It sometimes works: A lifting algorithm for repair of Stochastic Process Algebra models

Amin Soltanieh and Markus Siegle

Universität der Bundeswehr München, 85577 Neubiberg, Germany,
{amin.soltanieh|markus.siegle}@unibw.de

Abstract. The paper presents an algorithm for lifting rate modification information from a flat Markovian model to its high-level modular description, specified with the help of a Stochastic Process Algebra (SPA). During the lifting, a specific set of transition rates in the model components is changed, and – if necessary – also the interaction between the components will be modified, in order to realise context-dependent rate modifications. It is shown that the proposed algorithm cannot always find a suitable lifting, but if such a lifting exists, the algorithm is guaranteed to find one. Furthermore, the paper shows that for a certain class of SPA product form models, the lifting algorithm will always find a solution.

Keywords: Markov Chains, Stochastic Process Algebra, Probabilistic Model Checking, Compositional Model Repair, Product Form

1 Introduction

For the specification of performance and dependability models, Stochastic Process Algebra (SPA) such as PEPA [1], EMPA [2] or CASPA [3], are often used, since they allow users to specify complex models in a modular and hierarchical way. Probabilistic model checking, implemented in tools such as PRISM [4] and STORM [5], is a powerful technique to reason about the properties of a system which is modelled, for example, with the help of an SPA. Although those specifications are compositional, model checking usually takes place at the level of the flat state space, i.e. at the level of a monolithic Continuous-Time Markov Chain (CTMC), labelled with atomic state properties. (In this paper, we focus on fully probabilistic systems, as opposed to, say, Markov Decision Processes.)

In case a model does not satisfy a given requirement, the user may want to modify it, such that the modified model will indeed satisfy the requirement. This process is referred to as model repair. While different strategies exist for carrying out model repair (see e.g. [6–8]), we focus on model repair by rate modification. This means that during model repair, the structure of the model is preserved, only some of the transition rates of the Markov chain are modified in such a way that the probabilities of the satisfying paths will be affected in a positive

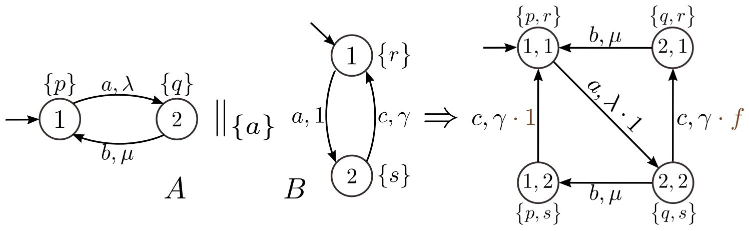

Fig. 1: Processes A and B and the resultant flat model

way. Earlier work on model repair by rate modification has been published in [9] for time-unbounded CSL [10] and asCSL [11] requirements, and in [12] for CSL Until requirements with upper or lower time bound. Those works addressed the problem of model repair at the level of a flat state-labelled Markov chain.

In this paper, we do not deal with the question of how to modify the transition rates of a Markovian model in order to satisfy a given requirement. We rather assume that the rate modification factors for the low-level flat Markov chain are already given. The question we address is the following: How, and under which conditions, is it possible to lift the information about rate modification from the level of the flat transition system to the level of the modular model specification? This is an important issue, since users do not want to work at the level of the flat Markov chain, they do not even want to see that low-level model. Users rather want to work with their high-level model specification, i.e. they wish to know how to change their model specification, in order to get a particular requirement satisfied. We now illustrate this problem by a motivating example.

In Fig. 1, inspired by [13], two processes A and B which are synchronised over action a, and also the resultant flat model are drawn. Each transition is specified by a tuple of the form (a, λ), where a is the action name and λ is the transition rate, and each state is labelled by atomic propositions (for the moment, the atomic propositions can be ignored). Now assume that – for the purpose of model repair – the rate of only one of the two c-transitions in the combined transition system should be multiplied by the factor f (as highlighted in Fig. 1), while the transition rate of the other c-transition should remain unchanged (thus the factor 1 in the figure). But both of these transitions stem from the same c-transition in component B. Therefore, a local repair, changing only the rate of the c-transition in component B, is not possible, since this would affect both c-transitions in the flat model. Whether or not to apply factor f depends on the context: On the c-transition originating in state $(2, 2)$ the factor should be applied, but in the one originating in $(1, 2)$ it should not.

This example shows that model repair at the level of the flat transition system, in general, cannot be lifted in a straight-forward way to the high-level model. Having realised this fact, the questions to be answered by this paper are: Is there a technique to make such a lifting possible? Is it always possible to find a suitable lifting? How does the algorithm work that decides whether lifting is possible, and how does such an algorithm construct a valid lifting?

This paper develops an algorithm that lifts rate modification information from the level of the flat Markov chain to the level of the compositional model specification. It is assumed that the model is specified with the help of a Markovian SPA, where, in particular, processes interact via action synchronisation. It turns out that such a lifting is not always possible, but our algorithm will detect this and stop with a suitable error message. Then, of course, the question arises whether it is possible to characterise the situations where the algorithm will be successful by suitable conditions. While we are not able to give necessary and sufficient conditions that cover all possible cases, Sec. 4 of the paper shows that for the class of SPA product-form models regarded in [14], the algorithm is always guaranteed to find a lifting.

This paper is structured as follows: Sec. 2 provides some background information on Stochastic Process Algebra and on model repair by rate modification. Sec. 3, the main section of the paper, contains the description of the new lifting algorithm. It starts by taking up the example from the introduction, before the actual algorithm is presented and discussed. Sec. 3 also contains a worked example where all the different cases occuring during the course of the algorithm are illustrated. In Sec. 4, we focus on a class of product form models, previously regarded in [14], for which, under some mild conditions, the lifting algorithm is always guaranteed to find a solution. Finally, Sec. 5 concludes the paper with a summary and a view into future extensions.

2 Modelling framework and state of the art

2.1 Stochastic Process Algebra (SPA)

We assume that the model to be analysed is specified with the help of a stochastic process algebra (SPA) such as PEPA [1], EMPA [2] or CASPA [3], where each transition is labelled by an action and a rate, the latter specifying an exponentially distributed delay. Complex models are constructed by parallel composition of components, which interact via action synchronisation. This yields an overall model with a modular or even hierarchical structure. The states of the overall model are tuples (i.e. vectors) of states of the constituent processes, and the semantic model is a Markovian (multi-)transition system [1].

For action synchronisation in SPA, different semantics have been discussed and compared in the past, concerning the rate of the synchronised transition (also called combined transition) as a function of the rates of the partner processes [15]. 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. Rate multiplication can be used, for instance, in such a way that during action synchronisation, one partner determines the basic rate and the other partner(s) are either passive (rate 1) or may contribute a factor (accelerating / decelerating) to the resulting rate. Technically, this is realised by rate factor multiplication. However, it is important to point out that the algorithm presented in Sec. 3 will also work for different synchronisation semantics, for instance if the combined rate is determined as the minimum of the two partner rates, or by consideration of the apparent rate of a given action type within the partner processes as in [1].

We do not go into the details of the syntax or semantics of stochastic process algebra. We only give the formal semantics (SOS rules) for parallel composition, since this is needed to understand the algorithm in Sec. 3:

$$\frac{P \xrightarrow{a,\lambda} P', Q \xrightarrow{a,\mu} Q'}{P \parallel_{\Sigma_s} Q \xrightarrow{a,op(\lambda,\mu)} P' \parallel_{\Sigma_s} Q'} \quad (a \in \Sigma_s)$$

$$\frac{P \xrightarrow{a,\lambda} P'}{P \parallel_{\Sigma_s} Q \xrightarrow{a,\lambda} P' \parallel_{\Sigma_s} Q} \quad (a \notin \Sigma_s) \qquad \frac{Q \xrightarrow{a,\mu} Q'}{P \parallel_{\Sigma_s} Q \xrightarrow{a,\mu} P \parallel_{\Sigma_s} Q'} \quad (a \notin \Sigma_s)$$

In these rules, P and Q are processes, $||$ is the parallel composition operator, $\Sigma_s \subseteq Act$ is the set of synchronising actions, $a \in Act$ is an action, and λ, μ are transition rates. The first rule realises the actual synchronisation via action a, where the resulting rate is obtained by combining the two partner rates by the binary operator *op* (which, as already said, we simply replace by multiplication). The other two (fully symmetrical) rules represent a move of one partner process while the other one remains stable.

2.2 Model repair by rate modification

The concept of model repair by rate modification has been developed in earlier works. In [9], the authors studied the model checking of action- and state-labelled CTMCs against requirements specified by the temporal logics CSL [10] and asCSL [11] (asCSL is an extension of CSL which allows one to describe complex path requirements, specified by regular expressions over action labels and state formulas). In particular, the paper [9] focused on time-unbounded CSL Until formulas and time-unbounded asCSL path requirements, to be checked on flat CTMC models. The central idea of the approach is as follows: If a given CSL requirement is violated, some specific transition rates in the model are changed deliberately, in order to affect the satisfaction probabilities in a positive way. For a violated CSL requirement, the model repair strategy of [9] consists of determining a particular subset of the CTMC's transitions whose rates should be reduced by a common factor. For violated asCSL requirements, the idea in [9] is similar, but the model repair algorithm is more complicated since it involves a product construction with a non-deterministic automaton representing the asCSL path formula.

In [12] similar approaches were developed, but for state-labelled CTMCs to be checked against CSL Until requirements with upper time bound. Again, the model repair strategy consists of determining rate reduction factors by which the transition rates between certain classes of states should be reduced. The theoretical background of perturbing the transition rates of a CTMC by a common factor, with applications to model repair, has been studied in [16].

While some previous work on model repair only considered rate reduction, in this work we allow rate modifications in both directions, i.e. the rate of a specific transition can be either reduced or increased. However, we do not allow reducing a rate to zero (thereby effectively deleting a transition in the CTMC), which guarantees that the graph of the CTMC is not changed by the repair.

Importantly, as already mentioned in the introduction, this paper does not concern itself with the question which rates of a given model should be modified, in order to satisfy the requirement at hand. It is assumed that the rate modification factors have already been determined by a model repair algorithm such as the ones mentioned above. But all mentioned previous work considered the problem of model repair at the level of a monolithic, flat model, which means that the rate modification factors are only known at the level of the flat model. However, when models are constructed from a modular or compositional model specification such as SPA, model repair should take place at the level of the high-level model, not at the level of the flat state space. This is the focus of the present paper: Answering the question if and how model repair information (in the form of transition rate modifications) can be lifted from the flat low-level model to the high-level model specification.

3 A lifting algorithm for modular model repair

3.1 Motivating examples

We return to the example from Sec. 1 and show how it can be fixed. In Fig. 1, we saw that local repair of just one component (component B) is not possible, since only one of the two c-transitions in the flat model should be changed. This means that there exists a context-dependency for the repair factor. We can implement this context-dependency by adding action c to the synchronisation set and inserting c-self-loops at the states of component A, as shown in Fig. 2. The such repaired components are now denoted A' and B'. The rates of those self-loops correspond to the desired modification factors. We can describe this approach mathematically by the following system of equations:

$$\begin{aligned} x_{11} \cdot y_{21} &= \gamma \cdot 1 \\ x_{22} \cdot y_{21} &= \gamma \cdot f \end{aligned}$$

where the x_{ii} are the rates of the self-loops in component A', and y_{21} is the rate of the c-transition in component B'. One solution to this system is $x_{11} = 1$, $x_{22} = f$ and $y_{21} = \gamma$, as depicted in Fig. 2.

These modifications obviously solve the model repair lifting problem for the example of Fig. 1. Unfortunately, such a solution is not always possible, as an extension of the same example shows (see Fig. 3). In this extended example, component A remains unchanged, but component B has been extended by a third state, as shown in Fig. 3. Now suppose that the two transitions $(2, 2) \xrightarrow{c,\gamma} (2, 1)$ and $(1, 2) \xrightarrow{c,\gamma} (1, 3)$ should both be multiplied each by factor f. Both of these transitions stem from component B, but again one can easily argue that a local repair, changing only component B, is not possible. Trying to solve the context-dependency by a similar approach as before will not work either

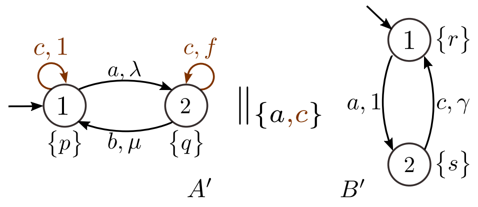

Fig. 2: Processes A' and B' with added self-loops and modified synchronising set

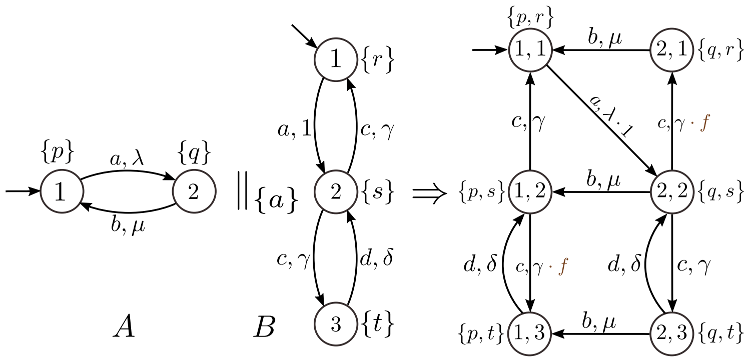

Fig. 3: Extended example

for this example. The reason is that the context (the state of component A) is contradictory in the following sense: When component A is in state 1, only the B-transition $2 \xrightarrow{c} 3$ should be modified, but not the B-transition $2 \xrightarrow{c} 1$, but when component A is in state 2, it is exactly the other way around! This cannot be controlled by c-self-loops on the states of component A. One can easily show this contradiction mathematically by the following system of nonlinear equations

$$\begin{aligned} x_{11} \cdot y_{21} &= \gamma \cdot 1 \\ x_{11} \cdot y_{23} &= \gamma \cdot f \\ x_{22} \cdot y_{21} &= \gamma \cdot f \\ x_{22} \cdot y_{23} &= \gamma \cdot 1 \end{aligned}$$

where, as before, x_{ii} are the rates of the c-self-loops that would be added to states of component A, and y_{21} and y_{23} are the rates of the c-transitions in component B. For $f \neq 1$, this system of equations does not possess a solution.

3.2 Idea of the algorithm

We present an algorithm which, for given rate multiplication factors, decides whether or not a lifting to the compositional model, by applying certain measures, is possible. If it is possible, the algorithm proposes a modification to the compositional high-level model, i.e. how exactly it should be changed. We now explain the general idea of the algorithm.

As its input, the algorithm takes a flat transition system whose states are n-tuples, each element characterising the current state of one of the n constituent processes. In order to keep the description simple, we assume for now that $n = 2$, but the algorithm can be extended to the case $n > 2$ in a rather straight-forward way. In addition to the flat transition system, the information about the rate modification factors is also given, i.e. it is known which transition rate of the flat model should be multiplied by which factor. The algorithm looks at the transitions whose rates are to be modified in a one by one fashion. A distinction is made, depending on whether the currently processed transition is a synchronising transition where several components are involved, or a local transition of only one single component.

Case (1): In case the currently processed transition is a synchronising transition, all transitions with the same action label are processed at once. For each of them, a nonlinear equation is created and a solution of the resulting system of equations is sought. If no solution exists, a lifting of the rate modification factors to the constituent components is not possible, so the algorithm terminates unsuccessfully. If a solution exists, the rates in the constituent processes are modified accordingly and the algorithm proceeds with the next transition to be modified.

Case (2): If the currently processed transition is a non-synchronising one, the algorithm first tries to fix its rate locally (i.e. by modifying only one of the components), thereby avoiding the creation of a (possibly large) system of equations. Such a local solution, however, is only possible if all modification factors of "parallel" transitions are identical, where "parallel" means that one component makes a specific move while the other component remains in any arbitrary fixed state. If a local solution is not possible, the modification factor of a non-synchronising transition depends on the context, i.e. the state of the other component. In this case, all transitions with the same action label are treated at once. The idea now is to change the transition from a non-synchronising transition to a synchronising one, thereby making it possible to control the modification factor depending on the context. For this reason, self-loops in the other component are added. In this case, a system of equations is created and solved. Again, it is possible that no solution exists, in which case a lifting is not possible.

3.3 Lifting algorithm

In order to keep the notation simple, we present the algorithm, assuming that the states of the flat transition system are 2-tuples. If the states are n-tuples, the algorithm can be adapted canonically.

We first introduce the notation used in the algorithm: Let M_1 and M_2 be two Markovian transition systems and $\Sigma_s \subseteq Act$ a set of (synchronising) actions. The flat transition system of $M_1||_{\Sigma_s}M_2$ is given by the set of tuples

$$T \subseteq (S_1 \times S_2) \times Act \times \mathbb{R}^{>0} \times (S_1 \times S_2)$$

where the first two elements denote the source state tuple, the last two elements denote the target state tuple, and the two elements in the middle denote the action label and the rate. Let S denote the reachable subset of $S_1 \times S_2$. Consider a subset of transitions $T_{mod} \subseteq T$. The rate of each transition $t_i \in T_{mod}$ is to be modified by an individual multiplicative factor $factor(t_i) \in \mathbb{R}^{>0} \setminus \{1\}$. For convenience, for transitions which are not to be modified, i.e. transitions $t_i \in T \setminus T_{mod}$, we define the modification factor as $factor(t_i) = 1$. For a combined transition $t = ((s,u) \xrightarrow{c,\gamma} (s',u')) \in T$, we use the following notation:

$$source(t) := (s,u),\ source_1(t) := s,\ source_2(t) := u$$

$$target(t) := (s',u'),\ target_1(t) := s',\ target_2(t) := u'$$

$$action(t) = c,\ rate(t) = \gamma$$

If for a transition $t \in T_{mod}$ we write $t = ((s,u) \xrightarrow{c,\gamma \cdot f} (s',u'))$ then we mean that $rate(t) = \gamma$ and $factor(t) = f \neq 1$ (although strictly speaking the modification factor is not stored as part of T_{mod}). If we write the same for a transition $t \in T \setminus T_{mod}$, then $factor(t) = f = 1$.

Given the desired rate modification factors for the flat composed model, the following algorithm computes the modified rates for M_1 and M_2, if a solution exists. Apart from changing rates in M_1 and M_2, the algorithm may also insert some self-loops and add actions to the synchronisation set Σ_s, in order to control the context in which a previously non-synchronising action takes place, which means controlling its rate in a context-dependent way. In other words, the algorithm lifts a given model repair strategy from the flat model to the compositional model, if such a lifting is possible at all. If successful, the algorithm returns the modified processes M_1' and M_2', as well as the potentially augmented synchronisation set Σ_s'. We now present the algorithm in pseudocode, where the explanation of all essential steps is given by the comments.

```
1:  **Algorithm** RepairLifting $(M_1, M_2, \Sigma_s, T, T_{mod}, factor)$
2:  // $T$ is the flat Markovian transition system of $M_1 ||_{\Sigma_s} M_2$.
3:  // The algorithm lifts the repair information given in the form of
4:  // rate modification factors $factor(t)$ for transitions $t \in T_{mod} \subseteq T$
5:  // to the high-level components $M_1$ and $M_2$, if possible.
6:  // The repaired system is returned as $M_1'$, $M_2'$ and $\Sigma_s'$.
7:  $M_1' := M_1$, $M_2' := M_2$, $\Sigma_s' := \Sigma_s$ // initialisation
8:  **while** $T_{mod} \neq \emptyset$ **do**
9:      choose $\hat{t} := ((\hat{s},\hat{u}) \xrightarrow{c,\hat{\gamma}\cdot\hat{f}} (\hat{s}',\hat{u}'))$ from $T_{mod}$
10:     // $\hat{t}$ is the transition processed during one iteration of the while-loop
11:
12:     // Case (1):
13:     **if** $action(\hat{t}) =: c \in \Sigma_s$ **then**
14:         // a synchronising transition needs to be modified
15:         $T_c := \{t \in T \mid action(t) = c\}$ // all $c$-transitions considered at once
16:         $T_{mod} := T_{mod} \setminus T_c$ // remove considered transitions from $T_{mod}$
```

```
17:     for each $t := ((s,u) \xrightarrow{c,\gamma\cdot f} (s',u')) \in T_c$ do
18:       create an equation $x_{ss'} \cdot y_{uu'} = \gamma \cdot f = \mathit{rate}(t) \cdot \mathit{factor}(t)$
19:       // or more general: create an equation $\mathit{op}(x_{ss'}, y_{uu'}) = \gamma \cdot f$
20:       // where $\mathit{op}$ is the operator determining the resulting rate (see Sec. 2.1)
21:     end for
22:     solve the system of nonlinear equations, i.e. find all $x_{ss'}$ and $y_{uu'}$
23:     if no solution exists then
24:       return “impossible”
25:     else
26:       for each $t := ((s,u) \xrightarrow{c,\gamma\cdot f} (s',u')) \in T_c$ do
27:         in $M_1'$ set $s \xrightarrow{c,x_{ss'}} s'$
28:         in $M_2'$ set $u \xrightarrow{c,y_{uu'}} u'$
29:       end for
30:     end if
31:
32:   // Case (2):
33:   else if $\mathit{action}(\hat{t}) =: c \notin \Sigma_s \wedge \hat{s} \neq \hat{s}'$ then
34:     // a non-synchronising $M_1$-transition needs to be modified,
35:     // the algorithm first tries to do this locally in $M_1$
36:     // by considering all “parallel” transitions at once
37:     $T_{c,\hat{s},\hat{s}'} := \{t \in T \mid \mathit{action}(t) = c \wedge \mathit{source}_1(t) = \hat{s} \wedge \mathit{target}_1(t) = \hat{s}'\}$
38:     if $\exists f_{com} \in \mathbb{R} : \forall t \in T_{c,\hat{s},\hat{s}'} : \mathit{factor}(t) = f_{com}$ then
39:       // there exists a common factor $f_{com}$ for all transitions in $T_{c,\hat{s},\hat{s}'}$
40:       in $M_1'$ set $\hat{s} \xrightarrow{c,\gamma_1\cdot f_{com}} \hat{s}'$ (where $\gamma_1$ is the current rate in $M_1'$)
41:       $T_{mod} := T_{mod} \setminus T_{c,\hat{s},\hat{s}'}$
42:       for each $t \in T_{c,\hat{s},\hat{s}'}$ do
43:         $\mathit{factor}(t) := 1$
44:         // the modification factor of the fixed transitions is changed to 1,
45:         // which is important in case they are considered again
46:         // when dealing with another $c$-transition from $T_{mod}$ later
47:       end for
48:     else
49:       // if the local fix in $M_1$ was not possible,
50:       // since non-synchronising action $c$ can also occur in $M_2$,
51:       // the algorithm now tries to modify all $c$-transitions at once
52:       // by making $c$ into a synchronising action.
53:       // but this only makes sense if there is no reachable combined state
54:       // from which $c$-transitions in both $M_1$ and $M_2$ are possible
55:       // (because if such a state existed, making $c$ into a synchronising
56:       // action would generate spurious (i.e. wrong) transitions)
57:       if $\exists (s,u) \in S : \exists t_1, t_2 \in T : (t_1 = ((s,u) \xrightarrow{c,\gamma_1} (s',u)) \wedge t_2 =$
          $((s,u) \xrightarrow{c,\gamma_2} (s,u')))$ then
58:         return “impossible”
59:       end if
```

```
60:          $T_c := \{t \in T \mid \mathit{action}(t) = c\}$
61:          $T_{mod} := T_{mod} \setminus T_c$
62:          for each $t := ((s,u) \xrightarrow{c,\gamma\cdot f} (s',u)) \in T_c$ do
63:             // an $M_1$-move
64:             create an equation $x_{ss'} \cdot y_{uu} = \gamma \cdot f = \mathit{rate}(t) \cdot \mathit{factor}(t)$
65:          end for
66:          for each $t := ((s,u) \xrightarrow{c,\gamma\cdot f} (s,u')) \in T_c$ do
67:             // an $M_2$-move
68:             create an equation $x_{ss} \cdot y_{uu'} = \gamma \cdot f = \mathit{rate}(t) \cdot \mathit{factor}(t)$
69:          end for
70:          solve the system of nonlinear equations, i.e. find all $x_{ss}, x_{ss'}, y_{uu}, y_{uu'}$
71:          if no solution exists then
72:             return “impossible”
73:          else
74:             for each $t := ((s,u) \xrightarrow{c,\gamma\cdot f} (s',u)) \in T_c$ do
75:                in $M_1'$ set $s \xrightarrow{c,x_{ss'}} s'$
76:                in $M_2'$ add self-loop $u \xrightarrow{c,y_{uu}} u$
77:             end for
78:             for each $t := ((s,u) \xrightarrow{c,\gamma\cdot f} (s,u')) \in T_c$ do
79:                in $M_2'$ set $u \xrightarrow{c,y_{uu'}} u'$
80:                in $M_1'$ add self-loop $s \xrightarrow{c,x_{ss}} s$
81:             end for
82:             $\Sigma_s' := \Sigma_s' \cup \{c\}$ // add action $c$ to the synchronisation set
83:          end if
84:       end if
85:
86:    // Case (2), symmetrical case:
87:    else if $\mathit{action}(\hat{t}) =: c \notin \Sigma_s \wedge \hat{u} \neq \hat{u}'$ then
88:       // an $M_2$-transition needs to be modified
89:       // this case is fully symmetrical to the previous case,
90:       // the code is analogous to lines 32-84
91:       ...
92:    end if
93: end while
94: return $M_1'$, $M_2'$, $\Sigma_s'$
```

3.4 Remarks on the algorithm

We now comment on the way the algorithm works and give some insight into how it can be made to work efficiently:

– The flat transition system T is not modified by the algorithm. That means for $t \in T$: $\mathit{rate}(t)$ stays unmodified during the course of the algorithm. The algorithm assigns new rates (which are the solutions of the systems of equations) to some transitions of M_1' and/or M_2' and possibly adds actions to the set of synchronising actions Σ_s'.

– The algorithm cannot always find a solution, in which case it returns "impossible" (lines 24, 58 and 72). However, the algorithm can always decide whether or not a solution exists, and if it exists the algorithm will find it.
– It is possible that the same transition rate in M'_i changes twice in the course of the algorithm. This will occur if a non-synchronising transition is first changed locally and needs to be considered again later, when another transition with the same action label cannot be changed locally. This is the reason, why resetting the modification factor of already fixed transitions in line 42 is necessary.
– The algorithm is not a truly compositional algorithm, but only a lifting algorithm, which has the disadvantage that the low-level model needs to be constructed. Thus, practical use of the algorithm is limited in case state space explosion occurs.
– Various optimisations of the algorithm are possible. E.g. there are conditions which can be checked in order to decide a priori whether the system of equations in lines 21 or 69 possesses a solution. In some cases, it is also possible to split a large system of equations into several smaller systems, thus reducing the effort to solve the equations. Elaborating on the details of such optimisations would be beyond the scope of the present paper.
– It is natural to ask for necessary / sufficient conditions under which the algorithm will find a solution. These could be conditions concerning the composition structure and other characteristics of the overall model, but also conditions on the requirement whose violation during model checking caused the need for the model to be repaired (e.g., if the requirement only refers to state properties of only a single model component). Ideally, one should be able to check those conditions at the specification level. The goal is to identify special classes of models which can be repaired at the specification level without having to analyse the low-level model. One such candidate is the class of product form models discussed in Sec. 4.

3.5 Illustration of the algorithm

In the following example we try to cover all possible cases which might occur during the course of the algorithm. Fig. 4 shows two processes M_1 and M_2, synchronised over actions a and b, and also the resultant flat model. Assume that all rates are equal to 1, and so also the rates in the flat model are all 1.

Model repair of synchronising transitions:
Let us assume that we need to modify the rates of a-transitions where a is a synchronising action. According to the algorithm (Case (1)), we create an equation for each of the a-transitions.

In this example there are six a-transitions in the flat model. Table 1 shows those transitions and the system of nonlinear equations created according to lines 17-21 of the algorithm. The rate modification factors are $f_1, \ldots, f_6$, and Fig. 5 shows M'_1 and M'_2 with only a-transitions where $x_{ss'}$ and $y_{uu'}$ are the modified rates.

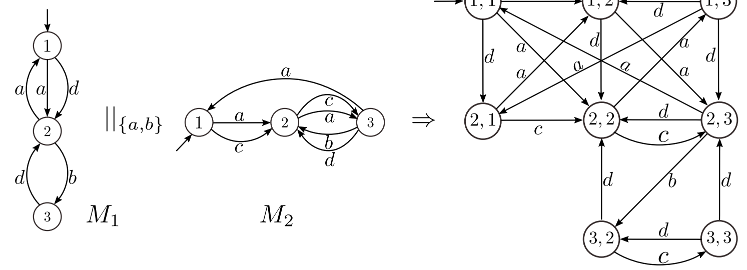

Fig. 4: Processes M_1 and M_2 and the resultant flat model

$$(1,1) \xrightarrow{a,1\cdot f_1} (2,2) \Longrightarrow x_{12}\cdot y_{12} = f_1$$
$$(1,2) \xrightarrow{a,1\cdot f_2} (2,3) \Longrightarrow x_{12}\cdot y_{23} = f_2$$
$$(1,3) \xrightarrow{a,1\cdot f_3} (2,1) \Longrightarrow x_{12}\cdot y_{31} = f_3$$
$$(2,1) \xrightarrow{a,1\cdot f_4} (1,2) \Longrightarrow x_{21}\cdot y_{12} = f_4$$
$$(2,2) \xrightarrow{a,1\cdot f_5} (1,3) \Longrightarrow x_{21}\cdot y_{23} = f_5$$
$$(2,3) \xrightarrow{a,1\cdot f_6} (1,1) \Longrightarrow x_{21}\cdot y_{31} = f_6$$

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

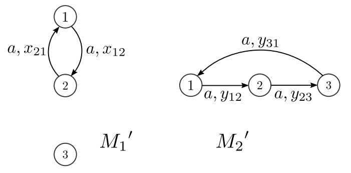

Fig. 5: Processes M_1' and M_2' with only a-transitions shown in Case (1).

The necessary condition for the system of equations in Table 1 to have a solution is $\frac{f_1}{f_4} = \frac{f_2}{f_5} = \frac{f_3}{f_6}$ $(^*)$. If this condition holds, one of the unknowns can be chosen freely, e.g. $x_{21} := 1$, and then the other unknowns are determined:

$$\begin{aligned} x_{12} &= f_1/f_4 \\ y_{12} &= f_4/x_{21} = f_4 \\ y_{23} &= f_5/x_{21} = f_5 \\ y_{31} &= f_6/x_{21} = f_6 \end{aligned}$$

So, if the modification factors are such that condition $(^*)$ is satisfied, there exists a solution and lifting is possible.

Local repair of non-synchronising transitions:
Let us assume that we need to modify two of the d-transitions with common modification factor f:

$$(3,2) \xrightarrow{d,1\cdot f} (2,2) \qquad (3,3) \xrightarrow{d,1\cdot f} (2,3)$$

Following the algorithm, at first one of those d-transitions is picked from T_{mod}. Since action d is non-synchronising, the possibility of local model repair in one of the components should be tested. For that purpose, all other "parallel" d-transitions where M_1 moves from state 3 to state 2 (denoted by $T_{d,3,2}$) are investigated to check if there is a common modification factor for all transitions (algorithm line 38). Both the above mentioned transitions stem from the same transition in M_1 which is $(3) \xrightarrow{d,1} (2)$. Since the modification factor f is the same for both transitions, it is clear that local model repair in M_1 is possible, so the algorithm sets the rate of transition $(3) \xrightarrow{d,1} (2)$ in M_1 to the value $1 \cdot f$.

Impossibility of model repair of non-synchronising transitions:
We now study a similar case as the previous one, but with different modification factors. Assume that the following d-transitions need to be changed:

$$(1,1) \xrightarrow{d,1\cdot f_1} (2,1) \qquad (1,2) \xrightarrow{d,1\cdot f_2} (2,2) \qquad (1,3) \xrightarrow{d,1\cdot f_3} (2,3)$$

where $f_1 \neq f_2$ or $f_1 \neq f_3$. Action d is a non-synchronising action and it is obvious that local model repair is not possible, since condition in line 38 of the algorithm is not satisfied.

According to the condition in line 57 of the algorithm, model repair is impossible, since the non-synchronising action d is present in M_1 and M_2 and state $(1,3)$ in the flat model (which is a reachable state) has two outgoing d-transitions, one in the M_1-dimension and one in the M_2-dimension (the same is true for state $(3,3)$). The algorithm would stop at this point, as it should. If the algorithm went ahead and made action d into a synchronising action, then two spurious transitions would be created: one from state $(1,3)$ to state $(2,2)$, and one from $(3,3)$ to $(2,2)$. So making d into a synchronising action is not an option here.

Successful model repair of non-synchronising transitions:
Assume that the following c-transitions need to be modified, where action c is not a synchronising action and $f_1, f_2 \neq 1$:

$$(1,1) \xrightarrow{c,1\cdot f_1} (1,2) \qquad (1,2) \xrightarrow{c,1\cdot f_2} (1,3)$$

The difference between this case and the former case is that there is no reachable state in the flat model (Fig. 4) with two outgoing c-transitions stemming from M_1 and M_2, so the impossibility condition of line 57 of the algorithm is not satisfied.

It is clear that local model repair is not possible, since the two transitions $(1,1) \xrightarrow{c,f_1} (1,2)$ and $(2,1) \xrightarrow{c,1} (2,2)$ are contradicting for $f_1 \neq 1$. Therefore according to the algorithm, action c is added to the synchronising set and we need to consider all reachable c-transitions:

$$\begin{aligned}
(1,1) &\xrightarrow{c,1\cdot f_1} (1,2) \Longrightarrow x_{11} \cdot y_{12} = f_1 \\
(2,1) &\xrightarrow{c,1} (2,2) \quad\;\; \Longrightarrow x_{22} \cdot y_{12} = 1 \\
(1,2) &\xrightarrow{c,1\cdot f_2} (1,3) \Longrightarrow x_{11} \cdot y_{23} = f_2 \\
(2,2) &\xrightarrow{c,1} (2,3) \quad\;\; \Longrightarrow x_{22} \cdot y_{23} = 1 \\
(3,2) &\xrightarrow{c,1} (3,3) \quad\;\; \Longrightarrow x_{33} \cdot y_{23} = 1
\end{aligned}$$

The necessary condition for this system of equations to have a solution is $\frac{y_{12}}{y_{23}} = \frac{f_1}{f_2} = \frac{1}{1}$ which implies $f_1 = f_2$. So only if this condition is satisfied, lifting is

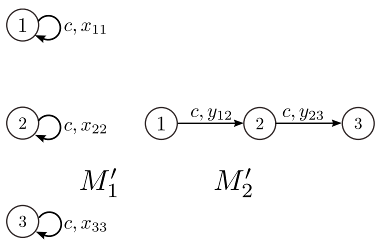

Fig. 6: Processes M'_1 and M'_2 (only c-transitions shown)

possible. One unknown variable can be chosen freely, say $x_{11} = 1$, then the other unknowns are determined:

$$x_{22} = x_{33} = 1/f_1$$
$$y_{12} = y_{23} = f_1$$

Fig. 6 shows the modified processes M'_1 and M'_2, with only c-transitions drawn.

4 Application to SPA models with product form

In this section, we identify a class of models for which the algorithm from Sec. 3 is guaranteed to always find a solution. In [14], a class of PEPA models is defined, based on Boucherie's framework [17] of Markov processes competing over resources, for which it is shown that the equilibrium probability distribution is of product form. These SPA models consist of a set of cyclic processes and a set of mutually independent resources

$$Sys = (M_1 \,||\, \ldots \,||\, M_K) \,||_L\, (R_1 \,||\, \ldots \,||\, R_M)$$

There is no direct interaction among processes M_k $(k = 1, \ldots, K)$, but indirect interaction because of the mutually exclusive use of the resources R_m $(m = 1, \ldots, M)$. Apart from the interaction with the processes, the resources must not have any internal behaviour, thus the resources are redundant at the state level. "Redundant" means that the current state of any of the resources can be inferred from the states of the K processes.

The original Boucherie framework of [17] requires a strong blocking condition, which means that when a process occupies a certain resource, all other processes competing over the same resource are fully blocked and not allowed to move at all. But in [14], instead of the strong blocking condition, there is a strong restriction on components, based on the notion of so-called *guarding* and *returning* resources. A resource is *guarding* with regard to a sequential component if all paths (which are actually cycles) starting from the initial state of the component and returning to the initial state, either cooperate with the resource on the first action of the cycle or are completely independent of the resource actions during the whole cycle. Similarly, a resource is *returning* with regard to a sequential component if all paths (cycles) starting from the initial state and returning to the initial state, either cooperate with the resource on the last action of the cycle or are completely independent of the resource actions during the whole cycle. For the formal definition of *guarding resource* and *returing resource* the reader is referred to [14].

We now study what happens when we apply our lifting algorithm from Sec. 3 to SPA product form models as described in [14]. As mentioned, resources are redundant at the state level, so their state is implicit and does not need to be recorded in the state descriptor of the combined model. The following theorem provides sufficient conditions for successful lifting of model repair information.

Theorem 1. *Let Sys be a SPA product form model as described in [14], i.e.* $Sys = (M_1 \parallel \ldots \parallel M_K) \|_L R$, *where* $M_1, \ldots, M_K$ *are cyclic sequential components and* R *is the independent parallel composition of distinct simple resource components which are guarding and returning with respect to each* M_k $(k = 1, \ldots, K)$ *and redundant in the state representation of the model. Let* T_{mod} *be the subset of transitions of the low-level model of Sys whose rates are to be multiplied by given factors. Lifting this model repair information to the specification level by applying the algorithm from Sec. 3 will be successful if:*

1. *all local actions within* M_k *(i.e. all actions of* $Act(M_k) \cap \overline{L}$*) are pairwise distinct (for all* $k = 1, \ldots, K$*)*
2. *the local actions of any two components are disjoint, i.e.* $Act(M_k) \cap Act(M_l) \cap \overline{L} = \emptyset$ *(for all* $k, l = 1, \ldots, K$*)*
3. *the rates of the synchronising actions are not to be modified, i.e.* $Act(T_{mod}) \cap L = \emptyset$

Note that in this Product Form framework there is no synchronisation among components, so Case (1) of the algorithm never occurs (all transitions in the combined model are just along one dimension, with exactly one process changing its state). Model repair will take place by either local repair inside one of the components M_k, or by inserting self-loops (for context-dependent control of the rates) and making the components synchronising over some action set $\Sigma_s \subseteq Act(T_{mod})$. The resulting repaired model is $(M'_1 \|_{\Sigma_s} \cdots \|_{\Sigma_s} M'_k) \|_L R$ where the M'_k are the modified components. We now give a proof sketch for the theorem.

Proof. For simplicity, the proof is carried out for the special case of $K = 2$ components, thus the system can be written as $Sys = (M_1 \| M_2) \|_L R$. The conditions in Theorem 1 indicate that any action $c \in Act(T_{mod})$ only exists in one of the components M_1 or M_2, and in this component there is only one c-transition. Assume that this c-transition is in M_1 and there are N reachable states in M_2, as shown in Fig. 7:
Then there exist (at most) N c-transitions in the flat model (with f_n, $n = 1, \ldots, N$, being the associated modification factors). Now either the system can be repaired by local repair of M_1, or we need to solve the following system of at most N equations (there might be fewer than N c-transitions if some combined

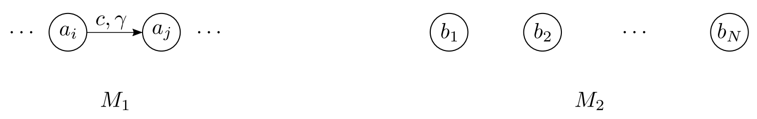

Fig. 7: Processes M_1 and M_2

states are unreachable):

$$\begin{array}{ccc} (a_i, b_1) \xrightarrow{c, 1 \cdot f_1} (a_j, b_1) & \qquad & x_{ij} \cdot y_{11} = f_1 \\ (a_i, b_2) \xrightarrow{c, 1 \cdot f_2} (a_j, b_2) & & x_{ij} \cdot y_{22} = f_2 \\ \vdots & & \vdots \\ (a_i, b_N) \xrightarrow{c, 1 \cdot f_N} (a_j, b_N) & & x_{ij} \cdot y_{NN} = f_N \end{array}$$

For arbitrary values of the multiplication factors $f_1, \ldots, f_N$, this system of equations always has a solution. One solution is $x_{ij} = 1$ and $y_{nn} = f_n$ for all $1 \leq n \leq N$. □

5 Summary and future work

In this paper, we have presented an algorithm which lifts model repair information, given in the form of transition rate modification factors, from a flat low-level CTMC to the associated high-level compositional model. The algorithm modifies a subset of the model components' transition rates, and it may also change the interaction between the components by adding actions to the synchronisation set. While such a solution to the lifting problem does not always exist, the algorithm is guaranteed to find a valid solution if it exists. For a special class of SPA product form models, we have shown that lifting is always possible.

In future work, we are planning to elaborate on the extension of the lifting algorithm to more than two processes. We are also planning to work on optimisations of the algorithm, such as splitting a large system of nonlinear equations, as created by the algorithm, into several smaller, independent systems of equations, whenever this is possible. There is also the question whether additional measures, apart from local rate modifications and augmenting the synchronisation set, could open up further opportunities for lifting. The long-term goal is to develop a truly compositional model repair approach, which – as opposed to the lifting approach presented here – would work directly at the level of the high-level compositional model specification.

Acknowledgement: The authors would like to thank Alexander Gouberman for critical comments on the manuscript.

References

1. J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, 1996.

2. M. Bernardo, R. Gorrieri, A tutorial on EMPA: A theory of concurrent processes with nondeterminism, priorities, probabilities and time, Theoretical Computer Science 202 (1) (1998) 1 – 54.
3. M. Kuntz, M. Siegle, E. Werner, Symbolic Performance and Dependability Evaluation with the Tool CASPA, in: M. Nunez, Z. Maamar, F. Pelayo (Eds.), FORTE 2004 Workshops, European Performance Engineering Workshop, Springer, LNCS 3236, 2004, pp. 293–307.
4. M. Kwiatkowska, G. Norman, D. Parker, PRISM 4.0: Verification of probabilistic real-time systems, in: Proc. 23rd Int. Conf. on Computer Aided Verification (CAV'11), Vol. 6806 of LNCS, Springer, 2011, pp. 585–591.
5. C. Dehnert, S. Junges, J.-P. Katoen, M. Volk, A storm is coming: A modern probabilistic model checker, in: R. Majumdar, V. Kunčak (Eds.), Computer Aided Verification, Springer International Publishing, 2017, pp. 592–600.
6. E. Bartocci, R. Grosu, P. Katsaros, C. Ramakrishnan, S. Smolka, Model Repair for Probabilistic Systems, in: TACAS'11, Springer LNCS 6605, 2011, pp. 326–340.
7. T. Chen, E. M. Hahn, T. Han, M. Kwiatkowska, H. Qu, L. Zhang, Model Repair for Markov Decision Processes, in: Proc. 7th Int. Symp. Theor. Aspects of Software Engineering (TASE), IEEE CS Press, 2013, pp. 85–92.
8. S. Pathak, E. Ábrahám, N. Jansen, A. Tacchella, J.-P. Katoen, A Greedy Approach for the Efficient Repair of Stochastic Models, in: NASA Formal Methods - 7th Int. Symposium, 2015, pp. 295–309.
9. B. Tati, M. Siegle, Parameter and Controller Synthesis for Markov Chains with Actions and State Labels, in: É. André, G. Frehse (Eds.), 2nd Int. Workshop on Synthesis of Complex Parameters (SynCoP'15), Vol. 44 of OpenAccess Series in Informatics (OASIcs), Dagstuhl, Germany, 2015, pp. 63–76.
10. C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, Model-Checking Algorithms for Continuous-Time Markov Chains, IEEE Transactions on Software Engineering 29 (6) (2003) 524–541.
11. C. Baier, L. Cloth, B. Haverkort, M. Kuntz, M. Siegle, Model Checking Markov Chains with Actions and State Labels, IEEE Transactions on Software Engineering 33 (4) (2007) 209–224.
12. B. Tati, M. Siegle, Rate reduction for state-labelled Markov chains with upper time-bounded CSL requirements, in: T. Brihaye et al. (Ed.), Proc. Cassting Workshop on Games for the Synthesis of Complex Systems and 3rd Int. Workshop on Synthesis of Complex Parameters, Open Publ. Assoc., El. Proc. in Theor. Computer Science Vol. 220, 2016, pp. 77–89.
13. B. Tati, Quantitative Model Repair of Stochastic Systems, Ph.D. thesis, Bundeswehr University Munich, Department of Computer Science (2018).
14. J. Hillston, N. Thomas, Product form solution for a class of PEPA models, Performance Evaluation 35 (3) (1999) 171 – 192.
15. J. Hillston, The Nature of Synchronisation, in: U. Herzog, M. Rettelbach (Eds.), Proc. of the 2nd Workshop on Process Algebras and Performance Modelling, Arbeitsberichte des IMMD 27(4), Universität Erlangen-Nürnberg, Regensberg/Erlangen, 1994, pp. 51–70.
16. A. Gouberman, M. Siegle, B. Tati, Markov Chains with Perturbed Rates to Absorption: Theory and Application to Model Repair, Performance Evaluation 130 (2019) 32–50.
17. R. J. Boucherie, A characterization of independence for competing Markov chains with applications to stochastic Petri nets, IEEE Transactions on Software Engineering 20 (7) (1994) 536–544.