_1 \parallel_S P_2$, and possibly recursive defining equations of the form $B \triangleq P$. The semantics of each process term P is constructed by applying operational rules that yield a state-transition graph $[\![P]\!]$ called labeled multitransition system, whose states correspond to process terms derivable from P and whose transitions are labeled with actions.

As an example of incremental services, consider an Italian restaurant. What can be served is an appetizer, or an appetizer followed by the first course (typically pasta), or an appetizer followed by the first course and the second course (typically meat or fish with some vegetables), or an appetizer followed by the two main courses and some cheese, or an appetizer followed by the two main courses, some cheese and a dessert, or an appetizer followed by the two main courses, some cheese, a dessert and some fruit, or a complete Italian meal (appetizer, first course, second course, cheese, dessert, fruit, and espresso).

Suppose that a client takes $1/\lambda$ time units on average to read the menu, that course i is prepared and served in $1/\mu_i$ time units on average for $1 \leq i \leq 7$, and that subsequence i of the complete meal is selected with probability p_i for $1 \leq i \leq 7$.

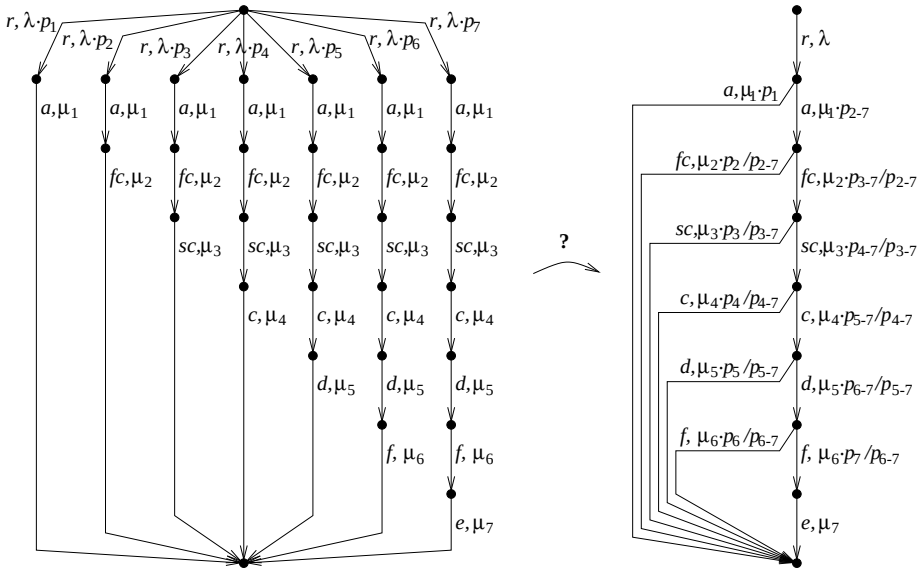
An eager client preselects the desired subsequence of the complete meal, hence the related behavior can be described as follows:

$$
\begin{aligned}
Eager \triangleq\ & <read, \lambda \cdot p_1>.<appetizer, \mu_1>.\underline{0} + \\
& <read, \lambda \cdot p_2>.<appetizer, \mu_1>.<first\_course, \mu_2>.\underline{0} + \\
& <read, \lambda \cdot p_3>.<appetizer, \mu_1>.<first\_course, \mu_2>. \\
& \qquad <second\_course, \mu_3>.\underline{0} + \\
& <read, \lambda \cdot p_4>.<appetizer, \mu_1>.<first\_course, \mu_2>. \\
& \qquad <second\_course, \mu_3>.<cheese, \mu_4>.\underline{0} + \\
& <read, \lambda \cdot p_5>.<appetizer, \mu_1>.<first\_course, \mu_2>. \\
& \qquad <second\_course, \mu_3>.<cheese, \mu_4>. \\
& \qquad <dessert, \mu_5>.\underline{0} + \\
& <read, \lambda \cdot p_6>.<appetizer, \mu_1>.<first\_course, \mu_2>. \\
& \qquad <second\_course, \mu_3>.<cheese, \mu_4>. \\
& \qquad <dessert, \mu_5>.<fruit, \mu_6>.\underline{0} + \\
& <read, \lambda \cdot p_7>.<appetizer, \mu_1>.<first\_course, \mu_2>. \\
& \qquad <second\_course, \mu_3>.<cheese, \mu_4>. \\
& \qquad <dessert, \mu_5>.<fruit, \mu_6>.<espresso, \mu_7>.\underline{0}
\end{aligned}
$$

A lazy client instead decides after each course whether to stop or to proceed with the next course, hence the related behavior can be described as follows where p_{l-r} stands for $\sum_{l \leq i \leq r} p_i$ with $1 \leq l \leq r \leq 7$:

$$
\begin{aligned}
Lazy &\stackrel{\Delta}{=} <read, \lambda>.Lazy_1 \\
Lazy_1 &\stackrel{\Delta}{=} <appetizer, \mu_1 \cdot p_1>.\underline{0} + <appetizer, \mu_1 \cdot p_{2-7}>.Lazy_2 \\
Lazy_2 &\stackrel{\Delta}{=} <first\_course, \mu_2 \cdot \tfrac{p_2}{p_{2-7}}>.\underline{0} + <first\_course, \mu_2 \cdot \tfrac{p_{3-7}}{p_{2-7}}>.Lazy_3 \\
Lazy_3 &\stackrel{\Delta}{=} <second\_course, \mu_3 \cdot \tfrac{p_3}{p_{3-7}}>.\underline{0} + <second\_course, \mu_3 \cdot \tfrac{p_{4-7}}{p_{3-7}}>.Lazy_4 \\
Lazy_4 &\stackrel{\Delta}{=} <cheese, \mu_4 \cdot \tfrac{p_4}{p_{4-7}}>.\underline{0} + <cheese, \mu_4 \cdot \tfrac{p_{5-7}}{p_{4-7}}>.Lazy_5 \\
Lazy_5 &\stackrel{\Delta}{=} <dessert, \mu_5 \cdot \tfrac{p_5}{p_{5-7}}>.\underline{0} + <dessert, \mu_5 \cdot \tfrac{p_{6-7}}{p_{5-7}}>.Lazy_6 \\
Lazy_6 &\stackrel{\Delta}{=} <fruit, \mu_6 \cdot \tfrac{p_6}{p_{6-7}}>.\underline{0} + <fruit, \mu_6 \cdot \tfrac{p_7}{p_{6-7}}>.Lazy_7 \\
Lazy_7 &\stackrel{\Delta}{=} <espresso, \mu_7>.\underline{0}
\end{aligned}
$$

The labeled multitransition systems $[\![Eager]\!]$ and $[\![Lazy]\!]$ are shown below, where every action name is represented through its initials:



As can be noted, $[\![Eager]\!]$ has 30 states and 35 transitions, whereas $[\![Lazy]\!]$ has only 9 states and 14 transitions. It can also be observed that the time taken by the client to consume a meal follows a phase-type distribution in both cases.

While the eager behavior occurs more frequently in practice, the lazy behavior is more convenient when analyzing incremental services due to its reduced state space. In general, if we consider a set of $k \geq 2$ incremental services, its eager description will have $\sum_{1 \leq i \leq k} i = \frac{k \cdot (k+1)}{2} = O(k^2)$ states, whereas its lazy description will have only $O(k)$ states.

Switching from one description to the other – and hence from one phase-type distribution to the other – is feasible only if they are equivalent. If we use

a behavioral equivalence like Markovian bisimilarity, which is highly sensitive to branching points, there is no hope to relate the two descriptions. What we need is thus a less discriminating equivalence. It can be easily shown that the two descriptions are Markovian testing equivalent. But, unfortunately, there is no aggregation algorithm that, consistently with Markovian testing equivalence, would allow us to transform the labeled multitransition system for the eager description into the labeled multitransition system for the lazy description.

3 Markovian Behavioral Equivalences

In this section we recall the definitions of Markovian bisimilarity, Markovian testing equivalence, and Markovian trace equivalence in our process algebraic framework.

3.1 Exit Rates and Computations

Markovian behavioral equivalences are based on concepts like the exit rates of process terms and the traces, the probabilities, and the durations of their computations.

The exit rate of a process term is the rate at which it is possible to leave the state corresponding to the term. We distinguish among (a) the rate at which the process term can execute actions of a given name that lead to a given set of terms, (b) the total rate at which the process term can execute actions of a given name, and (c) the total exit rate of the process term. The latter is the sum of the rates of all the actions that the process term can execute, and coincides with the reciprocal of the average sojourn time in the CTMC state corresponding to the process term.

Definition 1. *Let $P \in \mathcal{P}$, $a \in Name$, and $C \subseteq \mathcal{P}$. The exit rate of P when executing actions of name a that lead to C is defined through the following non-negative real function:*

$$\boxed{rate(P,a,C) \;=\; \sum\!\{\!|\, \lambda \mid \exists P' \in C.\, P \xrightarrow{a,\lambda} P' \,|\!\}}$$

where the summation is taken to be zero whenever the multiset is empty. ■

Definition 2. *Let $P \in \mathcal{P}$. The total exit rate of P is defined through the following non-negative real function:*

$$\boxed{rate_{\mathrm{t}}(P) \;=\; \sum_{a \in Name} rate(P,a,\mathcal{P})}$$

where $rate(P,a,\mathcal{P})$ is the total exit rate of P with respect to a. ■

A computation of a process term is a sequence of transitions that can be executed starting from the state corresponding to the term. The length of a computation is given by the number of transitions occurring in it. We say that

two computations are independent of each other if neither is a proper prefix of the other one. In the following, we denote by $\mathcal{C}_{\rm f}(P)$ and $\mathcal{I}_{\rm f}(P)$ the multisets of finite-length computations and of finite-length independent computations of $P \in \mathcal{P}$, respectively. Below we inductively define the trace, the execution probability, and the stepwise average duration of an element of $\mathcal{C}_{\rm f}(P)$, using symbol "$\circ$" to denote the sequence concatenation operator.

Definition 3. *Let $P \in \mathcal{P}$ and $c \in \mathcal{C}_{\rm f}(P)$. The trace associated with the execution of c is the sequence of action names labeling the transitions of c, which is defined by induction on the length of c through the following $Name^*$-valued function:*

$$\boxed{trace(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ a \circ trace(c') & \text{if } c \equiv P \stackrel{a,\lambda}{\longrightarrow} c' \end{cases}}$$

where ε is the empty trace. ■

Definition 4. *Let $P \in \mathcal{P}$ and $c \in \mathcal{C}_{\rm f}(P)$. The probability of executing c is the product of the execution probabilities of the transitions of c, which is defined by induction on the length of c through the following $\mathbb{R}_{]0,1]}$-valued function:*

$$\boxed{prob(c) = \begin{cases} 1 & \text{if } length(c) = 0 \\ \frac{\lambda}{rate_{\rm t}(P)} \cdot prob(c') & \text{if } c \equiv P \stackrel{a,\lambda}{\longrightarrow} c' \end{cases}}$$

We also define the probability of executing a computation of C as:

$$\boxed{prob(C) = \sum_{c \in C} prob(c)}$$

for all $C \subseteq \mathcal{I}_{\rm f}(P)$. ■

Note that $prob(C)$ would not be well defined if set C contained computations that are not indepedent of each other.

Definition 5. *Let $P \in \mathcal{P}$ and $c \in \mathcal{C}_{\rm f}(P)$. The stepwise average duration of c is the sequence of the average sojourn times in the states traversed by c, which is defined by induction on the length of c through the following $(\mathbb{R}_{>0})^*$-valued function:*

$$\boxed{time(c) = \begin{cases} \varepsilon & \text{if } length(c) = 0 \\ \frac{1}{rate_{\rm t}(P)} \circ time(c') & \text{if } c \equiv P \stackrel{a,\lambda}{\longrightarrow} c' \end{cases}}$$

where ε is the empty stepwise average duration. We also define the multiset of computations of C whose stepwise average duration is not greater than θ as:

$$\boxed{C_{\leq \theta} = \{\!\!\{\, c \in C \mid length(c) \leq length(\theta) \wedge \forall i = 1, \ldots, length(c).\, time(c)[i] \leq \theta[i] \,\}\!\!\}}$$

for all $C \subseteq \mathcal{C}_{\rm f}(P)$ and $\theta \in (\mathbb{R}_{>0})^*$. ■

The reason why we consider the stepwise average duration instead of the standard average duration (intended as the sum of the average sojourn times in the traversed states) is explained in [1].

3.2 Markovian Bisimilarity

Markovian bisimilarity [6] considers two process terms to be equivalent whenever they are able to mimic each other's functional and performance behavior step by step.

Whenever a process term can perform actions with a certain name that reach a certain set of terms at a certain speed, then any process term equivalent to the given one has to be able to respond with actions with the same name that reach an equivalent set of terms at the same speed. This can be formalized through the comparison of the process term exit rates.

Definition 6. *An equivalence relation $\mathcal{B} \subseteq \mathcal{P} \times \mathcal{P}$ is a Markovian bisimulation iff, whenever $(P_1, P_2) \in \mathcal{B}$, then for all action names $a \in Name$ and equivalence classes $C \in \mathcal{P}/\mathcal{B}$:*

$$rate(P_1, a, C) = rate(P_2, a, C)$$

■

Since the union of all the Markovian bisimulations can be proved to be the largest Markovian bisimulation, the definition below follows.

Definition 7. *Markovian bisimilarity, denoted by $\sim_{\mathrm{MB}}$, is the union of all the Markovian bisimulations.* ■

3.3 Markovian Testing Equivalence

Markovian testing equivalence [1] considers two process terms to be equivalent whenever an external observer interacting with them by means of tests is not able to distinguish between them from the functional or performance viewpoint.

A test can be represented through another process term, which interacts with the term to be tested by means of a parallel composition operator that enforces synchronization on all action names. Since a test should be conducted in a finite amount of time, for the test formalization we restrict ourselves to non-recursive, finite-state process terms. In our Markovian framework, tests are made out of passive actions, each equipped with a weight $w \in \mathbb{R}_{>0}$. The idea is that, in any of its states, a process term to be tested probabilistically generates the proposal of an action to be executed among those enabled in that state, then the test reacts by probabilistically selecting a passive action (if any) with the same name as the proposed action.

Definition 8. *The set $\mathcal{T}$ of tests is generated by the following syntax:*

$$\boxed{\begin{array}{l} T ::= \mathrm{s} \mid T' \\ T' ::= \langle a, *_w \rangle.T \mid T' + T' \end{array}}$$

where $a \in Name$, $w \in \mathbb{R}_{>0}$, and s *stands for success.* ■

Let us denote by $\longrightarrow_{\mathrm{T}}$ the transition relation for tests. The following operational rule defines the interaction of $P \in \mathcal{P}$ and $T \in \mathcal{T}$:

$$\frac{P \xrightarrow{a,\lambda} P' \qquad T \xrightarrow{a,*_w}_{\mathrm{T}} T'}{P \parallel T \xrightarrow{a,\lambda \cdot \frac{w}{weight(T,a)}} P' \parallel T'}$$

where $weight(T,a) = \sum\{\!|\, w \mid \exists T'.\, T \xrightarrow{a,*_w}_{\mathrm{T}} T' \,|\!\}$ is the weight of T with respect to a.

Definition 9. *Let $P \in \mathcal{P}$ and $T \in \mathcal{T}$. The interaction system of P and T is process term $P \parallel T$, where we say that:*

- *A configuration is a state of the labeled multitransition system underlying $P \parallel T$.*
- *A configuration is successful iff its test component is s.*
- *A computation is successful iff so is the last configuration it reaches.*

We denote by $\mathcal{SC}(P,T)$ the multiset of successful computations of $\mathcal{C}_{\mathrm{f}}(P \parallel T)$. ■

Note that $\mathcal{SC}(P,T) \subseteq \mathcal{I}_{\mathrm{f}}(P \parallel T)$, because of the maximality of the successful test-driven computations, and that $\mathcal{SC}(P,T)$ is finite, because of the finitely-branching structure of the considered terms.

Markovian testing equivalence relies on comparing the process term probabilities of performing a successful test-driven computation within a given sequence of average amounts of time.

Definition 10. *Let $P_1, P_2 \in \mathcal{P}$. We say that P_1 is Markovian testing equivalent to P_2, written $P_1 \sim_{\mathrm{MT}} P_2$, iff for all tests $T \in \mathcal{T}$ and sequences $\theta \in (\mathbf{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{SC}_{\leq\theta}(P_1,T)) = prob(\mathcal{SC}_{\leq\theta}(P_2,T))$$

■

3.4 Markovian Trace Equivalence

Markovian trace equivalence [8] considers two process terms to be equivalent whenever they are able to execute computations with the same functional and performance characteristics.

It relies on comparing the process term probabilities of performing a computation within a given sequence of average amounts of time.

Definition 11. *Let $P \in \mathcal{P}$, $c \in \mathcal{C}_{\mathrm{f}}(P)$, and $\alpha \in Name^*$. We say that c is compatible with α iff:*

$$trace(c) = \alpha$$

We denote by $\mathcal{CC}(P,\alpha)$ the multiset of finite-length computations of P that are compatible with α. ■

Note that $\mathcal{CC}(P,\alpha) \subseteq \mathcal{I}_{\rm f}(P)$, because of the compatibility of the computations with the same trace α, and that $\mathcal{CC}(P,\alpha)$ is finite, because of the finitely-branching structure of the considered terms.

Definition 12. *Let $P_1, P_2 \in \mathcal{P}$. We say that P_1 is Markovian trace equivalent to P_2, written $P_1 \sim_{\rm MTr} P_2$, iff for all traces $\alpha \in Name^*$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{CC}_{\leq\theta}(P_1,\alpha)) = prob(\mathcal{CC}_{\leq\theta}(P_2,\alpha))$$

■

4 Induced CTMC-Level Relations

All of the three Markovian behavioral equivalences recalled in the previous section induce CTMC-level relations. In order to ease the formalization of these relations, we take an arbitrary CTMC with state space S, whose transitions are labeled not only with rates but also with the same action name a. In other words, a CTMC is viewed as a labeled transition system in which the label set is $\{a\} \times \mathbb{R}_{>0}$. This allows us to define the induced CTMC-level relations as special cases of the considered Markovian behavioral equivalences.

Markovian bisimilarity is consistent with a CTMC-level relation known under the name of ordinary lumpability [6, 3].

Definition 13. *Let $s_1, s_2 \in S$. We say that s_1 and s_2 are ordinarily lumpable iff there exists a partition $\mathcal{O}$ of S such that s_1 and s_2 belong to the same equivalence class and, whenever $z_1, z_2 \in O$ for some $O \in \mathcal{O}$, then for all $O' \in \mathcal{O}$:*

$$\sum_{z' \in O'} \{\!|\, \lambda \mid z_1 \xrightarrow{a,\lambda} z' \,|\!\} = \sum_{z' \in O'} \{\!|\, \lambda \mid z_2 \xrightarrow{a,\lambda} z' \,|\!\}$$

■

Markovian testing and trace equivalences induce a coarser CTMC-level relation that we call T-lumpability [1].

Definition 14. *Let $s_1, s_2 \in S$. We say that s_1 and s_2 are T-lumpable iff for all tests $T \in \mathcal{T}$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{SC}_{\leq\theta}(s_1,T)) = prob(\mathcal{SC}_{\leq\theta}(s_2,T))$$

■

Since the transitions of the CTMC are all labeled with the same action name, the branching structure of tests adds no distinguishing power with respect to traces. Thus, the definition above is equivalent to the one below.

Definition 15. *Let $s_1, s_2 \in S$. We say that s_1 and s_2 are T-lumpable iff for all traces $\alpha \in Name^*$ and sequences $\theta \in (\mathbb{R}_{>0})^*$ of average amounts of time:*

$$prob(\mathcal{CC}_{\leq\theta}(s_1,\alpha)) = prob(\mathcal{CC}_{\leq\theta}(s_2,\alpha))$$

■

5 An Axiom-Based View of State Space Reduction

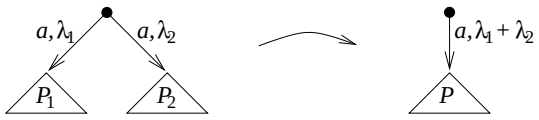
According to Markovian bisimilarity (resp. ordinary lumpability), two equivalent states can reach sets of equivalent states by performing transitions with the same names and the same total rates. This causes equivalent states of a labeled multitransition system (resp. CTMC) to be precisely the states that can be aggregated. In other words, the partition induced by the equivalence relation, whose elements correspond to equivalence classes, can be directly exploited for state space reduction purposes. This can easily be seen through the equational law characterizing $\sim_{\rm MB}$, which establishes that:

$$\boxed{<a,\lambda_1>.P_1 + <a,\lambda_2>.P_2 \sim_{\rm MB} <a,\lambda_1+\lambda_2>.P}$$

whenever:

$$\boxed{P_1 \sim_{\rm MB} P \sim_{\rm MB} P_2}$$

Its effect at the state space reduction level is shown below:



From $\sim_{\rm MB} \subset \sim_{\rm MT} \subset \sim_{\rm MTr}$ we derive that more states can be related to each other when using Markovian testing and trace equivalences (resp. T-lumpability). However, these additional equivalent states must be carefully treated at state space reduction time. The reason is that they cannot be aggregated. What turns out is that the states they reach – which are not necessarily equivalent to each other – can be sometimes aggregated. This is clearly shown by the characterizing law for $\sim_{\rm MT}$ – which subsumes the one for $\sim_{\rm MB}$ – and the characterizing law for $\sim_{\rm MTr}$ – which subsumes the one for $\sim_{\rm MT}$ [1].

The first characterizing law establishes that $\sim_{\rm MT}$ allows choices to be deferred as long as they are related to branches starting with actions having the same name – as in the $\sim_{\rm MB}$ case – that are immediately followed by actions having the same names and the same total rates in all the branches. Formally:

$$\boxed{\begin{array}{l}\displaystyle\sum_{i\in I} <a,\lambda_i>.\sum_{j\in J_i} <b_{i,j},\mu_{i,j}>.P_{i,j} \;\sim_{\rm MT} \\ \qquad <a,\displaystyle\sum_{k\in I}\lambda_k>.\sum_{i\in I}\sum_{j\in J_i} <b_{i,j},\frac{\lambda_i}{\sum_{k\in I}\lambda_k}\cdot\mu_{i,j}>.P_{i,j}\end{array}}$$

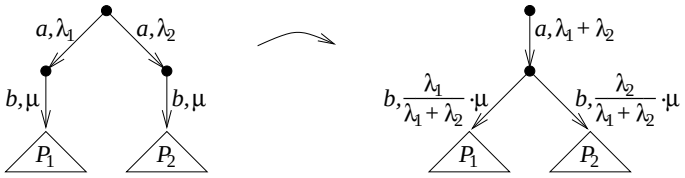
whenever:

- I is a finite index set with $|I| \geq 2$.
- J_i is a finite index set for all $i \in I$, with the related summation being $\underline{0}$ whenever $J_i = \emptyset$.
- For all $i_1, i_2 \in I$ and $b \in Name$:

$$\boxed{\sum_{j\in J_{i_1}, b_{i_1,j}=b} \mu_{i_1,j} = \sum_{j\in J_{i_2}, b_{i_2,j}=b} \mu_{i_2,j}}$$

with each summation being zero whenever its index set is empty.

The effect at the state space reduction level of the simplest instance of the law above is shown below:



where the two initial states are related by $\sim_{\mathrm{MT}}$ but not by $\sim_{\mathrm{MB}}$ (unless $P_1 \sim_{\mathrm{MB}} P_2$), while the two states reached by the initial state on the left are aggregated on the right although they are not necessarily equivalent in any sense.

The second characterizing law establishes that $\sim_{\mathrm{MTr}}$ allows choices to be deferred as long as they are related to branches starting with actions having the same name – as in the $\sim_{\mathrm{MB}}$ and $\sim_{\mathrm{MT}}$ cases – that are followed by terms having the same total exit rate. Note that – unlike the $\sim_{\mathrm{MT}}$ case – the names and the total rates of the initial actions of such derivative terms can be different in the various branches. Formally:

$$\sum_{i \in I} \langle a, \lambda_i \rangle . \sum_{j \in J_i} \langle b_{i,j}, \mu_{i,j} \rangle . P_{i,j} \;\; \sim_{\mathrm{MTr}} \;\; \langle a, \sum_{k \in I} \lambda_k \rangle . \sum_{i \in I} \sum_{j \in J_i} \langle b_{i,j}, \frac{\lambda_i}{\Sigma_{k \in I} \lambda_k} \cdot \mu_{i,j} \rangle . P_{i,j}$$

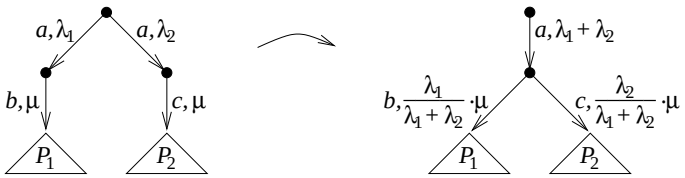
whenever:

- I is a finite index set with $|I| \geq 2$.
- J_i is a finite index set for all $i \in I$, with the related summation being $\underline{0}$ whenever $J_i = \emptyset$.
- For all $i_1, i_2 \in I$:

$$\sum_{j \in J_{i_1}} \mu_{i_1,j} = \sum_{j \in J_{i_2}} \mu_{i_2,j}$$

  with each summation being zero whenever its index set is empty.

The effect at the state space reduction level of the simplest instance of the law above is shown below:



where the two initial states are related by $\sim_{\mathrm{MTr}}$ but not by $\sim_{\mathrm{MT}}$ (unless $b = c$). As before, the two states reached by the initial state on the left are aggregated on the right although they are not necessarily equivalent in any sense.

It is worth observing that the two laws characterizing $\sim_{\mathrm{MT}}$ and $\sim_{\mathrm{MTr}}$, respectively, differ only for functional details, whereas they are identical from the performance point of view – consistently with the fact that both $\sim_{\mathrm{MT}}$ and $\sim_{\mathrm{MTr}}$ rely on the same CTMC-level relation.

6 Aggregation Algorithm for T-Lumpability

For Markovian bisimilarity (resp. ordinary lumpability) it is well known that a partition refinement algorithm in the style of [7] can be used, which exploits the fact that the equivalence relation can be characterized as a fixed point of successively finer relations. In fact, for $\sim_{\rm MB}$ we have:

$$\sim_{\rm MB} = \bigcap_{i \in \mathbb{N}} \sim_{{\rm MB},i}$$

where $\sim_{\rm MB,0} = \mathcal{P} \times \mathcal{P}$ and $\sim_{{\rm MB},i}$ is defined as follows for $i \geq 1$: whenever $(P_1, P_2) \in \sim_{{\rm MB},i}$, then for all $a \in \mathit{Name}$ and $C \in \mathcal{P}/\sim_{{\rm MB},i-1}$:

$$\mathit{rate}(P_1, a, C) = \mathit{rate}(P_2, a, C)$$

In other words, $\sim_{\rm MB,0}$ induces a trivial partition with a single equivalence class that coincides with $\mathcal{P}$, $\sim_{\rm MB,1}$ refines the previous partition by creating an equivalence class for each set of terms that possess the same total exit rates with respect to the same action names, and so on.

The minimization algorithm for $\sim_{\rm MB}$ (resp. ordinary lumpability) proceeds as follows:

1. Build the initial partition with a single class including all the states, then initialize a list of splitters with this class.
2. Refine the current partition by splitting each of its classes according to the exit rates towards one of the splitters, then remove this splitter from the list.
3. For each split class, insert into the list of splitters all the resulting subclasses except for the largest one.
4. If the list of splitters is not empty, go back to the refinement step.

The time complexity is $O(m \cdot \log n)$, where n is the number of states and m is the number of transitions. In order to achieve this complexity, it is necessary to resort to a splay tree when representing the subclasses arising from the splitting of a class [5].

Since $\sim_{\rm MT}$ and $\sim_{\rm MTr}$ (resp. T-lumpability) are coarser than $\sim_{\rm MB}$ (resp. ordinary lumpability), the algorithm above can be used as a basis to construct an aggregation algorithm for T-lumpability-based relations. The idea is to include a preprocessing step at the beginning of the algorithm, which rewrites the state space in a way that the additional states that can be aggregated as discussed in Sect. 5 are actually aggregated before the partition refinement process starts:

0. Preprocess the state space by aggregating the states reachable by a state that satisfies the conditions associated with the law characterizing $\sim_{\rm MT}$ or $\sim_{\rm MTr}$.

More precisely, two states s_1, s_2 can be aggregated in the above preprocessing step if the following four constraints are satisfied:

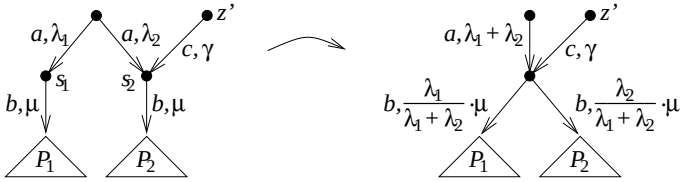
(i) There exists at least one state z with outgoing transitions reaching both s_1 and s_2 that are labeled with the same action name.
(ii) s_1 and s_2 have the same total exit rate (in the $\sim_{\rm MT}$ case, this constraint must hold with respect to individual action names).

(*iii*) There is no state z' with outgoing transitions reaching only one between s_1 and s_2.

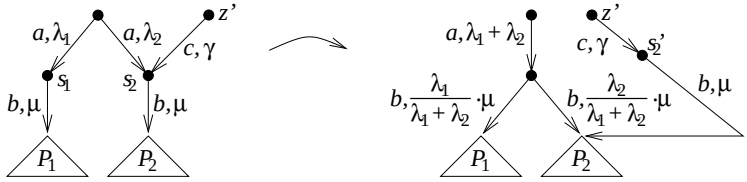
(*iv*) Given any two states z_1, z_2 with outgoing transitions reaching both s_1 and s_2, for $i = 1, 2$ the probability with which z_1 reaches s_i is equal to the probability with which z_2 reaches s_i (in the $\sim_{\mathrm{MT}}$ case, this constraint must hold with respect to individual action names).

Constraints (*i*) and (*ii*) are straightforward consequences of the characterizing laws, whereas constraints (*iii*) and (*iv*) describe the context in which T-lumpability-consistent aggregations can safely take place.

Constraint (*iii*) avoids aggregations like the following:

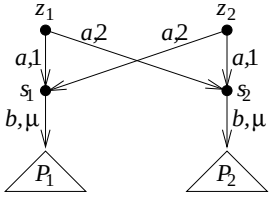


where z' would be enabled to reach P_1. We observe that in this scenario a correct aggregation would be the following:



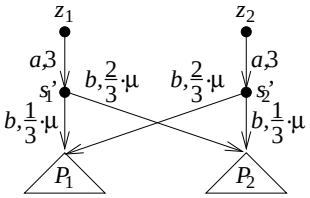
This requires a duplicate s_2' of s_2 – which in process algebraic terms could be formalized as $s_2 + \underline{0}$ – hence it may be in contrast with our objective of reducing the state space. In general, the suitability of duplicating states depends on the difference between the numbers of states before and after the duplication/aggregation process, which is zero in the example above.

Constraint (*iv*) avoids aggregations in situations like the following:



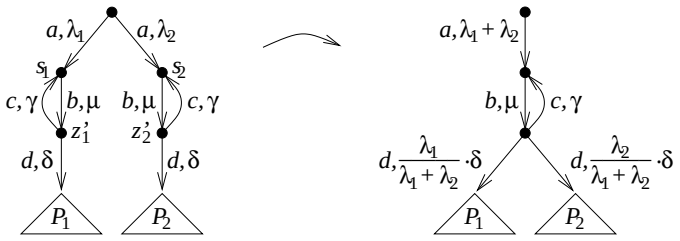
Here the problem is that the new state resulting from the aggregation of s_1 and s_2 should reach P_1 with probability $1/3$ and P_2 with probability $2/3$ when coming from z_1, while it should reach P_1 with probability $2/3$ and P_2 with probability $1/3$ when coming from z_2. This is not possible in a memoryless stochastic model like a CTMC. It could only be achieved by duplicating both s_1 and s_2 in such a way that z_1 reaches $s_{1,1}$ and $s_{1,2}$ while z_2 reaches $s_{2,1}$ and $s_{2,2}$. At that point

$s_{1,1}$ and $s_{1,2}$ could be aggregated into s'_1 and $s_{2,1}$ and $s_{2,2}$ could be aggregated into s'_2, as shown below:



Also in this case duplication may be in contrast with our objective of reducing the state space, hence the same remark as for the previous constraint applies.

From the point of view of singling out states that can be aggregated according to T-lumpability but not according to ordinary lumpability, constraints (i) and (ii) are strictly necessary as they are connected to the characterizing laws. Likewise, constraint (iv) is strictly necessary as it guarantees that the stochastic model obtained after the aggregation process is still a CTMC. By contrast, constraint (iii) expresses a sufficient but not necessary condition. Consider for instance the following scenario:



Constraint (iii) is violated because there is a state (z'_1) reaching only s_1 and another state (z'_2) reaching only s_2. Nevertheless, the aggregation shown above is consistent with T-lumpability.

The preprocessing step to be applied before the refinement process of the minimization algorithm for $\sim_{\mathrm{MB}}$ (resp. ordinary lumpability) can be implemented through a depth-first visit of the state space, provided that both the reachability relation and its inverse relation are represented in the labeled multitransition system to be reduced. In other words, every state has to encode not only its outgoing transitions, but also its incoming transitions. During the visit, for each state we check whether some of its derivative states satisfy constraints (i) and (ii). If this is the case, we check whether constraints (iii) and (iv) are satisfied as well by those derivative states (here inverse reachability is needed). If so, those derivative states are aggregated into a new state according to the characterizing law for the equivalence relation of interest ($\sim_{\mathrm{MT}}$ or $\sim_{\mathrm{MTr}}$). The time complexity of the visit is $O(n + m)$, where n is the number of states and m is the number of transitions of the labeled multitransition system.

If we go back to the incremental service example of Sect. 2, we see that the preprocessing step trasforms $[\![\mathit{Eager}]\!]$ into $[\![\mathit{Lazy}]\!]$, thus achieving a quadratic reduction of the state space size. As usual, the other steps of the resulting ag-

gregation algorithm come into play in case of symmetries. Suppose that at the Italian restaurants there are $h \geq 2$ independent clients, each identical to the eager client. Then the description of the whole set of clients will have $O(\alpha^h)$ states for some $\alpha > 1$, while the aggregated state space will have $O(h)$ states at the end of the partition refinement process.

In the case of identical clients using incremental services, we observe that the T-lumpability-consistent aggregation algorithm has not only the effect of further reducing the size of the state space with respect to the ordinary-lumpability-consistent minimization algorithm. In fact, if applied to each client individually before composing the clients in parallel, it speeds up the execution of the last application of the algorithm itself to the whole system.

7 Conclusion

We have exploited the characterizing laws of Markovian testing and trace equivalences to enhance the potential for exact state space reduction of ordinary lumpability. Moreover, we have singled out a class of systems – those providing incremental services – to which the resulting algorithm can be profitably applied.

Concerning future work, we would like to develop some heuristics that help deciding whether and when it is convenient to perform local state duplications in order to satisfy constraints (*iii*) and (*iv*) in situations like those in Sect. 6.

Furthermore, we would like to investigate whether the algorithm for verifying classical testing equivalence proposed in [4] – which reduces testing equivalence verification over labeled transition systems to the verification of a generalization of bisimilarity over acceptance graphs – can be of help in order to strengthen our T-lumpability-consistent aggregation algorithm.

References

1. M. Bernardo, *"Non-Bisimulation-Based Markovian Behavioral Equivalences"*, in Journal of Logic and Algebraic Programming 72:3-49, 2007.
2. M. Bernardo, *"A Survey of Markovian Behavioral Equivalences"*, in *Formal Methods for Performance Evaluation*, LNCS 4486:180-219, 2007.
3. P. Buchholz, *"Exact and Ordinary Lumpability in Finite Markov Chains"*, in Journal of Applied Probability 31:59-75, 1994.
4. R. Cleaveland and M. Hennessy, *"Testing Equivalence as a Bisimulation Equivalence"*, in Formal Aspects of Computing 5:1-20, 1993.
5. S. Derisavi, H. Hermanns, and W.H. Sanders, *"Optimal State-Space Lumping in Markov Chains"*, in Information Processing Letters 87:309-315, 2003.
6. J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996.
7. R. Paige and R.E. Tarjan, *"Three Partition Refinement Algorithms"*, in SIAM Journal on Computing 16:973-989, 1987.
8. V. Wolf, C. Baier, and M. Majster-Cederbaum, *"Trace Machines for Observing Continuous-Time Markov Chains"*, in Proc. of the *3rd Int. Workshop on Quantitative Aspects of Programming Languages (QAPL 2005)*, ENTCS 153(2):259-277, Edinburgh (UK), 2005.