Solving Systems of Bilinear Equations for Transition Rate Reconstruction

Amin Soltanieh, Markus Siegle

Solving Systems of Bilinear Equations for Transition Rate Reconstruction

Amin Soltanieh and Markus Siegle

Universität der Bundeswehr München, Neubiberg 85579, Germany
amin.soltanieh@unibw.de
markus.siegle@unibw.de

Abstract. Compositional models, specified with the help of a Markovian Stochastic Process Algebra (SPA), are widely used in performance and dependability modelling. The paper considers the problem of transition rate reconstruction: Given two SPA components with unknown rates, and given their combined flat model with fixed rates, the task is to reconstruct the rates in the components. This problem occurs frequently during so-called model repair, if a certain subset of transition rates of the flat model needs to be changed in order to satisfy some given requirement. It is important to have a structured approach to decide whether or not the rate reconstruction, satisfying the desired low-level model changes, is possible or not. In order to realize such a reconstruction, every combined model transition is transformed into an equation, resulting – for each action type – in a system of bilinear equations. If the system of equations meets a consistency condition, rate reconstruction is indeed possible. We identify a class of SPA systems for which solving the system of equations is not necessary, since by checking a set of simple conditions we can check the consistency of the system of equations. Furthermore, for general models outside this class, an iterative algorithm for solving the system of equations efficiently is proposed.

Keywords: Stochastic Process Algebra, Bilinear System of Equations, Model Reconstruction.

1 Introduction

Stochastic Process Algebras (SPA) such as PEPA [11], EMPA [4] and CASPA [14] are often used in performance and dependability modeling, since they allow users to define complex systems in a modular and hierarchical way as interacting components. For checking system requirements in a model, probabilistic model checking, implemented in tools like PRISM [15], MRMC [13], iscasMC [8] and STORM [6], is used. Model checking usually takes place at the level of the combined model, where states are labelled with atomic propositions, and transitions are labelled with action names and rates.

In this paper, an interesting question is addressed: Given a compositional SPA model, albeit with unknown transition rates, and given the transition rates

of the associated low-level flat Markov chain, how and under which conditions can we reconstruct the transition rates of the high-level model, i.e. of the SPA components? In other words, how can we lift the combined model rate information to the high-level model? This is an important question, since users usually prefer not to work with the (large) flat model of the system, but instead work with the much smaller CTMCs of the components.

This question arises frequently in the field of model repair [2]. That is, if a property is not satisfied in a model, we may apply some modifications to the model such that the modified (repaired) model satisfies the property [3,5]. Earlier works on model repair, based on transition rate modification, has been published in [20] and [19] for different time-bounded and time-unbounded reachability properties expressed in the logic CSL [1]. However, all these researches address model repair solution at the level of the flat, combined model of the system. Whenever, for the sake of a property fulfillment, some transition rates in the combined model of the system are supposed to change, we need to know if reconstructing the components accordingly is feasible or not. Recently, in [17], a lifting algorithm has been introduced which receives the combined model rate modifications and lifts this information to the compositional high-level model, if such a lifting is possible at all. At the core of this algorithm is the construction and solution of a system of bilinear equations.

With a simple example, inspired by [18] and developed in [17], the rate reconstruction problem is explained: In Figure 1, two components, A and B, are synchronised over action a and the resultant combined model is shown. Every transition is labelled by the tuple (act, r), where act is the action name and r is the transition rate. Assume that the combined model transition rate values

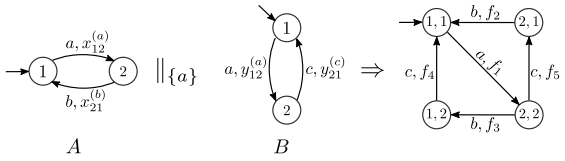

Fig. 1: Processes A and B and the resultant combined model

$\{f_1, \ldots, f_5\}$ are given and all transition rates of the components are unknown variables $(x_{12}^{(a)}, x_{21}^{(b)}, y_{12}^{(a)}, y_{21}^{(c)})$. The transition rates in the combined model are functions of the transition rates in the components, where for action synchronisation in SPA, different semantics have been discussed and compared in the past concerning the rate of the combined transition rate [10]. In this paper, we assume that the rate of a combined transition is calculated as the product of the rates of the transitions to be synchronised, similar to the synchronisation of CTMC modules in PRISM.

For reconstructing the rates of the components, every action type should be considered independently:

Action a**:** Action a is a synchronising action and there is only one a-transition in the combined model. So, the only equation is $x_{12}^{(a)} \cdot y_{12}^{(a)} = f_1$, and obviously, for any arbitrary value of f_1, it is always possible to choose $x_{12}^{(a)}$ freely and then set $y_{12}^{(a)}$ accordingly.
Action b**:** Assume $f_2 = f_3 = f$, then *local lifting* is possible, i.e. lifting the combined model transition rates to the high-level model. In this case, the only equation is $x_{21}^{(b)} = f$. In general, local lifting means that reconstruction of combined model transition rates is possible by assigning some transition rates in only one of the components.
Action c**:** If $f_4 = f_5$, then we have the same case as for action type b above. But let us now consider the case $f_4 \neq f_5$. The two c-transitions in the combined model stem from the only c-transition in component B, and if this transition rate is changed, both c-transitions in the combined model are equally affected. The system of equations includes two contradicting equations: $y_{21}^{(c)} = f_4$ and $y_{21}^{(c)} = f_5$. So, in this example, we cannot reconstruct the rates in the high-level components in a straight-forward manner, because there exists a context dependency for the transition rates f_4 and f_5.

However, as a solution for this problem, it has been observed in [17] that we can implement this context dependency by adding action c to the synchronisation set and inserting c-selfloops at the states of component A with unknown transition rates $x_{11}^{(c)}$ and $x_{22}^{(c)}$, as shown in Figure 2. The modified components are denoted A' and B'. In order to find the unknown rates $x_{11}^{(c)}$, $x_{22}^{(c)}$ and $y_{21}^{(c)}$, we

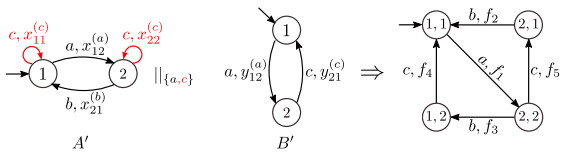

Fig. 2: Processes A' and B' with added selfloops and modified synchronising set

can transform the combined model c-transitions into the mathematical equations $x_{11}^{(c)} \cdot y_{21}^{(c)} = f_4$ and $x_{22}^{(c)} \cdot y_{21}^{(c)} = f_5$. For any pair of values f_4 and f_5, this system of equations has a solution, so reconstructing the high-level models is now possible! One possible solution is $x_{11}^{(c)} = f_4$, $x_{22}^{(c)} = f_5$ and $y_{21}^{(c)} = 1$. Note, however, that in general, the system of equations may or may not have a solution.

It is always possible to form a system of equations, representing the combined model transitions with regard to a specific action type, if the rates of the flat Markov chain are given. This system of equations, for a combined system of two interacting components, always has a specific bilinear form. Solving this system of equations efficiently can be challenging, especially for systems with very large state space. In this report, we identify a class of systems where without solving the system of equations and only by checking some simple conditions,

the consistency of the system of equations can be checked and a possible solution can be obtained. For general systems outside this class, an iterative algorithm is introduced which can solve the system of equations efficiently.

2 Background

A bilinear equation is one in which the variables may be partitioned into two disjoint subsets such that the left hand side is linear in each set separately and the right hand side is a given scalar. The system of l bilinear equations can be shown in the following matrix form:

$$x^T A_i y = d_i, \qquad i = 1, 2, \ldots, l \tag{1}$$

where x and y are two disjoint variable vectors of length n and m, respectively, and $A_i \in \mathbb{R}^{n \times m}$.

In [12], the solution theory is provided for *complete* bilinear systems of equations for the case that all A_i matrices are linearly independent. A bilinear system of equations is complete if the number of equations is the product of the number of x and y variables ($l = n \cdot m$). It is proved that the consistency and the solution of the complete bilinear system of equations can be obtained using matrix $G \in \mathbb{R}^{n \times m}$ from the following equation:

$$\mathrm{vec}(G) = (A^{-1})^T d \tag{2}$$

where vec is the vectorization operator which transforms a matrix into a column vector by stacking the columns of the matrix on top of one another, and A is a matrix with columns $\mathrm{vec}(A_i)$. If the rank of matrix G is one, then the system of equations is consistent and has a solution.

In the general form of a bilinear system of equations (1), every equation may include the sum of several bilinear terms. But in our situation, which is the system of two interacting components, we have a simpler case where every equation has only one bilinear term. In this case, it is always possible to sort the equations such that matrix A becomes the identity matrix. Then $\mathrm{vec}(G) = d$, which leads to the result obtained in this paper in Lemma 1.

In Sect. 3, we derive some conditions which occur frequently in models with synchronising or non-synchronising actions. We will see that by checking such a set of simple conditions, we can decide whether a solution exists or not, and if a solution exists we can obtain one. If these conditions are not satisfied, we need to solve the system of equations using other methods.

3 Special Cases Leading to Complete System

In this section, we will see that under some conditions, we can verify the consistency of the system of equations by checking a simple condition and find a possible solution. We divide this section into two parts, synchronising and non-synchronising actions, i.e. whether the system of equations corresponds to a

synchronising action or a non-synchronising one. At first, we define the framework and notation for the system and then in two lemmas the *complete* systems wrt a synchronising or non-synchronising action are defined. In the rest of this report, the action type is omitted from the variable notation, since it is implicit in the context.

Definition 1 (System Definition).

Let $Sys = M_1 \parallel_{\Sigma_s} M_2$, where M_1 and M_2 are two Markovian transition systems with state spaces S_1 and S_2 , and $\Sigma_s \subseteq Act$ is the set of synchronising actions. $S \subseteq S_1 \times S_2$ denotes the reachable state space of the combined model Sys (reachable from the specified initial states in M_1 and M_2).

For action $c \in Act$, $en_1 = \{a_1, \ldots, a_p\} \subseteq S_1$ and $en_2 = \{b_1, \ldots, b_r\} \subseteq S_2$ are the states in M_1 and M_2 where action c is enabled. Let $ne_1 = S_1 \setminus en_1 = \{a_{p+1}, \ldots, a_{p+q}\}$ and $ne_2 = S_2 \setminus en_2 = \{b_{r+1}, \ldots, b_{r+s}\}$ be the sets of states where action c is not enabled, so $|S_1| = p + q$ and $|S_2| = r + s$.

k_i and l_j are the number of outgoing c-transitions in states a_i and b_j, where $1 \le i \le p$ and $1 \le j \le r$. Let $m = k_1 + \ldots + k_p$ and $n = l_1 + \ldots + l_r$ be the total number of c-transitions in M_1 and M_2 respectively.

$a_i^{(j)}$ and $b_i^{(j)}$ are target states of c-transitions in S_1 and S_2 respectively, where $a_i^{(j)} \in S_1, 1 \le i \le p, 1 \le j \le k_i$ and $b_i^{(j)} \in S_2, 1 \le i \le r, 1 \le j \le l_i$.

$$
k_1 \begin{cases} a_1 \xrightarrow{x_{a_1 a_1^{(1)}}} a_1^{(1)} \\ \vdots \\ a_1 \xrightarrow{x_{a_1 a_1^{(k_1)}}} a_1^{(k_1)} \end{cases}
\qquad
l_1 \begin{cases} b_1 \xrightarrow{y_{b_1 b_1^{(1)}}} b_1^{(1)} \\ \vdots \\ b_1 \xrightarrow{y_{b_1 b_1^{(l_1)}}} b_1^{(l_1)} \end{cases}
$$

$$
\vdots \qquad\qquad \vdots
$$

$$
k_p \begin{cases} a_p \xrightarrow{x_{a_p a_p^{(1)}}} a_p^{(1)} \\ \vdots \\ a_p \xrightarrow{x_{a_p a_p^{(k_p)}}} a_p^{(k_p)} \end{cases}
\qquad
l_r \begin{cases} b_r \xrightarrow{y_{b_r b_r^{(1)}}} b_r^{(1)} \\ \vdots \\ b_r \xrightarrow{y_{b_r b_r^{(l_r)}}} b_r^{(l_r)} \end{cases}
$$

$$
a_{p+1} \xrightarrow{\not c} \qquad\qquad b_{r+1} \xrightarrow{\not c}
$$

$$
\vdots \qquad\qquad \vdots
$$

$$
a_{p+q} \xrightarrow{\not c} \qquad\qquad b_{r+s} \xrightarrow{\not c}
$$

$x = \{x_{a_i a_i^{(j)}} \in \mathbb{R}^+ | 1 \le i \le p, 1 \le j \le k_i\}$ and $y = \{y_{b_i b_i^{(j)}} \in \mathbb{R}^+ | 1 \le i \le r, 1 \le j \le l_i\}$ are the sought rates of the c-transitions in M_1 and M_2 respectively[1].

3.1 Synchronising action

Lemma 1 (Complete System for Synchronising Action). *For the system Sys described in Def. 1 where $c \in \Sigma_s$, if all c-transitions in M_1 can move*

[1] x_{ab} is the transition rate from state a to b in M_1, and as a special case, x_{aa} is the rate of the selfloop at state a, used in particular in Sect. 3.2. Similarly y_{ab} for M_2.

simultaneously with all c-transitions in M_2, i.e. $en_1 \times en_2 \subseteq S$, then the resultant system of equations with regards to action c is always complete. The system of equations (3) represents all c-transitions in the combined model.

$$\left\{\begin{array}{ll} (a_1,b_1) \xrightarrow{d_1} (a_1^{(1)},b_1^{(1)}) & x_{a_1a_1^{(1)}}y_{b_1b_1^{(1)}} = d_1 \\ \vdots & \vdots \\ (a_1,b_r) \xrightarrow{d_n} (a_1^{(1)},b_r^{(l_r)}) & x_{a_1a_1^{(1)}}y_{b_rb_r^{(l_r)}} = d_n \\ \hline \vdots & \vdots \\ \hline (a_p,b_1) \xrightarrow{d_{(m-1)\cdot n+1}} (a_p^{(k_p)},b_1^{(1)}) & x_{a_pa_p^{(k_p)}}y_{b_1b_1^{(1)}} = d_{(m-1)\cdot n+1} \\ \vdots & \vdots \\ (a_p,b_r) \xrightarrow{d_{m\cdot n}} (a_p^{(k_p)},b_r^{(l_r)}) & x_{a_pa_p^{(k_p)}}y_{b_rb_r^{(l_r)}} = d_{m\cdot n} \end{array}\right. \tag{3}$$

$d = (d_1,\ldots,d_{mn})$ is the vector of the given rates of the c-transitions in the combined model. The consistency of the system of equations can be checked using conditions (4) and a possible solution can be obtained by (5).

Proof. If $en_1 \times en_2 \subseteq S$, then all the combined model states in (3) are reachable and there are $m\cdot n$ c-transitions in the combined model. Every combined model c-transition represents a bilinear equation. The total number of bilinear equations is the product of the number of x and y variables. So, the system of bilinear equations is complete.

The necessary and sufficient conditions for the system of equations (3) to have a solution is :

$$\left\{\begin{array}{l} \frac{d_1}{d_2} = \frac{d_{n+1}}{d_{n+2}} = \ldots = \frac{d_{(m-1)n+1}}{d_{(m-1)n+2}} \\ \frac{d_1}{d_3} = \frac{d_{n+1}}{d_{n+3}} = \ldots = \frac{d_{(m-1)n+1}}{d_{(m-1)n+3}} \\ \vdots \\ \frac{d_1}{d_n} = \frac{d_{n+1}}{d_{2n}} = \ldots = \frac{d_{(m-1)n+1}}{d_{mn}} \end{array}\right. \tag{4}$$

If these conditions are satisfied, a solution can be obtained using the following equations (5). We can choose one variable freely, say $y_{b_1b_1^{(1)}} = h$, and obtain the other variables.

$$\left\{\begin{array}{l} \frac{y_{b_1b_1^{(1)}}}{y_{b_1b_1^{(2)}}} = \frac{d_1}{d_2} \\ \vdots \\ \frac{y_{b_1b_1^{(1)}}}{y_{b_rb_r^{(l_r)}}} = \frac{d_1}{d_n} \end{array}\right. \Rightarrow \left\{\begin{array}{l} y_{b_1b_1^{(2)}} = h\frac{d_2}{d_1} \\ \vdots \\ y_{b_rb_r^{(l_r)}} = h\frac{d_n}{d_1} \end{array}\right. \Rightarrow \left\{\begin{array}{l} x_{a_1a_1^{(1)}} = \frac{d_1}{h} \\ \vdots \\ x_{a_pa_p^{(k_p)}} = \frac{d_{(m-1)n+1}}{h} \end{array}\right. \tag{5}$$

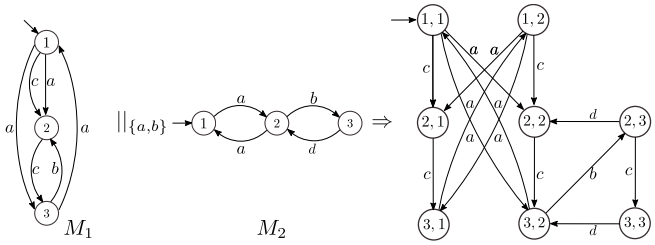

Fig. 3: Processes M_1 and M_2 and the resultant combined model

Example In Figure 3 there are two processes M_1 and M_2 synchronised over actions $\Sigma_s = \{a, b\}$ and the resultant combined model. We need to reconstruct the rates of a-transitions. For every a-transition in the combined model, an equation is created and the system of bilinear equations is formed.

Table (1) shows the a-transitions and the system of equations for the rates $d = (d_1, \ldots, d_6) = (3, 2, 1.5, 1, 10.5, 7)$. Figure 4 shows M_1 and M_2 with only a-transitions. Applying Def. 1 for M_1, we would have $a_1 = 1$, $k_1 = 2$ and $a_2 = 3$, $k_2 = 1$ and similar for M_2.

$$(1,1) \xrightarrow{a,d_1} (2,2) \Longrightarrow x_{12} \cdot y_{12} = d_1$$
$$(1,2) \xrightarrow{a,d_2} (2,1) \Longrightarrow x_{12} \cdot y_{21} = d_2$$
$$(1,1) \xrightarrow{a,d_3} (3,2) \Longrightarrow x_{13} \cdot y_{12} = d_3$$
$$(1,2) \xrightarrow{a,d_4} (3,1) \Longrightarrow x_{13} \cdot y_{21} = d_4$$
$$(3,1) \xrightarrow{a,d_5} (1,2) \Longrightarrow x_{31} \cdot y_{12} = d_5$$
$$(3,2) \xrightarrow{a,d_6} (1,1) \Longrightarrow x_{31} \cdot y_{21} = d_6$$

Table 1: a-transitions and the system of nonlinear equations.

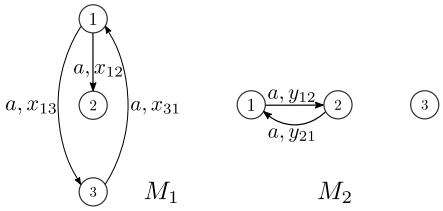

Fig. 4: Processes M_1 and M_2 with a-transitions only.

The necessary condition in Lemma (1) is satisfied, since the number of a-transitions in the combined model is equal to $m \cdot n = 6$ where $m = 3$ is the number of a-transitions in M_1 and $n = 2$ is the number of a-transitions in M_2. According to condition (4), the necessary condition for the system of equations in Table (1) to have a solution is: $\frac{d_1}{d_2} = \frac{d_3}{d_4} = \frac{d_5}{d_6}$ and vector d satisfies this condition. Therefore, this system of equations has a solution, y_{12} can be chosen freely, e.g. $y_{12} = h = 1$, and the other unknowns are determined: $y_{21} = d_2/d_1 = 2/3$, $x_{13} = d_3/h = 1.5$, $x_{12} = d_1/h = 3$, $x_{31} = d_5/h = 10.5$.

3.2 Non-synchronising action

In this section we consider the transition rate reconstruction for non-synchronising actions. If local lifting is not possible, the action is added to the synchronising set, the necessary selfloops are added and a system of bilinear equations is formed. We will see that for a class of systems, this system of equations is always complete.

Spurious Transitions: Our aim is to reconstruct the transition rates of the high-level model by transition rate assignment, possibly adding some actions to the synchronising set and adding selfloops to the components, but not by adding nor removing any transition(s) in the combined model. Adding action c to the synchronising set incurs the danger of creating some incorrect *spurious* transitions. For instance, in the system described in Def. 1, if state (a_1, b_1) in the combined model is reachable, having two transitions $(a_1, b_1) \xrightarrow{c} (a_1^{(1)}, b_1)$ and $(a_1, b_1) \xrightarrow{c} (a_1, b_1^{(1)})$, creates a spurious transition $(a_1, b_1) \xrightarrow{c} (a_1^{(1)}, b_1^{(1)})$ which should not exist in the combined model. Therefore, the set of states $SP = en_1 \times en_2$, in the combined model must be unreachable ($SP \cap S = \emptyset$) to avoid spurious transitions. The same condition is also checked by the algorithm in [17], thus guaranteeing that the behaviour of the system is not affected by the extended synchronisation set and the added selfloops.

Lemma 2 (Complete system for Non-synchronising Action). *In the system described in Def. 1, assume that $c \notin \Sigma_s$ and local lifting is not possible, so action c is added to Σ_s and c-selfloops are added to states $a_{p+1}, \ldots, a_{p+q}$ and to states $b_{r+1}, \ldots, b_{r+s}$. If $(en_1 \times ne_2) \cup (ne_1 \times en_2) \subseteq S$ and $SP \cap S = \emptyset$, then the resultant system of equations includes two independent complete bilinear systems of equations. The consistency of these systems of equations can be checked using conditions (7), and the possible solution can be derived by equations (8).*

Proof. Every c-transition in the combined model is transformed to a bilinear equation and the following systems (6) are formed.

$$
\begin{cases}
x_{a_1 a_1^{(1)}} \cdot y_{b_{r+1} b_{r+1}} = d_1 \\
\vdots \\
x_{a_1 a_1^{(k_1)}} \cdot y_{b_{r+1} b_{r+1}} = d_{k_1} \\
\hline
\vdots \\
\hline
x_{a_p a_p^{(1)}} \cdot y_{b_{r+1} b_{r+1}} = d_{k_1 + \ldots + k_{p-1} + 1} \\
\vdots \\
x_{a_p a_p^{(k_p)}} \cdot y_{b_{r+1} b_{r+1}} = d_m \\
\hline
\end{cases}
\quad
\begin{cases}
x_{a_{p+1} a_{p+1}} \cdot y_{b_1 b_1^{(1)}} = e_1 \\
\vdots \\
x_{a_{p+1} a_{p+1}} \cdot y_{b_1 b_1^{(l_1)}} = e_{l_1} \\
\hline
\vdots \\
\hline
x_{a_{p+1} a_{p+1}} \cdot y_{b_r b_r^{(1)}} = e_{l_1 + \ldots + l_{r-1} + 1} \\
\vdots \\
x_{a_{p+1} a_{p+1}} \cdot y_{b_r b_r^{(l_r)}} = e_n \\
\hline
\end{cases}
\quad
\begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \vdots \end{matrix}
\qquad
\begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \vdots \end{matrix}
\tag{6}
$$

$$
\begin{cases}
\vdots & \vdots \\
\hline
x_{a_1 a_1^{(1)}} \cdot y_{b_{r+s} b_{r+s}} = d_{(s-1)m+1} & x_{a_{p+q} a_{p+q}} \cdot y_{b_1 b_1^{(1)}} = e_{(q-1)\cdot n+1} \\
\vdots & \vdots \\
x_{a_1 a_1^{(k_1)}} \cdot y_{b_{r+s} b_{r+s}} = d_{(s-1)m+k_1} & x_{a_{p+q} a_{p+q}} \cdot y_{b_1 b_1^{(l_1)}} = e_{(q-1)\cdot n+l_1} \\
\hline
\vdots & \vdots \\
\hline
x_{a_p a_p^{(1)}} \cdot y_{b_{r+s} b_{r+s}} = d_{s\cdot m-k_p+1} & x_{a_{p+q} a_{p+q}} \cdot y_{b_r b_r^{(1)}} = e_{q\cdot n-l_r+1} \\
\vdots & \vdots \\
x_{a_p a_p^{(k_p)}} \cdot y_{b_{r+s} b_{r+s}} = d_{s\cdot m} & x_{a_{p+q} a_{p+q}} \cdot y_{b_r b_r^{(l_r)}} = e_{q\cdot n}
\end{cases}
$$

where $d = (d_1, \ldots, d_{s\cdot m})$ and $e = (e_1, \ldots, e_{q\cdot n})$ are the c-transition rates of the combined model.

The LHS system of equations of (6) is independent of the RHS since they have no common variables. The number of equations in the LHS system of equations is $m \cdot s$ which is the product of the number of x and y variables in this system of equations. Similarly, the number of equations in the RHS system of equations is $n \cdot q$ which is the product of the number of x and y variables in this system of equations. Therefore, both the RHS and LHS systems of equations are complete bilinear systems. Satisfying the condition $(en_1 \times ne_2) \cup (ne_1 \times en_2) \subseteq S$, we can make sure that all the equations in (6) exist, otherwise there would be fewer equations and the systems of bilinear equations would be incomplete.

The necessary conditions for these two independent systems of equations to be consistent are:

$$
\begin{cases}
\frac{d_1}{d_2} = \frac{d_{m+1}}{d_{m+2}} = \ldots = \frac{d_{(s-1)m+1}}{d_{(s-1)m+2}} \\
\frac{d_1}{d_3} = \frac{d_{m+1}}{d_{m+3}} = \ldots = \frac{d_{(s-1)m+1}}{d_{(s-1)m+3}} \\
\vdots \\
\frac{d_1}{d_m} = \frac{d_{m+1}}{d_{2m}} = \ldots = \frac{d_{(s-1)m+1}}{d_{s\cdot m}}
\end{cases}
\qquad
\begin{cases}
\frac{e_1}{e_2} = \frac{e_{n+1}}{e_{n+2}} = \ldots = \frac{e_{(q-1)n+1}}{e_{(q-1)n+2}} \\
\frac{e_1}{e_3} = \frac{e_{n+1}}{e_{n+3}} = \ldots = \frac{e_{(q-1)n+1}}{e_{(q-1)n+3}} \\
\vdots \\
\frac{e_1}{e_n} = \frac{e_{n+1}}{e_{2n}} = \ldots = \frac{e_{(q-1)n+1}}{e_{q\cdot n}}
\end{cases}
\tag{7}
$$

If vectors d and e satisfy the conditions (7) then high-level transition rate reconstruction is possible by making action c synchronising and adding the selfloops.

We can choose two variables freely, say $x_{a_1 a_1^{(1)}} = h_1$ and $y_{b_1 b_1^{(1)}} = h_2$, and obtain the other unknowns:

$$\begin{cases} x_{a_1 a_1^{(2)}} = h_1 \cdot \frac{d_2}{d_1} \\ \vdots \\ x_{a_p a_p^{(k_p)}} = h_1 \cdot \frac{d_m}{d_1} \\ y_{b_{r+1} b_{r+1}} = \frac{d_1}{h_1} \\ \vdots \\ y_{b_{r+s} b_{r+s}} = \frac{d_{(s-1)m+1}}{h_1} \end{cases} \qquad \begin{cases} y_{b_1 b_1^{(2)}} = h_2 \cdot \frac{e_2}{e_1} \\ \vdots \\ y_{b_r b_r^{(l_r)}} = h_2 \cdot \frac{e_n}{e_1} \\ x_{a_{p+1} a_{p+1}} = \frac{e_1}{h_2} \\ \vdots \\ x_{a_{p+q} a_{p+q}} = \frac{e_{(q-1)n+1}}{h_2} \end{cases} \tag{8}$$

4 Iterative Algorithm to Solve System of Equations

So far, we have identified special cases for which we can obtain the solution, if one exists, by checking a simple condition over the given rates d. For general systems which do not fit in with this framework, we need to solve the system of equations directly, which is computationally expensive. For example, in case of a synchronising action c, if the number of c-transitions in the combined model is less than the product of the number of c-transitions in the components, then we cannot use Lemma 1. In other words, if some of the transitions in (3) do not exist (because some states in the combined model are non-reachable), then the system is *incomplete* and Lemma 1 is not applicable anymore.

We now introduce an iterative algorithm which exploits the fact that the system of equations always has a specific form. The algorithm works for synchronising actions c. But it can also be used for non-synchronising actions c, after it has become clear that no local solution exists and after adding c to the synchronising set and adding the necessary selfloops.

Iterative Algorithm The system of equations to be solved contains the variables $x = (x_1, \ldots, x_m)$ and $y = (y_1, \ldots, y_n)$, and the equations have the specific form $x_r \cdot y_s = d_i$, where $i \in \{1, \ldots, m \cdot n\}$, $r \in \{1, \ldots, m\}$, $s \in \{1, \ldots, n\}$ and x_r and y_s are elements of x and y vectors, respectively. The unknown variable vectors x and y represent the transition rates of the components, and $(d_1, \ldots, d_k)$ are the given combined model transition rates. Although this system of equations is a nonlinear multivariate system, this form of equations enables us to develop an iterative algorithm to solve it efficiently.

1: **Algorithm** SolveBilinEqns (*Eqns*)
2: // The algorithm solves a system of k bilinear equations
3: // of the form $x_r y_s = d_i$, where $i \in \{1, \ldots, m \cdot n\}$.
4: // *Eqns* points to the totally ordered list of equations.
5: **Initialisation:** $Computed := \emptyset$
6: // The set *Computed* contains variables whose value has already been computed.
7: **repeat**
8: *curreqn* := first remaining equation from *Eqns*
9: // throughout, *curreqn* denotes the currently processed equation
10: delete *curreqn* from *Eqns*

```
11:     assign an arbitrary non-zero value to the first variable of curreqn and calculate the other
        variable
12:     add these two variables and their values to Computed
13:     repeat
14:       curreqn := equation which is next in order in Eqns
15:       if both of the variables of curreqn are in the Computed set then
16:         evaluate curreqn
17:         if curreqn holds then
18:           remove curreqn from Eqns
19:         else
20:           return Error and exit the algorithm
21:           // the system of equations is inconsistent and has no solution
22:         end if
23:       else if exactly one of the variables of curreqn is in the Computed set then
24:         calculate the other variable of curreqn and add it to Computed
25:         remove curreqn from Eqns
26:       else
27:         // both variables are not yet in Computed
28:         skip curreqn // the algorithm will return to this equation later
29:       end if
30:     until the last equation in Eqns has been reached
31:   until  Eqns = ∅ // i.e., until all equations have been processed
```

This algorithm takes a totally ordered set of equations *Eqns* as its input (the ordering is arbitrary, but remains fixed). The currently processed equation is called *curreqn*, and after an equation has been processed, it is removed from *Eqns*. The variables that have already been assigned a value are added to a set called *Computed*, which is initially empty. The algorithm processes the (remaining) equations in order, possibly in several iterations. When the algorithm checks the next equation in *Eqns*, three possible scenarios may occur: (1) If both of the variables are already in the *Computed* set, the equation is evaluated, and if it holds, it is removed from *Eqns*; otherwise, the algorithm terminates unsuccessfully, because the system of equations is inconsistent. (2) If exactly one of the variables in the current equation is in the *Computed* set, then the other variable is calculated and also added to *Computed*, and the equation is removed from *Eqns*. (3) If neither of the variables is in the *Computed* set, the equation is skipped, and will be checked again in the next iteration of the outer repeat-until loop. The reason is that the algorithm needs to avoid setting variables too early, since that might cause some of the following equations not to hold even if the system of equations is consistent. The algorithm terminates successfully once all the equations are checked, at which state the set *Eqns* is empty.

5 Case Study: Tandem queueing network

In this section, we study a tandem queueing network taken from [9] as a benchmark and compare the runtimes of the different methods for solving the system of bilinear equations.

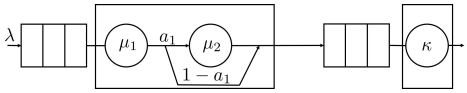

Fig. 5: M/Cox2/1-queue sequentially composed with an M/M/1-queue

5.1 System Definition

The model consists of an M/Cox2/1-queue sequentially composed with an M/M/1-queue, resulting in a finite-state CTMC[2]. Each queue has the capacity $c > 0$, where $c \in \mathbb{N}$ is an adjustable parameter. Figure 5 shows a schematic of the system. A job arrives at the system with rate λ, and the server of the first station executes service in one or two phases with rates μ_1 and μ_2. The probability that the first station executes service in both phases is a_1, and κ is the second station's service rate. We perform our experiments for different values of c and take the following values for the parameters of the queues: $\lambda = 4 \cdot c$, $a_1 = 0.1$, $\mu_1 = \mu_2 = 2$ and $\kappa = 4$.

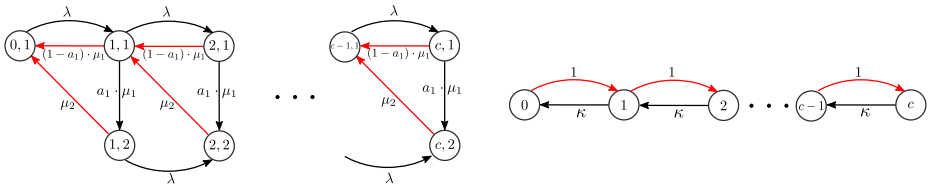

Fig. 6: CTMC models of M/Cox2/1 and M/M/1 queues

CTMC models of these two queues are shown in Figure 6, where the states of the M/Cox2/1-queue are 2-tuples, the first number showing the number of jobs, and the second number indicating the service phase. The SPA representation of the system is:

$$Sys = (M/Cox2/1) \parallel_{\{route\}} (M/M/1) \tag{9}$$

This model has one synchronising action, which is called *route* (highlighted in red in Figure 6). In the first queue this action means that a job leaves the queue, and in the second queue that a job enters the queue. These two events must always occur simultaneously in both queues. Each of the $2c$ *route*-transitions of the first queue can synchronise with each of the c *route*-transitions of the second queue, so in the combined model there are $2c^2$ *route*-transitions. In other words, using the terminology of Section 2, the system is *complete* regarding the action *route*, so that we can use Lemma 1 to verify the system of equations' consistency and obtain a possible solution.

[2] The PRISM source code of this model is provided at `https://www.prismmodelchecker.org/casestudies/tandem.php`.

5.2 Runtime comparison

In this section, we use a slight modification of our iterative algorithm from Section 4 to generate random sets of values d for the rates of the combined system's *route*-transitions, thereby ensuring that the resulting system of equations is consistent. This modification works as follows: Initially, we generate a fully random set of values d and start running the algorithm. Once the algorithm reaches an equation which does not hold (line 20 of the algorithm), the value of the rate d_i is simply changed such that the equation becomes consistent. Similarly, we go through the remaining equations and generate the modified set d.

For such a consistent set of the combined model's *route*-transition rates $d = \{d_1, \ldots, d_{2c^2}\}$, we will now compare the runtimes needed by the different solution methods. We compare the approaches of Lemma 1, our iterative algorithm and Gurobi [7], all implemented in (resp. called from) a Matlab [16] environment. Table 2 shows the size of the system for different values of c, where the number of *route*-transitions equals the size of the system of equations, which is also the size of vector d. Table 3 shows the runtime for the three methods[3].

c	200	250	300	350	400	450	500
Total number of states	80601	125751	180901	246051	321201	406351	501501
Total number of transitions	280599	438249	630899	858549	1121199	1418849	1751499
Number of *route*-transitions	80000	125000	180000	245000	320000	405000	500000

Table 2: Size of the system depending on parameter c

c / method	200	250	300	350	400	450	500
Lemma 1	24.5	69.7	171.1	314.7	443.8	723.4	1051.8
Iterative Algorithm	29.3	86.1	196.6	350.2	477.1	801.9	1134.5
Gurobi Solver	62.2	216.2	349.6	612.3	793.3	1114.1	1492.4

Table 3: Runtime in seconds

As we see in Table 3, Lemma 1 is the fastest method to check the consistency of the system of equations (at least 41% faster than Gurobi), followed by our iterative algorithm. Gurobi is always the slowest method. In this comparison, one has to consider that Gurobi is a highly optimised commercial tool (one of the most powerful solvers currently available), while our own implementations consist of simple, non-optimised Matlab code. However, this result was to be expected since in Lemma 1 only some simple conditions over d are checked, and in our iterative algorithm the special structure of the bilinear system of equations

[3] Executed on Intel Core i7-8650U CPU@ 1.90GHz-2.11GHz

is exploited, both processes being computationally much less expensive than general equation solving.

But Gurobi has the significant advantage that it allows the user to define a broad range of objectives and obtain an optimised solution. If the bilinear system of equations has a solution, it is not unique. This fact is clearly shown in Lemma 1 and Lemma 2, where an infinite number of solutions can be found by freely choosing some variables. This optimisation feature becomes especially interesting in applications of model repair, where we prefer to find the components' transition rates close to their original rates. So the following objective function, employing the Euclidean distance, can be defined and minimised by Gurobi: $\min||x - ox|| = \min\sqrt{\sum_i (x_i - ox_i)^2}$, where ox is the set of the original transition rates wrt x as given in the PRISM specification of the system. Similarly, we can write the objective function for y variables. Such an optimisation process is computationally expensive.

We fixed the value of c at $c = 400$ and generated 50 random sets of d, always ensuring that the systems of equations is consistent. Then Gurobi was run to verify the consistency, once with optimiser and once without optimiser. The runtime comparison is shown in Table 4. As shown in the table, on average, the runtime of the Gurobi solver with optimiser is more than 80% higher than the solver's runtime. For 88% of the d vectors, Gurobi solver with optimiser returned

c=400	Gurobi Solver	Gurobi Solver & Optimiser
Mean Time Value	753.6	1369.1
Min	617.2	638.3
Max	2581.7	6082.9

Table 4: Runtime comparison in seconds for 50 random sets of d

the solution in less than 1000 seconds. The maximum runtime for solver with optimiser is 6082.9, which is more than twice the solver's maximum runtime.

6 Conclusion

In this paper, we have studied the system of bilinear equations that arises during the reconstruction of transition rates in a two-component compositional model. The same type of system of equations has recently been described in the lifting algorithm from [17]. The problem is that solving such a bilinear system becomes very expensive if the number of equations is large.

In order to tackle this problem, we have identified a special class of systems where the consistency of the system of equations can be verified easily by checking some simple conditions. Once the consistency check – formulated in Lemma 1 for synchronising actions and in Lemma 2 for non-synchronising actions – holds, the solution to the system of equations is readily available. Furthermore, for general cases, where the system of equations does not possess such a special form,

we have developed a simple but efficient iterative algorithm which computes a solution, if a solution exists at all.

We have compared these different solution methods to each other, but also to the popular and powerful solver Gurobi, and the comparison turned out to be very favorable for our methods. Gurobi, however, is also able to construct solutions that satisfy additional optimality conditions (such as being close to some pre-specified values), which is also demonstrated in the paper by means of an example. In the future, we plan to extend the techniques developed in this paper to systems of more than two components.

References

1. Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans SW Eng **29**(6), 524–541 (2003)
2. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: TACAS. pp. 326–340. Springer LNCS 6605 (2011)
3. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C.R., Smolka, S.A.: Model repair for probabilistic systems. In: Abdulla, P.A., Leino, K.R.M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. pp. 326–340. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
4. Bernardo, M., Gorrieri, R.: A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time. TCS **202**(1), 1 – 54 (1998)
5. Chatzieleftheriou, G., Katsaros, P.: Abstract model repair for probabilistic systems. Information and Computation **259**, 142–160 (2018)
6. Dehnert, Junges, Katoen, J.P., Volk, M.: A Storm is coming: A modern probabilistic model checker. In: CAV. pp. 592–600. Springer LNCS 10427 (2017)
7. Gurobi Optimization, L.: Gurobi optimizer reference manual (2020), `http://www.gurobi.com`
8. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: IscasMC: A web-based probabilistic model checker. In: 19th Int. Symp. Formal Methods Europe. pp. 312–317. Springer LNCS 8442 (2014)
9. Hermanns, H., Meyer-Kayser, J., Siegle, M.: Multi terminal binary decision diagrams to represent and analyse continuous time Markov chains. In: Proc. 3rd Int. Workshop Numerical Solution of Markov Chains (NSMC'99). pp. 188–207. Prensas Universitarias de Zaragoza (1999)
10. Hillston, J.: The Nature of Synchronisation. In: Proc. 2nd Workshop on Process Algebras and Performance Modelling. pp. 51–70. Arbeitsberichte des IMMD 27(4), Universität Erlangen-Nürnberg (1994)
11. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press (1996)
12. Johnson, C.R., Link, J.A.: Solution theory for complete bilinear systems of equations. Numerical Linear Algebra with Applications **16**(11-12), 929–934 (2009)
13. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perf Eval **68**(2), 90 – 104 (2011)
14. Kuntz, M., Siegle, M., Werner, E.: Symbolic performance and dependability evaluation with the tool CASPA. In: Applying Formal Methods: Testing, Performance, and M/E-Commerce. pp. 293–307. Springer LNCS 3236 (2004)

15. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: CAV. pp. 585–591. Springer LNCS 6806 (2011)
16. MATLAB: version 9.6.0 (R2019a). The MathWorks Inc. (2019a)
17. Soltanieh, A., Siegle, M.: It sometimes works: A lifting algorithm for repair of stochastic process algebra models. In: Measurement, Modelling and Evaluation of Computing Systems. pp. 190–207. Springer LNCS 12040 (2020)
18. Tati, B.: Quantitative model repair of stochastic systems. Ph.D. thesis, Bundeswehr University Munich (2018)
19. Tati, B., Siegle, M.: Rate reduction for state-labelled Markov chains with upper time-bounded CSL requirements. EPTCS **220**, 77–89 (2016)
20. Tati, B.S.K., Siegle, M.: Parameter and Controller Synthesis for Markov Chains with Actions and State Labels. In: 2nd Int Workshop Synthesis of Complex Parameters (SynCoP'15). OpenAccess Series in Informatics (OASIcs), vol. 44, pp. 63–76. Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik (2015)