Lumping partially symmetrical stochastic models

S. Baarir[2], M. Beccuti[2], C. Dutheillet[1], G. Franceschinis[2], S. Haddad[3], J.M. Ilié[4],

[1]LIP6
Université Paris 6
claude.dutheillet@lip6.fr

[2]Dip. di Informatica
Univ. Piemonte Orientale
{baarir,beccuti,franceschinis}@mfn.unipmn.it

[3]LAMSADE
Université Paris-Dauphine
haddad@lamsade.dauphine.fr

[4]LIP6
IUT Paris 5
Jean-Michel.Ilie@lip6.fr

Abstract

Performance and dependability evaluation of complex systems by means of dynamic stochastic models may be impaired by the combinatorial explosion of their state space. Among the possible methods to cope with this problem, symmetry-based ones can be applied to systems including several similar components. Often however these systems are only partially symmetric: occasionally similar components behavior may diverge from the usual symmetrical one.

In this paper two methods to efficiently analyse partially symmetrical models are presented in a general setting and the requirements for their efficient implementation are discussed. A complex realistic case study is presented to show the methods effectiveness and their applicative interest.

1 Introduction

As software systems and hardware architectures are more and more complex, their verification and evaluation become critical issues. Analysis methods are often subject to the problem of combinatorial explosion due to the increasing system complexity. Several approaches have been undertaken to cope with this problem: decomposition methods take advantage of the modular structure of the system; for performance evaluation, approximate and bounding methods substitute a simpler system to the original one; diagram decision based methods symbolically manage sets of states rather than representing states explicitly, etc. Here we present symmetry-based methods that exploit the presence of several similar components in the system.

The general principle of these methods consists to substitute to the state graph a quotient graph w.r.t. some equivalence relation. This relation considers two states as equivalent if they can be obtained from each other permuting equivalent components. These methods have been first introduced in order to check safeness properties (see e.g. [10, 11]), then generalized in order to check temporal logic formulae (see e.g. [6]) and also adapted to performance evaluation *via* the quantitative counterpart of symmetry, i.e. *lumpability* (see e.g. [4, **?**, **?**]). It should be stressed that the requirements w.r.t. lumpability are generally stronger than the ones that ensure equivalence between qualitative (symmetrical) behaviors and thus the design of such methods needs more elaboration.

In order to successfully exploit symmetries it is required to (1) define in a generic way, at the conceptual level, what method can be used to reduce the state space through symmetries (2) select a formalism where symmetries are automatically detected (3) define how the method can be efficiently implemented in practice. The design at the conceptual level is based on the operations of a permutation group; the formalism must allow a simple way to express similar components; the implementation should be based on a symbolic representation of set of states and transitions and their efficient manipulation.

However the systems seldom have completely symmetric behavior (for example in distributed algorithms we often have a symmetric specification, together with some symmetry-breaking criteria - e.g. based on unique process identity - to solve conflicts, deadlocks, etc.) so it is useful to define and implement methods to deal with partial symmetries. In the literature partial symmetry methods have been proposed for qualitative analysis [1, 7, 8, 9].

In this paper we propose two generic methods to apply lumping in partially symmetrical models, and discuss how they can be efficiently implemented in the context of the Stochastic Well-Formed Net (SWN) formalism [4]. The first one, called DS method, starts from a completely symmetric Markov chain (MC) and an additional automata describing the asymmetries: in this case a lumped MC satisfying the exact lumpability condition is built. The second one, called TLS method, instead starts from an over-aggregated MC from which a lumped MC can be derived by applying a refinement algorithm: it can use either the strong lumpability or the exact lumpability condition (which have different impact on the type of performance indices that can be com-

puted and may lead to different degrees of aggregation).

The two methods can be efficiently applied to SWNs[1] models, in fact this formalism is designed so that symmetries can be automatically detected and exploited. However here they are presented in a general setting so that they could be adapted to other kinds of high level stochastic models. One of the main differences between the two methods is that the TLS (Two-Levels Symmetry) method uses two different aggregation criteria depending on the current phase of the behavior (symmetric or asymmetric), while the DS (Dynamic Symmetry) method aggregates states in a more dynamic way, possibly using several different aggregation criteria which may correspond to a more articulated classification of the behavior phases: symmetric behavior or one among several asymmetric behaviors.

We have implemented our methods in the GreatSPN tool [5] allowing us to perform several experiments. A significant case study is presented in the paper, describing a remote service system modeled by means of the SWN formalism: the goal of the study is to evaluate the impact of accounting for different priority user classes, by computing the overall service throughput and other related performance indices. The two methods are applied to compute such indices: besides providing the measures of interest, the experimental results show that relevant savings in the state space size can be achieved through both approaches, and that they can be alternatively applied in the most appropriate situations.

The paper unifies and extends two results presented in [2, 3], revisiting them in a more general setting; a particular emphasis is given to the case study. It is organized as follows: in Sec. 2 some basic notions on MC and lumpability are defined, in Sec. 3 the TLS and DS methods are presented; in Sec. 4 a discussion on their implementation (including a comparison) is presented, finally in Sec. 5 a significant case study is presented and analyzed. We conclude in Sec. 6.

2 Markov chain lumpability

2.1 Strong, weak and exact lumpability

The quantitative evaluation of dynamic systems proposed in this paper implies the following three steps (a) the specification of the stochastic process representing the target system, (b) the definition of the required performance (or dependability) indices and (c) the possibility of applying efficient algorithms for transient or steady state measures computation.

The analysis of stochastic processes in general is an hard problem, in fact often simulation is the only viable option, while in some cases algorithms for the computation of approximations or bounds on the desired measures can be applied.

When the dynamic system behavior can be described through a finite Discrete Time MC (DTMC) or Continuous Time MC (CTMC), the solution is conceptually simpler, however, in realistic case studies, it is still computationally expensive; for this reason state space reduction techniques have been studied, such as the so called MC *lumping* technique.

Lumping of (finite) MCs is a useful method for dealing with large chains [12]. The principle is simple: substitute to the MC an "equivalent" one, where each state of the lumped chain is a set of states of the original one. There are different versions of lumpability related to the fact that the lumpability condition holds for every initial distribution (*strong lumpability*) or for at least one (*weak lumpability*). First, we briefly introduce MCs. Due to space constraints, we only deal with CTMCs. However our methods also apply to DTMCs and we indicate later on the interest of dealing with DTMCs even in a continuous time setting.

Definition 1 (Markov Chains) *A CTMC $\mathcal{C} = \langle S, \pi_0, \pi_0 \rangle$ is defined by a state space S, an infinitesimal generator Q (that is a $S \times S$ matrix whose off-diagonal elements are non negative reals, while each diagonal element is defined as $Q[s,s] = -\sum_{s \neq s'} Q[s,s']$, and π_0, an initial probability distribution over S. We note $\{X_t\}_{t \in \mathbb{R}_{\geq 0}}$ the associated stochastic process.*

Notation. S_0 denotes the subset of "initial" states, i.e., $S_0 = \{s \in S \mid \pi_0(s) > 0\}$.

We now introduce lumpability concepts.

Definition 2 *Let $\mathcal{C}$ be a CTMC and $\{S_i\}_{i \in I}$ be a partition of the state space. Let Y_t be a random variable defined by $Y_t = i \Leftrightarrow X_t \in S_i$. Then:*

- *Q is* strongly lumpable *w.r.t. $\{S_i\}_{i \in I}$ iff $\forall \pi_0$, $\{Y_t\}_{t \in \mathbb{R}_{\geq 0}}$ is a CTMC,*
- *Q is* weakly lumpable *w.r.t. $\{S_i\}_{i \in I}$ iff $\exists \pi_0$ s.t. $\{Y_t\}_{t \in \mathbb{R}_{\geq 0}}$ is a CTMC.*

Whereas the characterization of strong lumpability w.r.t. the infinitesimal generator is straightforward, checking for weak lumpability is much harder [13]. Here, we introduce the exact lumpability, a simpler case of weak lumpability.

Definition 3 *Let $\mathcal{C}$ be a CTMC and $\{S_i\}_{i \in I}$ be a partition of the state space. Let Y_t be a random variable defined by $Y_t = i \Leftrightarrow X_t \in S_i$. Then:*

- *An initial distribution π_0 is equiprobable w.r.t. $\{S_i\}_{i \in I}$ if $\forall i \in I, \forall s, s' \in S_i, \pi_0(s) = \pi_0(s')$.*
- *Q is* exactly lumpable *w.r.t. $\{S_i\}_{i \in I}$ iff $\forall \pi_0$ equiprobable w.r.t. $\{S_i\}_{i \in I}$ $\{Y_t\}_{t \in \mathbb{R}_{\geq 0}}$ is a CTMC.*

[1] SWNs are high level stochastic Petri nets with a restricted syntax for expressing color domains of places and transitions, arc functions and transition guards.

Exact and strong lumpability have easy characterizations [17] given by the following proposition.

Proposition 4 *Let $\mathcal{C}$ be a CTMC and $\{S_i\}_{i\in I}$ be a partition of the state space. Then:*

- *Q is strongly lumpable w.r.t. $\{S_i\}_{i\in I}$ iff $\forall i \neq j \in I$, $\forall s, s' \in S_i, \sum_{s''\in S_j} Q(s, s'') = \sum_{s''\in S_j} Q(s', s'')$,*
- *Q is exactly lumpable w.r.t. $\{S_i\}_{i\in I}$ iff $\forall i, j \in I$, $\forall s, s' \in S_i, \sum_{s''\in S_j} Q(s'', s) = \sum_{s''\in S_j} Q(s'', s')$.*

The following corollary establishes a sufficient condition for exact lumpability in CTMCs which will be useful in order to check the correctness of one of our methods.

Corollary 5 *Let $\mathcal{C}$ be a CTMC and $\{S_i\}_{i\in I}$ be a partition of the state space. Then Q is exactly lumpable w.r.t. $\{S_i\}_{i\in I}$ if:*

1. $\forall i \neq j \in I, \forall s, s' \in S_i$, $\sum_{s''\in S_j} Q(s'', s) = \sum_{s''\in S_j} Q(s'', s')$.
2. $\forall i \in I, \forall s, s' \in S_i$, $\sum_{s''\neq s\in S_i} Q(s'', s) = \sum_{s''\neq s'\in S_i} Q(s'', s')$.
3. $\forall i \in I, \forall s, s' \in S_i, Q(s, s) = Q(s', s')$.

When the strong lumpability condition holds the infinitesimal generator of the lumped chain can be directly computed from the original generator as expressed by the following proposition.

Proposition 6 *Let $\mathcal{C}$ be a CTMC that is strongly lumpable w.r.t. a partition of the state space $\{S_i\}_{i\in I}$. Let Q^{lp} be the generator associated with this lumped CTMC, then:*

$$\forall i, j \in I, \forall s \in S_i, Q^{lp}(i, j) = \sum_{s'\in S_j} Q(s, s').$$

As for strong lumpability, also in case of exact lumpability the infinitesimal generator of the lumped chain can be directly computed from the original generator. Observe that starting with the probability mass equidistributed on the states of every subset of the partition, the distribution at any time is still equidistributed. Consequently, if the CTMC is ergodic, its steady-state distribution is equidistributed between states of every subset of the partition. In other words, with the knowledge of the lumped chain generator, one may compute its steady-state distribution, and deduce (by *local* equidistribution) the steady-state distribution of the original chain. It must be emphasized that this last step is impossible with strong lumpability since it does not ensure equiprobability of the states in an aggregate.

Proposition 7 *Let $\mathcal{C}$ be a CTMC that is exactly lumpable w.r.t. a partition of the state space $\{S_i\}_{i\in I}$. Let Q^{lp} be the generator associated with this lumped CTMC, then:*

- *$\forall i, j \in I, \forall s \in S_j$, $Q^{lp}(i, j) = (\sum_{s'\in S_i} Q(s', s)) \times (|S_j|/|S_i|)$*
- *If $\forall i \in I, \forall s, s' \in S_i, \pi_0(s) = \pi_0(s')$ then $\forall t \in \mathbb{R}_{\geq 0}$, $\forall i \in I, \forall s, s' \in S_i, \pi_t(s) = \pi_t(s')$, where π_t is the probability distribution at time t.*
- *If Q is ergodic and π is its steady-state distribution then $\forall i \in I, \forall s, s' \in S_i, \pi(s) = \pi(s')$.*

2.2 Dealing with DTMCs.

As said before, very similar results hold for DTMCs. Furthermore even in a continuous time setting, there are two situations where using DTMCs is useful. Some semi-Markovian processes are analyzable *via* an *embedded* DTMC which only takes into account state changes. This DTMC could be lumpable thus enlarging this technique to semi-Markovian processes. Furthermore, it may happen that even in a case of CTMC, the embedded DTMC has a greater reduction factor by lumpability. We have experienced this phenomenon when benchmarking our methods.

2.3 Computation of performance indices.

Let us now recall how it is possible to characterize the performance index (or indices) of interest on a given CTMC, and then discuss the implications of lumping on its computability.

The performance indices of interest can be computed in a transient or steady state setting. Examples of performance indices are the steady state availability of a server, the probability that a given connection be active at time instant τ, or the average number of clients being served in a system.

A general way of defining performance indices on CTMCs is through the use of *reward functions*: their domain is the set S of CTMC states while the co-domain is $\mathbb{R}$. In fact, a function r can be seen as a performance index and, given a (steady state or transient) state probability distribution π, the (average or instantaneous) performance index measure can be expressed as: $\sum_{s\in S} \pi(s) \cdot r(s)$.

If the reward function r expressing the performance index of interest is constant within each aggregate, then the probability distribution of the aggregates is enough to compute the value of the performance index (we can say that the reward function is compatible with the aggregation). However if this is not the case, only exact lumpability still gives us the possibility to compute the performance index value.

Finally observe that the efficient computation of performance indices corresponding to unconstrained reward functions in the exact lumpability case requires a way of efficiently computing the cardinality of each aggregate, and of the subset of states within the aggregate characterized by the same reward function value.

3 The DS and TLS methods

3.1 Lumpability of partially symmetrical MCs

This section presents the *Dynamic Symmetry* (DS) method: it is applied to partially symmetrical MCs.

Partially symmetrical CTMCs. The model of partially symmetrical systems that we develop here is defined as a

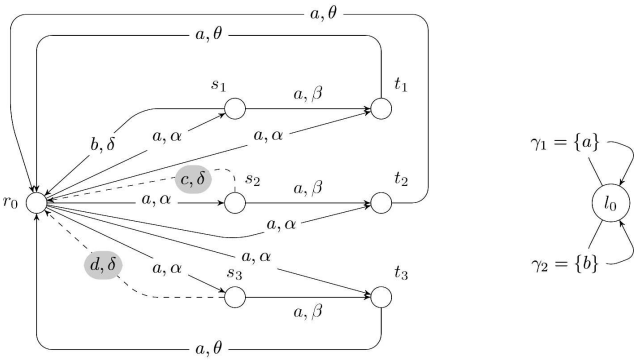

Figure 1. A labeled CTMC and its control automaton

CTMC obtained by some synchronized product between a (symmetrical) CTMC and a control automaton. Let us first formalize this product. Synchronizing the behavior of the two components requires to "label" the CTMC with events.

Notation. Let $\mathcal{C}$ be a CTMC, we associate with each pair of states $s \neq s'$ a label in some alphabet $\Sigma \cup \{\varepsilon\}$, denoted $\Lambda(s, s')$. We require that $\Lambda(s, s') = \varepsilon$ iff $Q(s, s') = 0$.

Since the automaton is introduced in order to modify the behavior of the CTMC, the label of each edge is a predicate that selects the events allowed to occur in the current location of the automaton.

Definition 8 *Let $\mathcal{C}$ be a CTMC, then $\mathcal{A} = \langle L, l_0, \rightarrow \rangle$ a control automaton of $\mathcal{C}$ is defined by:*

- *L, the set of automaton locations,*
- *l_0, the initial location,*
- *$\rightarrow \subseteq L \times 2^{\Sigma} \times L$, the transitions of the automaton.*
  A transition (l, γ, l') will be denoted by $l \xrightarrow{\gamma} l'$.
  Furthermore, if $l \xrightarrow{\gamma} l'$ and $l \xrightarrow{\gamma'} l'$ with $\gamma \neq \gamma'$ then $\gamma \cap \gamma' = \emptyset$.

In standard automata, the last requirement can be easily ensured by merging the two transitions into a single one labeled by $\gamma \cup \gamma'$. However the interest of letting distinct the two transitions will be discussed later.

Fig.1 represents a CTMC and its control automaton. Standard letters are labels, while Greek letters represent transition rates. The initial distribution is: $\pi(r_0) = 1$.

In the synchronized product defined below, the CTMC is the "active" component whereas the automaton is the "passive" component waiting for a transition of the CTMC in order to synchronize it with one of its transitions. Consequently, the rates (resp. the initial distribution) associated with the product depends only on the rates (resp. the initial distribution) of the CTMC.

Definition 9 *Let $\mathcal{C}$ be a CTMC and $\mathcal{A}$ some control automaton of $\mathcal{C}$. The synchronized product of $\mathcal{C}$ and $\mathcal{A}$, $\mathcal{C}_{\mathcal{A}} = \langle S \times L, \pi'_0, Q' \rangle$ is a CTMC defined by:*

- *$\forall s, \pi'_0(s, l_0) = \pi_0(s) \wedge \forall l \neq l_0, \pi'_0(s, l) = 0$*
- *$\forall s \neq s' \in S, \forall l, l' \in L$, if $l \xrightarrow{\gamma} l' \wedge \Lambda(s, s') \in \gamma$ then $Q'((s, l), (s', l')) = Q(s, s')$ else $Q'((s, l), (s', l')) = 0$*
- *$\forall s \in S, \forall l \neq l' \in L$, $Q'((s, l), (s, l')) = 0$*

Remarks. Due to the constraint on the labeling function Λ, a transition with null rate cannot be synchronized with an automaton transition. The requirement related to transitions of the control automaton ensures that given a current location l, a possible next location l' and a label $\alpha \in \Sigma$ (triggered by a transition of the CTMC) there is at most one transition of the automaton that reaches l' from l accepting label α. In realistic applications, the control automaton is only used in order to restrict the behavior of the original CTMC. However observe that the outgoing transition rate of a state (s, l) can be greater than the one of s. Take for instance $\Lambda(s, s') = \alpha$, $l \xrightarrow{\{\alpha\}} l'$ and $l \xrightarrow{\{\alpha\}} l''$ and assume that $Q(s, s) = -Q(s, s')$ (i.e., s' is the only successor of s). Then $Q((s, l), (s, l)) = 2Q(s, s)$ due to the two automaton arcs. We choose this more general setting since for specific applications, it could be useful.

In the example of Fig.1, the control automaton actually forbids transitions that are not labeled with a or b. Hence, $\mathcal{C}_{\mathcal{A}}$ is obtained from $\mathcal{C}$ by removing the dotted arcs. Formally, the states of $\mathcal{C}_{\mathcal{A}}$ are pairs (s_i, l) but as there is only one location in the automaton, we will omit it in the representation of states throughout the example.

From a theoretical point of view, the specification of the system symmetries relies on group theory, applied to the states and the events of the system. The next definition recalls the appropriate notions.

Definition 10 *Let G be a group, with neutral element id and whose internal operation is denoted ($\bullet$). Let E be a set.*

- *An* operation *of G on E is a mapping from $G \times E$ to E s.t. the image of (g, e), denoted by $g.e$, fulfills: $\forall e \in E, id.e = e \wedge \forall g, g' \in G, (g \bullet g').e = g.(g'.e)$*
- *The* isotropy subgroup *of a subset $E' \subseteq E$ is defined by: $G_{E'} = \{g \in G \mid \forall e \in E', g.e \in E'\}$*
- *Let H be a subgroup of G, the* orbit *of e by H denoted $H.e$, is defined by: $\{g.e \mid g \in H\}$. The set of orbits by H defines a partition of E.*

We simultaneously introduce the notions of symmetrical and partially symmetrical CTMCs. Informally, a CTMC is *symmetrical* w.r.t. some group if the operation of the group on the state space preserves its initial distribution and stochastic behavior. A CTMC is *partially symmetrical* if it is a synchronized product of a symmetrical CTMC with a (non symmetrical) control automaton.

Definition 11 *A CTMC $\mathcal{C}$ is symmetrical w.r.t. G a group operating on S and Σ iff:* $\forall g \in G, \forall s \neq s' \in S, \pi_0(g.s) = \pi_0(s) \wedge Q(g.s, g.s') = Q(s, s')$ *and* $\Lambda(g.s, g.s') = g.\Lambda(s, s')$.

Let $\mathcal{C}$ be symmetrical w.r.t. G and $\mathcal{A}$ be a control automaton of $\mathcal{C}$, then $\mathcal{C}_\mathcal{A}$ is said to be partially symmetrical w.r.t. G.

We associate with each γ occurring in a transition of $\mathcal{A}$ a subgroup $H_\gamma \subseteq G$ defined by: $g \in H_\gamma$ iff $\forall a \in \Sigma, a \in \gamma \Leftrightarrow g.a \in \gamma$.

The size of the subgroup H_γ is an indicator of the symmetry of the associated edge. When $H_\gamma = G$, the edge is "fully" symmetrical whilst when $H_\gamma = \{id\}$, the edge is "fully" asymmetrical. Here we see the interest of keeping distinct transitions of the control automaton with same sources and destinations. Indeed when merging them, the subgroup associated with the new transition could be smaller than one of (or even both) the subgroups associated with the original transitions.

Back to the example of Fig.1, let G be the group of permutations of $\{1, 2, 3\}$ generated by binary permutations $p_{i,j}$ which exchange i and j. The operations of G on S and Σ are defined by:
$\forall p_{i,j}, p_{i,j}.r_0 = r_0 \wedge p_{i,j}.a = a$
$\forall p_{i,j}, p_{i,j}.s_i = s_j \wedge p_{i,j}.t_i = t_j$
$\forall p_{i,j}, p_{i,j}.s_j = s_i \wedge p_{i,j}.t_j = t_i$
$\forall p_{i,j}, k \notin \{i, j\}, p_{i,j}.s_k = s_k \wedge p_{i,j}.t_k = t_k$
$p_{1,2}.b = c \wedge p_{1,2}.c = b \wedge p_{1,2}.d = d$
$p_{1,3}.b = d \wedge p_{1,3}.c = c \wedge p_{1,3}.d = b$
$p_{2,3}.b = b \wedge p_{2,3}.c = d \wedge p_{2,3}.d = c$

It is easy to verify that the CTMC is symmetrical w.r.t. G. The subgroups associated with the labels of $\mathcal{A}$ are $H_{\gamma_1} = G$ and $H_{\gamma_2} = \{id, p_{2,3}\}$. Observe that if instead we had merged the transitions, the group would have been $\{id, p_{2,3}\}$ and thus the full symmetry of the edge γ_1 would have been lost.

A subset construction for lumpability. Given a partially symmetrical CTMC $\mathcal{C}_\mathcal{A}$, our method builds a smaller (but equivalent) CTMC based on the building of some "subset" reachability graph that we call $\mathcal{G}_\mathcal{A}$. Algorithm 1 describes its construction.

Let us detail how it works. The nodes of this graph are pairs consisting in a location of $\mathcal{A}$ and a subset of states of $\mathcal{C}$ which equivalently denotes a subset of states $\mathcal{C}_\mathcal{A}$ with same location. An edge of this graph is labeled by a transition $l \xrightarrow{\gamma} l'$ of $\mathcal{A}$ and it represents a (non empty) set of transitions of $\mathcal{C}_\mathcal{A}$. More precisely, such a transition links some state of the source subset to some state of the destination subset that can be reached using $l \xrightarrow{\gamma} l'$.

The key idea of this construction is the following: along any path of this graph (and independently on the instants of transition firings corresponding to the arcs of this path) starting from the initial distribution, the occurrence probability of all states of the subset associated with the last node of this path are identical.

Algorithm 1: Building of $\mathcal{G}_\mathcal{A}$

1: $nodes = \emptyset$; $edges = \emptyset$;
2: Partition $S_0 = \uplus_{i=1}^{n_0} S_{0,i}$
s.t. every $S_{0,i}$ is the orbit of some $s_i \in S_0$ by G;
3: $add\ \perp$ to *nodes*;
4: **for** $i \in \{1, \ldots, n_0\}$ **do**
5: $push(stack, \perp \xrightarrow{init} (l_0, S_{0,i}))$;
6: **end for**
7: **while** *stack* is not empty **do**
8: $(l, R) \xrightarrow{\gamma} (l', R') = pop(stack)$;
9: Compute $\Gamma = \{\gamma' \mid \exists l'' \xrightarrow{\gamma'} l'', \exists s' \in R', \exists s'' \in S, \Lambda(s', s'') \in \gamma'\}$;
10: Compute $H = G_{R'} \cap \bigcap_{\gamma' \in \Gamma} H_{\gamma'}$;
11: Partition $R' = \uplus_{i=1}^{m} R_i$
s.t. every R_i is the orbit of some $r_i \in R'$ by H;
12: **for** $i \in \{1, \ldots, m\}$ **do**
13: **if** $(l', R_i) \in nodes$ **then**
14: $add(l, R) \xrightarrow{\gamma} (l', R_i)$ to *edges*;
15: **else**
16: $add(l', R_i)$ to *nodes*;
17: $add(l, R) \xrightarrow{\gamma} (l', R_i)$ to *edges*;
18: **for** $l' \xrightarrow{\gamma'} l''$ **do**
19: Compute $\mathcal{SECTS} = \{H.s^* \mid \Lambda(r_i, s^*) \in \gamma'\}$;
20: **for** $S' \in \mathcal{SECTS}$ **do**
21: $push(stack, (l', R_i) \xrightarrow{\gamma'} (l'', S'))$;
22: **end for**
23: **end for**
24: **end if**
25: **end for**
26: **end while**

In fact, the construction maintains the following invariants: (1) The graph represents all possible behaviors except the ones that start from some node of the graph with an edge that is present in the stack. (2) The nodes (i.e., the corresponding subset of states) of the graph fulfill all the conditions of corollary 5. (3) The subsets which are destination of an edge in the stack fulfill the two first conditions of corollary 5.

The **while** loop extracts an edge from the stack (line 8). Then it splits the destination subset R' (lines 9-11) in order to ensure the third condition of corollary 5 since inside a subset R_i, the states allow the same transitions of the control automaton. Furthermore, $r_i \in R_i$ is selected. If R_i is a node of the graph (lines 13-14) then one adds the edge to the graph (while preserving the conditions of corollary 5). Otherwise one creates R_i as a new node and the corresponding

incoming edge and one computes the outgoing edges of R_i (lines 18-24). The variable $\mathcal{SET}\mathcal{S}$ contains orbits w.r.t. H reachable from R_i using a transition whose label belongs to γ'. These edges are pushed onto the stack. Again by construction, the destination subsets of states fulfill the two first conditions of corollary 5. Furthermore, the choice of the r_i (line 19) is irrelevant since R_i is the orbit under H of any of its item. So whatever the choice, the set of subsets $\mathcal{SET}\mathcal{S}$ will be identical.

The initial stage consists in partitioning the initial states (S_0) w.r.t. G (line 2). Since there is no incoming edge the two first conditions of corollary 5 are satisfied. We have added a fictitious node $\perp$ in order to handle the $S_{0,i}$ subsets in the main loop (lines 3-6).

In order to prove the soundness of this construction, we first introduce a CTMC $\mathcal{C}_{\mathcal{A}}^{G}$, which is bigger than $\mathcal{C}_{\mathcal{A}}$.

In $\mathcal{C}_{\mathcal{A}}^{G}$, states of $\mathcal{C}_{\mathcal{A}}$ are replicated in instances, and instances are organized w.r.t. the subsets associated with the nodes of $\mathcal{G}_{\mathcal{A}}$. By construction, all the instances that belong to the same subset have the same associated location of the automaton. We will denote (s, l, R) the instance of (s, l) s.t. s belongs to such a subset R. In the next definition, *nodes* (resp. *edges*) refers to the nodes (resp. edges) of $\mathcal{G}_{\mathcal{A}}$.

Definition 12 *Let $\mathcal{C}_{\mathcal{A}}$ be partially symmetrical CTMC w.r.t. G, then the CTMC $\mathcal{C}_{\mathcal{A}}^{G} = \langle S'', \pi_0'', Q'' \rangle$ is defined by:*

- *The set of states S'' is defined by:* $S'' = \{(s, l, R) \mid (l, R) \in nodes \wedge s \in R\}$.
- $\forall i \in \{1, \ldots, n_0\}, \forall R$ *s.t. R is an item of the partition of $S_{0,i}$,* $\forall s \in R$, $\pi_0''(s, l_0, R) = \pi_0'(s, l_0) (= \pi_0(s))$. *For every other* $(s, l, R) \in S''$, $\pi_0''(s, l, R) = 0$.
- $\forall (s, l, R) \neq (s', l', R') \in S''$, *If* $\exists (l, R) \xrightarrow{\gamma} (l', R')$ *is in edges then* $Q''((s, l, R), (s', l', R')) = Q(s, s')$. *Otherwise* $Q''((s, l, R), (s', l', R')) = 0$.

The stochastic process we want to build is obtained by forgetting the instances and only memorizing the subsets.

Definition 13 *Let $\mathcal{C}_{\mathcal{A}}$ be partially symmetrical w.r.t. G, then the stochastic process $(\mathcal{C}_{\mathcal{A}}^{G})^{lp}$ is defined by: $X_t^{lp} = (R, l)$ iff $X_t'' \in \{(s, l, R)\}$.*

The next proposition is the theoretical core of our method. It states that $(\mathcal{C}_{\mathcal{A}}^{G})^{lp}$ is obtained from $\mathcal{C}_{\mathcal{A}}$ by the inverse of a strong followed by an exact lumping.

Proposition 14 *Let $\mathcal{C}_{\mathcal{A}}$ be partially symmetrical w.r.t. G, then:*

- *Denoting $(s_0, l_0) \ldots, (s_n, l_n)$ the state space of $\mathcal{C}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{A}}$ is a strong lumping of $\mathcal{C}_{\mathcal{A}}^{G}$ w.r.t. the partition $\biguplus sl_i$ where $sl_i = \{(s_i, l_i, R) \in S''\}$.*
- *Denoting $\{(R_0, l_0), \ldots, (R_k, l_k)\}$ the state space of $(\mathcal{C}_{\mathcal{A}}^{G})^{lp}$, $(\mathcal{C}_{\mathcal{A}}^{G})^{lp}$ is an exact lumping of $\mathcal{C}_{\mathcal{A}}^{G}$ w.r.t. the partition $\biguplus Rl_i$ where $Rl_i = \{(s, l_i, R_i) \in S''\}$.*

Proof

Let (s, l) be a state of $\mathcal{C}_{\mathcal{A}}$ and let (s, l, R) be an instance of this state in $\mathcal{C}_{\mathcal{A}}^{G}$, we show that there is a bijective mapping from the transitions out of (s, l) onto the transitions out of (s, l, R). So we can suppose that s is examined when looking for successors of (l, R). Then $\exists s', \exists l \xrightarrow{\gamma} l'$ s.t. $\Lambda(s, s') \in \gamma \Leftrightarrow \exists R', \exists s' \in R', \exists l \xrightarrow{\gamma} l'$ s.t. $\Lambda(s, s') \in \gamma$ with (l', R') a successor of (l, R). Since this mapping preserves the rate of the transitions the condition of Prop. 4 for strong lumpability is fulfilled.

Let (s_1, l, R) and (s_2, l, R) be two states of $\mathcal{C}_{\mathcal{A}}^{G}$, we show that there is a bijective mapping from the input transitions of (s_1, l, R) onto the input transitions of (s_2, l, R). Let (v_1, l', R') be such that $\exists l' \xrightarrow{\gamma} l$ and $\Lambda(v_1, s_1) \in \gamma$. Let H be the group of line 10 related to l', R', then $\exists g \in H \subseteq G_{R'} \cap H_{\gamma}$ s.t. $s_2 = g.s_1$. Now define $v_2 = g.v_1$, then $v_2 \in R'$ and $\Lambda(v_2, s_2) \in \gamma$. This implies the existence of the required mapping. Since this mapping preserves the rates of transitions, the two first conditions of corollary 5 for exact lumpability are fulfilled. The third one is ensured by the splitting of line 11 which has produced (l, R). $\diamondsuit$

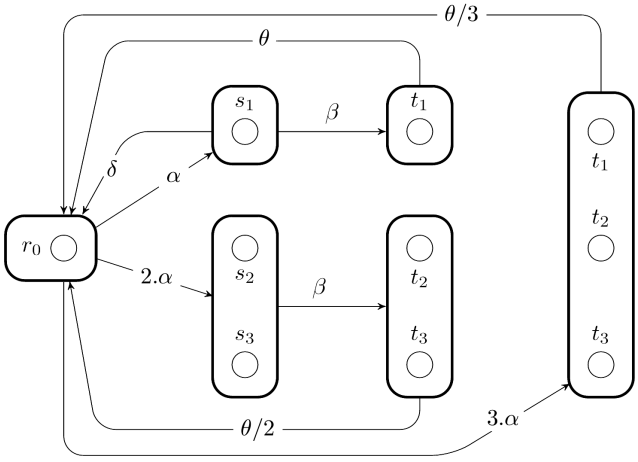

Figure 2. CTMC $(\mathcal{C}_{\mathcal{A}}^{G})^{lp}$

Illustration. We illustrate the algorithm on the CTMC of Fig. 1. The lumped CTMC $(C_{\mathcal{A}}^{G})^{lp}$ is given in Fig. 2. We have represented inside each node the states corresponding to the subset associated with that node. Let us describe the first steps of the algorithm. We push on the stack the edge $\perp \xrightarrow{init} (l_0, \{r_0\})$. When we pick it, we determine that only the automaton transition labeled by a can be synchronized. Thus the subgroup of line 10, H is equal to G. The transition $(r_0, l) \xrightarrow{a} (s_1, l)$ (resp. $(r_0, l) \xrightarrow{a} (t_1, l)$) yields to push on the stack an edge whose destination set is $\{s_1, s_2, s_3\}$ (resp. $\{t_1, t_2, t_3\}$). When the edge with destination $\{s_1, s_2, s_3\}$ is popped, the two transitions of the automaton can be synchronized and thus the group H of line

10 becomes $\{id, p_{2,3}\}$. The orbits of $\{s_1, s_2, s_3\}$ w.r.t. H are $\{s_1\}$ and $\{s_2, s_3\}$. At the end, observe that states t_i appear twice: in $\{t_1, t_2, t_3\}$ and in some orbit of $\{id, p_{2,3}\}$. We can intuitively explain it as follows. When the CTMC reaches directly the states t_i from r_0 then their occurrence is equiprobable which is only the case for t_2 and t_3 when going through s_i.

Our generic method can now be described. Assume first that the CTMC $\mathcal{C}_\mathcal{A}$ associated with the high-level model $\mathcal{M}$ we want to analyze is partially symmetrical. Assume also that we are able to compute directly $(\mathcal{C}_\mathcal{A}^G)^{lp}$ from $\mathcal{M}$. Note π_t the unknown distribution of $\mathcal{C}_\mathcal{A}$ at time t and $\pi_t^{(lp)}$ the (computed) distribution of $(\mathcal{C}_\mathcal{A}^G)^{lp}$ at time t. Then $\pi_t(s,l) = \sum_{s \in R}(1/|R|) \times \pi_t^{(lp)}(R,l)$. The equality also holds for the steady-state distributions.

Although theoretically difficult, we can give some hints of how the space complexity decreases using our approach. In the lumped CTMC, the original states have been substituted by subsets. Note that these subsets may intersect. However these subsets are always the orbit of a state by a subgroup of G. Thus, the larger these subgroups, the better the method. Note that each time a new subset is built, the group is reduced (by intersection with some groups H_γ) and then is enlarged by implicitly substituting to these intersections, the isotropy subgroup of the subset. Interpreting this phenomenon at the model level, we deduce that the complexity reduction factor is high whenever the effect of an asymmetrical event is forgotten in a close future. Experimentations will illustrate this interpretation.

3.2 Lumpability of Almost Symmetrical MCs

In this section we shall define the second method for the (strong or exact) lumpability of a finite CTMC, called *Two-Levels Symmetry* (TLS) method: in this case the starting point is a CTMC and an initial indication of partition of states into aggregates that could potentially already satisfy the lumpability conditions; we call such CTMC and initial partition *almost symmetrical CTMC specification*. A lumpability check algorithm must be applied to such structure, to return a possibly refined partition satisfying the strong and/or exact lumpability conditions.

The *almost symmetrical CTMC specification* contains state transition arcs at two different abstraction levels: the *generic arcs*, expressing transitions between state aggregates, and the instantiated arcs, expressing transitions between CTMC states. Observe that the presence of generic arcs make the symmetries present in the model explicit. Observe also that an efficient implementation of the proposed method must rely on some compact (symbolic) representation of both generic arcs and of state aggregates, that allows to avoid the explicit representation of the corresponding instantiated arcs and states. Of course a way of retrieving such explicit representation must be given, to be used if needed during the refinement.

Definition 15 (Almost symmetrical CTMC) *An almost symmetrical specification of a CTMC $\mathcal{C} = \langle S, \pi_0, Q\rangle$ is defined by:*

- *a partition of the state space $\{S_i\}_{i\in I}$ such that $S = \uplus_{i\in I}\{S_i\}$*
- *two types of state transition arcs:*

  generic arcs*: $i \xrightarrow{\lambda,f} j$ where $i, j \in I$ are the source and destination, $\lambda \in \mathbb{R}_{>0}$ is a rate, and f is a function $f : S_i \to 2^{S_j}$, such that*

  1. $\forall s, s' \in S_i, |f(s)| = |f(s')|$, *and*
  2. $\forall s, s' \in S_j |f^{-1}(\{s\})| = |f^{-1}(\{s'\})|$

  instantiated arcs*: $s \xrightarrow{\lambda} s'$ where s and s' are states and $\lambda \in \mathbb{R}_{>0}$ is a rate.*
- *the infinitesimal generator Q, defined as:*

$$\forall s \in S_i, \forall s' \in S_j, Q[s,s'] = \sum_{i \xrightarrow{\lambda,f} j, s' \in f(s)} \lambda + \sum_{s \xrightarrow{\mu} s'} \mu$$

If there are only generic arcs, then with respect to the given partition, strong lumpability is ensured by condition (1) on generic arcs, while exact lumpability is ensured by condition (2) on generic arcs plus the following initial condition $\forall i \in I, \forall s, s' \in S_i, \pi_0(s) = \pi_0(s')$. In the general case a refinement is required to ensure exact or strong lumpability.

In the sequel an algorithm to check strong/exact lumpability of almost symmetrical CTMCs is given: it is based on (an adaptation of) the Paige and Tarjan's partition refinement algorithm [14, 15] and exploits the properties of generic arcs whenever possible to reduce the number of checks to be performed. It should be emphasized that the achieved lumpable CTMC could have more aggregates than the one obtained without an initial partition constraint, however in the case where the initial aggregates cannot anyway be lumped, then the number of steps of the present algorithm is less than the number of steps required when applying Paige and Tarjan's algorithm directly on the original CTMC. Observe that the reason for imposing an initial partition is due to the efficient (implicit and symbolic) representation of macrostates used in practice to represent the almost symmetrical CTMC, moreover it can be related to the way performance indices are specified (e.g. through a reward function that is forced to have uniform value for all states within the same initial aggregate).

Algorithm 2: Algorithm for the exact lumpability check

```
1  A, X : Set Of Sets of States (SSS);
2  B, D : Set Of States (SS);
3  Lel : Set of tuples ⟨real, integer, S⟩;
4  PartLel : Set of tuples ⟨SS, real, integer⟩;
5  X.Create(AS_CTMC);
6  A = X.PreSplit();
7  while X ≠ A do
8      D = X.Remove() s.t. ∀A_i ∈ A, A_i ≠ D;
9      B = A.Pick(D) s.t. ∃B ⊆ D ⇒ ∀A_i ⊆
       D, |B| ⩾ |A_i|;
10     X.Insert(B);
11     X.Insert(D \ B);
12     Lel = CompAllSucc(B, Part_a);
13     PartLel = Partition_wrt_rate_A(Lel);
14     A.Split(PartLel) ;
15 return A;
```

Algorithm 3: $SSSM :: Split(PartLel)$

```
1  Set, A_i : SSM;
2  for ⟨S, rate, i⟩ ∈ PartLel do
3      Set = ∅;
4      A_i = GetElement(i);
5      Set = A_i \ S;
6      Substitute(i, Set);
7      Add(S);
```

The algorithm for checking exact lumpability A mapping between the *stability condition* of Paige and Tarjan's algorithm and the strong or exact lumpability condition is possible. This is easy for strong lumpability condition, since the stability condition is implied by it. In fact the stability condition requires that all elements in each aggregate reach the same set of destination aggregates, while the strong lumpability condition also requires that they do so with the same rate.

Instead for the exact lumpability the mapping requires to consider the arcs as if they were reversed: when considering reversed arcs, again the stability condition is weaker than the exact lumpability condition. In fact the "reversed arcs" stability condition requires that all elements in each aggregate are reached by the same set of source aggregates while the exact lumpability condition also requires that they do so with the same rate; moreover it is required that the global output rate of states in the same aggregate must be equal.

Like the Paige and Tarjan's Partition refinement algorithm, our extension uses the following data structures:

- A (called Q in the original algorithm, and renamed here to avoid clash with the symbol used for the CTMC infinitesimal generator) is the current partition of states[2]; every element of the list will be called *block*. A single block contains a set of elements of type *Node*. Moreover a *Node* can be a single state iff it represents only one state; or "macrostate" iff it represents an aggregate.

- X represents another possible partition into aggregates, such that A is a refinement of X and A satisfies the lumpability condition with respect to every block of X.

Algorithm 2 shows the pseudo-code of the algorithm. It has two main phases: the initialization (lines 5-6) and the iterative refinement (lines 7-14).

(1) The initial phase. $Create$ initializes the set of X on the basis of the initial partition of the almost symmetrical CTMC specification: for each aggregate having only generic input and output arcs, a new block is inserted into X, containing only one element of type "macrostate". For each aggregate having also instantiated input and/or output arcs a new subset is inserted into X, containing as many elements of type "state" as the states contained in this aggregates. In the simple example of Fig. 3(a), X initially contains three blocks, two of which contain a single element of type "macrostate" (aggregates S_0, S_2), the third contains three elements of type 'state' (states s_1, s_2 and s_3 of aggregate S_1). The notation used in the sequel for X is: $X = \{x_0\{S_0\}, x_1\{s_1, s_2, s_3\}, x_2\{S_2\}\}$.

$PreSplit$ (line 6) returns a refinement set of X, such that each element in this refinement set satisfies the exact lumpability condition with respect to each element of X.

$$\forall s_1, s_2 \in A_i, \sum_{s_k \in X_j} w_{k,1} = \sum_{s_k \in X_j} w_{k,2} \qquad (1)$$

where $w_{k,i}$ represent the rate associated with the arc from s_k to s_i. Observe that this condition will be also the invariant of the interactive refinement phase.

The new refinement set is performed by splitting those sets of X that are reached by one or more instantiated arcs and/or one or more instantiated arcs depart from it. Its splitting is performed considering the weights and source aggregates of the ingoing instantiated transitions, plus the global output rate of each state[3]. Finally (line 6), the new refinement set is stored in A.

[2] It will be clarified later how the initial partition is chosen and how the iterated refinement steps leading to each successive refinement work.

[3] This is sufficient to assure the previous condition in 1, because the

In the simple example the A and X sets after the pre-splitting are: $A = \{a_0\{S_0\}, a_1\{s_1\}, a_2\{S_2\}, a_3\{s_2, s_3\}\}$

(2) The iterative refinement phase. The algorithm core consists of repeating a *refinement step* until X converges to A ($X = A$). The so-called refinement step is performed as follows :

In the partition X, an element D that has been refined in a previous step is selected (line 8), then the largest[4] element $B \in A$ s.t. $B \subseteq D$ is chosen (line 9). Finally, X is updated by replacing D with B and $D \setminus B$ (lines 10-11). In the example, X block x_1 is chosen and the A block a_3 is chosen in the role of B.

All successors of B are computed and the following information are stored in Lel (line 12): the successor, the rate with which it is reached and the A-element index containing it. Then, the function $Partition\_wrt\_rate\_A$ performs a partitioning of Lel grouping the tuples with the same second and third element. In particular all the found "macrostate" elements are instantiated, so that the "macrostate" elements will be substituted by all the states that represent and the generic arcs are instantiated using function f. In the example the "macrostate" S_2, reached by block a_3, is instantiated in s_4, s_5 and s_6, and $PartLel$ is $\{p_1\langle\{s_5, s_6\}, \beta, a_2\rangle\}$ (there is only one set of states, belonging to block a_2, reached by a_3 with rate β).
At this point, A must be refined according to the new partition represented by $PartLel$, as described in Algo 3. In Algo 3, for each element $\langle rate, i, S\rangle \in PartLel$, we remove the elements of S from A_i. Line 6 replaces the old representation of A_i, while line 7 inserts S as a new set in A.

In the example the block a_2 is split in two blocks $a_2\{s_4\}$ and $a_4\{s_5, s_6\}$, so that list A is $\{a_0\{S_0\}, a_1\{s_1\}, a_2\{s_4\}, a_3\{s_2, s_3\}, a_4\{s_5, s_6\}\}$, while list X is $\{x_0\{S_0\}, x_1\{s_1\}, x_2\{s_4, s_5, s_6\}, x_3\{s_2, s_3\}\}$. The refinement is repeated choosing block a_4 in the role of B. This does not cause any new splitting: the final partition is the one of A above as illustrated in Fig. 3(b).

Observe that the algorithm may have to instance some states and arcs that will be aggregated again in the final CTMC, so that the peak of memory usage during execution exceeds the size of the final lumped CTMC, and may limit the applicability of the method.

Algorithm for strong lumpability check. It is easy to adapt the previous algorithm to check the strong lumpability condition instead of the exact lumpability one. In fact only a small part of the previous algorithm must be modified. First we need to modify slightly the pre-splitting phase. In

$$f_1 : f_1(s_0) = \{s_i, i = 1, 2, 3\})$$
$$f_2 : f_2(s_i) = \{s_{i+3}\}$$
$$f_3 : f_3(s_i) = \{s_0\}, i = 4, 5, 6$$

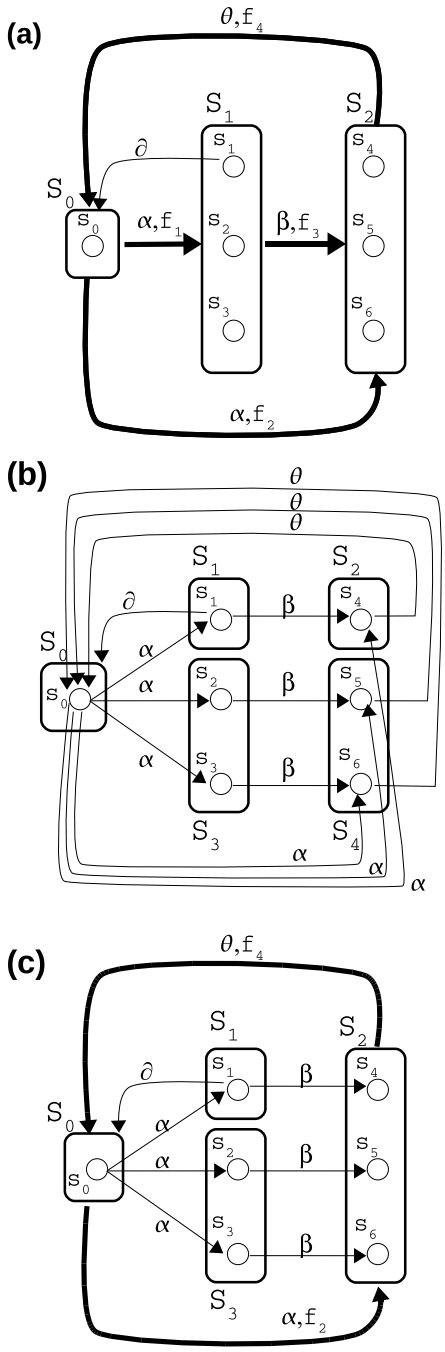

Figure 3. A simple example of almost symmetrical CTMC (a) and the result of exact (b) and strong (c) lumpability algorithm application.

subset that are not reached by one or more instantiated arcs and/or one or more instantiated arcs depart from it do not need refinement, already satisfy 1

[4]In terms of number of contained elements.

line 6 the new refinement set is performed by splitting those sets of X where one or more instantiated arcs depart from them. Fig. 3(a) the pre-split phase for strong lumpability provides the same result as the one already presented for exact lumpability.

In the refinement step only the following change is necessary: after selecting the block B and updating X, for every *Node* element in B we compute the elements reaching it, and the following information are stored in Lel (line 12): the predecessor, the rate with which predecessor reaches it and the A-element index containing predecessor.

After this, the refinement step works similarly to the exact lumpability version.

In the example a_3 is chosen again in the role of B but now the list $PartLel$ is: $PartLel = \{p1\langle\{s_0\}, \alpha, a_0\rangle\}$. This step requires the instantiation of aggregate S_0 and of the generic arc labeled α, f_1. Since S_0 contains only s_0 no split is performed and the algorithm ends. The final partition as illustrated in Fig.3(c)

Comparing the two final partitions (obtained using the two algorithms) we can observe that the strong lumpability condition for this example requires to instance less "macrostate" w.r.t to the exact one. This is not always true, it depends on the characteristics of the model to be studied. For selecting the better one w.r.t. a particular model a first choice must be driven by the performance measures that we want to compute. If probabilities of individual markings (SMs) are needed, then the strong lumpability cannot be used since it only gives the probabilities of aggregates Instead if the performance measures can be expressed at the level of aggregates, then both approaches are available and an heuristic rule for defining the approach minimizing the number of instanced "macrostate" can be used.

Lumped Markov chain generation. In this section, we explain how to generate the lumped *CTMC* from the refined structure. Let us start with the case of a refined structure satisfying the strong lumpability condition; the $Q^{lp}(i,j)$ is computed as follows (see Proposition 6):

$$Q^{lp}(i,j) = \Big(\sum_{S_i \xrightarrow{\lambda,f} S_j} \lambda |f(s)| \Big) + \Big(\sum_{s' \in S_j : s \xrightarrow{\mu} s'} \mu \Big)$$

where s, in both terms, denotes any state belonging to aggregate S_i.

The computation of the infinitesimal generator of the lumped CTMC in the case of exact lumpability is instead performed as follows (see Proposition 7):

$$Q^{lp}(i,j) = \Big(\frac{|S_j|}{|orig(S_j)|} \sum_{S_i \xrightarrow{\lambda,f} S_j} \lambda |f(s)| \Big)$$

$$+ \Big(\frac{|S_j|}{|S_i|} \sum_{s \in S_i : s \xrightarrow{\mu} s'} \mu \Big)$$

where $s \in S_i$ and $s' \in S_j$, and $orig(S_j)$ denotes the aggregate to which states in S_j belonged in the initial almost symmetrical CTMC. This is needed because there might be generic arcs connecting a non split aggregate to an aggregate that has been split (there will be a replica of such generic arc for each refined aggregate substituting the original one).

In the examples of Figg. 3(b) and 3(c) the instantiated arcs created during the algorithm execution are shown instead of the arcs in the final lumped CTMC, to illustrate the algorithm operation and the memory peak problem. There will be only one arc between each pair of aggregates in the lumped CTMC, whose rate can be easily derived from the above formulae: e.g. in Fig. 3(b) the rate from S_0 to S_4 is 2α, the rate from S_3 to S_4 is β, in Fig. 3(c) the rate from S_0 to S_2 is 3α, from S_2 to S_0 is θ, from S_0 to S_3 is 2α.

4 Implementation issues and comparison

Both methods handle sets of states which require a symbolic representation to efficiently manage them and a symbolic computation of the set of successors. Decision Diagrams (DD) could be used to this aim. However DD would not take into account that these sets are somewhat special since they are orbits of subgroups. By contrast, some formalisms are tailored to take advantage of the symmetries during the modeling and the analysis stages.

Thus we have implemented our methods using the SWN formalism [4], a kind of high-level Petri nets that has been the starting point of numerous efficient symmetry-based analysis methods already implemented in the tool GreatSPN [5]. Here we only describe the main features of SWNs. In a SWN, a *color domain* is associated with places and transitions. The colors of a place label the tokens contained in this place, whereas the colors of a transition define different ways of firing it. In order to specify these firings, a *color function* is attached to every arc which, given a color of the transition connected to the arc, determines the colored tokens that will be added to or removed from the corresponding place. Finally the initial marking is defined by a multi-set of colored tokens in each place. A color domain is a Cartesian product of *color classes* which may be viewed as primitive domains. A class can be divided into *static subclasses*. The colors of a class have the same nature

(e.g. processes) whereas the colors inside a static subclass have the same potential behavior (e.g. batch processes). A color function is built by standard operations (linear combination, composition, etc.) on predefined basic functions. In the case study, we use a single kind of basic function: a projection which selects an item of a tuple and is denoted by a typed variable (e.g., x, y). Transitions can be guarded by expressions. An expression is a Boolean combination of predefined atomic predicates like $[x \neq y]$.

The implicit symmetry of an SWN, obtained by its restrictive syntax, leads to a group G operating on color classes (and by extension on markings and firing instances). G is the intersection of the isotropy subgroups of static subclasses. In other words, any permutation in G maps any static subclass onto itself. Given a marking m and a permutation g of G, the behavior of the net from the marking $g.m$ is the same as the behavior from m up to permutation g. The Symbolic Reachability Graph (SRG) construction lies on symbolic markings, namely a compact representation for a set of equivalent ordinary markings. A symbolic marking is a generic representation, where the actual identity of tokens is forgotten and only their distributions among places are stored. Tokens with the same distribution and belonging to the same static subclass are grouped into a so-called *dynamic subclass*. Then, the SRG can be automatically built using a symbolic firing rule that directly applies on symbolic markings [4].

The critical factor for efficiency of the SRG method is the partition of a class into static subclasses. Finer is the partition, less effective is the reduction of space. Thus the implementation of both DS and TLS methods aims at keeping this partition as coarse as possible.

In order to implement the DS method for a partially symmetrical CTMC, we specify this chain as the synchronized product of an SWN without static subclasses (so, G is the group of all permutations on color classes) representing the symmetrical MC, and a control automaton whose labels are set of instances of transitions like $\{t(a,b), t(b,b), t(a,a), t(b,a)\}$ equivalently denoted $\bigvee_{x,y\in\{a,b\}} t(x,y)$. Thus the isotropy subgroup of a transition may be represented by a "local partition" in static subclasses (e.g. $\{a,b\},\{c,d\}$ for the label described above). The symbolic representation of a state of the lumped MC is then given by a local partition of color classes (roughly corresponding to the isotropy subgroup of the set of associated states), a symbolic marking w.r.t. this partition, and a state of the automaton. The symbolic firing rule is close to the original one except that a refinement w.r.t. to the partitions of the synchronized transitions must be *a priori* performed, and a merging of static subclasses must be *a posteriori* performed in order to represent the isotropy subgroup of the new set of states. The graph which is built is called Dynamical SRG (DRSG), emphasizing that the partition in static subclasses depends on each node.

In order to implement the TLS method for an almost symmetrical CTMC, we specify this chain as an SWN where the transitions are split in symmetrical transitions, whose specification does not depend on static subclasses, and asymmetrical ones whose specification depends on them. An almost symmetrical CTMC is then generated from this specification. It corresponds to the Extended SRG (ESRG) [9] whose main feature is that a node has a two-level representation. At the higher level, a node is a symbolic marking w.r.t. the SWN without static subclasses: this symbolic marking is enough to check and fire symmetrical transitions. At the lower level, the symbolic marking is a substituted by a set of symbolic markings taking into account the static subclasses partitions allowing to check and fire asymmetrical transitions. These two representations correspond to the same set of ordinary markings. The aim of the ESRG construction is to avoid developing the lower level representation for nodes as often as possible. This can be done when all ordinary makings of the node are known to be reachable and when none allows an asymmetrical firing (these conditions can be symbolically checked). Thus, the ESRG is the starting point of our adaptation of the Paige-Tarjan algorithm where some avoided lower level representations are now developed in order to meet the lumpability requirements.

We can compare the proposed methods w.r.t. different criteria. The size of the lumped chain is generally smaller with the TLS method as it is based on the minimization of a MC w.r.t. lumpability. However the TLS method requires to explicitly develop "asymmetrical" set of states during the refinement process thus facing the problem of a peak in memory usage, contrary to the DS method. On the one hand, when a model is efficiently handled by the methods, the final sizes are of the same magnitude order. On the other hand, when the asymmetry of the model propagates throughout the state space, it may yield a combinatorial explosion and the size of the lumped chain of the DS method may become bigger than the original one. This cannot happen by construction with the TLS method. Finally, the DS method is parametrized in the following sense: as lumping is based on labels the modeler can freely change the numerical values associated with labels without need to recompute the graph associated with the lumped chain; only the numerical values have to be updated. In order for the TLS method to support such a parametrization the refinement algorithm must be based on transition labels instead of rate values.

5 Case study

5.1 Modeling a client-server architecture

In this section a client-server architecture model is introduced: the goal is to evaluate the impact of introducing and managing different priorities among user classes on the

efficiency of the system. Users in different classes behave similarly: the asymmetry is only due to the priority rules described later. Fig. 4 shows an SWN model of the system with priority among user classes; however except for one subnet, all parts of the model with or without priority are identical. So we will explain how a completely symmetric model can be derived from that of Fig. 4. Shaded boxes, labeled N_i, highlight the main parts. In the sequel these labels are used to refer to each subnet.

The system is composed of a finite number of terminals and a Remote Terminal Server (RTS). In subnet N_1, the initial marking of place $Clients$ corresponds to the number of terminals. Via a terminal, a client tries to open a connection with the RTS. This connection is accepted if the maximum load of the RTS has not been reached yet, then it is authenticated. The maximum load is given by the initial marking of place $MaxReq$. The authentication is performed within subnet N_2. Variable x associated with transition $AuthOk$, memorizes the *user class* of the client.

Once authenticated, a client asks for a service that can be *non-critical* (e.g. a read transaction) or *critical* (e.g. a write transaction). Non-critical services can be handled simultaneously (inside subnet N_3) while a critical service must be performed in mutual exclusion with any other service. The system ensures a weak priority for non-critical services based on a *wave* mechanism. The wave consists of the clients currently accepted by the RTS. Once a client chooses a critical service (transition $ChCs$) accepted by the RTS (transition $AccCs$), no more clients can join the wave (inhibitor arc from place $Wave$ to transition $AccR$). Critical services are performed only when there are no more clients in the authentication stage or in a non-critical service execution. Place $NbReq$ is used to control this requirement. When the last critical service of the wave completes, a new wave can start.

Subnet N_3 models the handling of a non-critical service. A service identity (variable i) is attached to the two parallel tasks that perform the service in order for them to synchronize at the end (transition $eNCs$).

For efficiency reasons, during a wave the RTS accepts a limited number of different concurrent user classes (initial marking of place $MaxQueues$) in the critical services. This management is modeled by subnet N_4. A critical service request related to a user class not already in competition (i.e., without its colour in place $Queues$) is rejected (transition Rej) if the maximum number of concurrent user classes has been reached.

A critical service is divided into two sequential stages: a preprocessing step that can be performed concurrently and a main step that is performed in mutual execution (see subnet N_6). If a priority rule is applied then the requests access the critical section following the order of the user classes. Observe that in this case the first critical service that has achieved its preprocessing step must wait if it does not belong to the highest priority user class in competition (see subnet N_5). If there is no priority between user classes then the access to the critical section is granted as soon as possible after having completed the preprocessing phase. This can be represented by a fully symmetric net obtained by deleting subnet N_5.

From a modeling point of view, the priority among user classes is managed in N_5 using a *swap* mechanism based on the asymmetrically *guarded (*$[y < x]$*)* transition $Swap$. It ensures that place $Elected$ always contains the highest user class of the remaining critical services requests. In order to guarantee that $Swap$ is always performed before allowing the next critical section entry, transition $Swap$ is given the highest priority (this is denoted $prio_3$: we consider nets with different transition priorities). In order to apply the the DS method we model this guard by a *control automaton* with one state and two loops: one for all transition instances except $Swap$ and one labeled by the set $\{Swap(c, c')\}$ with $c < c'$. With respect to the partition of transitions related to the TLS method the only asymmetrical transition is $Swap$ (whose guard is actually expressed as a boolean formula involving basic predicates based on the partition of the users color class into cardinality one static subclasses).

From a quantitative point of view, the expected performance behavior of the system with priorities is a decrease of the global throughput of the system (w.r.t. the version without priorities), due to the fact that the users in the highest priority class must be served first when they require a critical service; on the other hand we expect that the high priority user class will experience a smaller critical-service time than in the symmetric system. Hence we are interested to evaluate the throughput vs. service time trade off.

5.2 Performance analysis

To study and compare the two discussed service policies, we must introduce and compute some performance indices: the throughput of the server ($\overline{X}_{CS+NS}$); the average service time for Non-critical Service requests ($\overline{T}_{NS}$); the average service time for Critical Service requests of the i^{th} user class ($\overline{T}_{CS_i}$). Then, we compare the global (resp. the user class dependent) average critical service time of the RTS according to each policy (with or without priority).

As long as we perform the analysis on a lumped MC based on the exact lumpability condition, there is no loss of information on the state probabilities. Hence, we can compute not only the global performance indices, but also the user class dependent ones. In this context Lumped MCs can be obtained either applying DS or TLS.

Besides computing the performance indices according to the initial goal, we compare the behavior (in terms of memory consumption) of the two methods and of the traditional

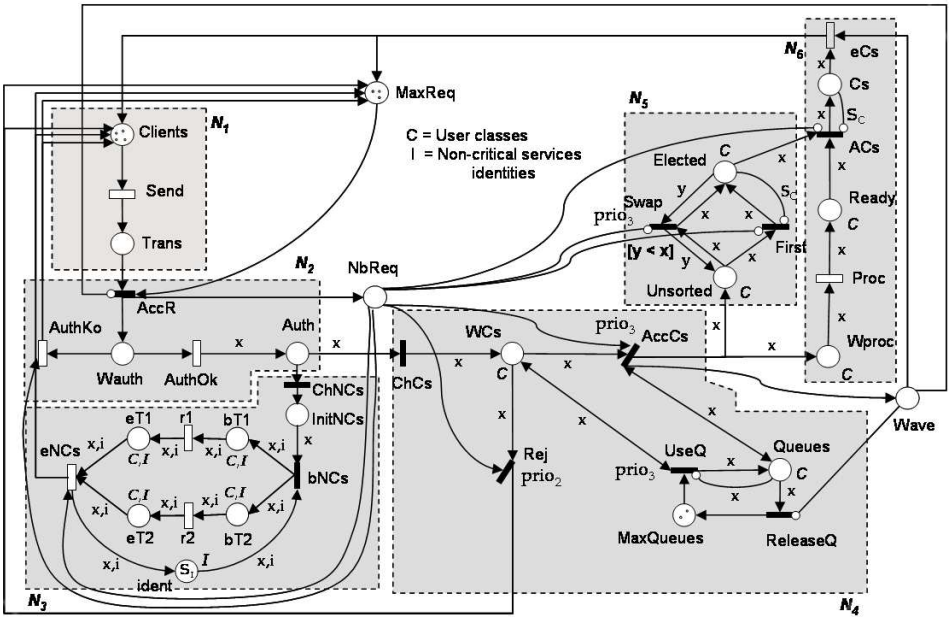

Figure 4. SWN of the Client/Server System.

SRG-based lumping technique [5] Also, a reference point is given by the size of the Ordinary MC (OMC).

Table 1 shows some of the computed performance indices for different configurations, expressed by the values of four parameters: *Cli*, *maxR*, *maxQ* and *Prio* (listed in the first columns). (*Cli*) is the initial markings of place $Clients$ (the numbers of clients); (*maxR*) the initial marking of place $MaxReq$ (the maximum load of the RTS); (*maxQ*) the initial marking of place $MaxQueues$ (the maximum allowed number of different user classes). Column $Prio$ gives the number of user classes. The maximum number of non-critical service requests that can be concurrent in the server is fixed to 3 (not explicitly represented).

The other columns are each labeled with the corresponding performance index introduced earlier in this section. The pairs of values in the column labeled $gain/loss(\%)$ represents the gain (resp. loss) for the highest (resp. lowest) priority user class on the average critical service time in the asymmetric model w.r.t the symmetric one. As expected, the decrease of the throughput and the increase of the average critical service time in the asymmetric RTS is counterbalanced by an acceleration in the treatment of the highest priority user class. This phenomenon is emphasized when increasing the number of the clients, and the number of user classes.

Table 2 summarizes some results on the state space reduction induced by different analysis methods, obtained on our asymmetric system with different values of the parameters. The first columns express again the parameter values (*Cli*, *maxR*, *maxQ* and $Prio$). Again the maximum number of non-critical service requests that can be concurrent in the server is 3. For each method, column $St.$ (resp. T.) gives the number of states (resp. the computation time) of the (lumped) MC.[6] Column $Peak$ represents the maximum number of intermediate states constructed while applying the TLS method.

We observe a significant reduction (exponential savings) achieved by our two methods w.r.t. the OMC and the SRG. Moreover, we remark that DS and TLS behave similarly, in the sense that the number of constructed states using DS is always situated between the final number of states and the peak of TLS. It is worth noting that the (premature) explosion for the TLS method is not caused by its intrinsic behavior: for the shown cases, it was impossible to generate the *almost symmetrical CTMC* constituting its input (i.e., the ESRG structure). Actually, the ESRG construction algorithm suffers from a memory peak problem similar to that discussed for the TLS method.

Although theoretically difficult to prove, we can intuitively say that the two methods are complementary and their respective efficiency heavily depends on the presence of *strong synchronization points* between symmetric and asymmetric behaviors of the system. These are regeneration points w.r.t the consequences of any asymmetric transition occurrence and block the propagation of states instantiation and splitting in the algorithms.

[5] We recall here that the SRG satisfies both strong and exact lumpability conditions.

[6] The construction time of OMC is not given because the results are evaluated based on the SRG outputs.

				Asymmetric model			Symmetric model			$gain/loss(\%)$
Cli	maxR	maxQ	Prio	$\overline{X}_{CS+NS}$	$\overline{T}_{NS}$	$\overline{T}_{CS}$	$\overline{X}_{CS+NS}$	$\overline{T}_{NS}$	$\overline{T}_{CS}$	
3	2	2	3	0.51305	2.5	2.782178	0.51570	2.5	2.726234	(1.42, -5,52)
3	2	2	5	0.53829	2.5	2.819253	0.54189	2.5	2.750577	(1.74, -6.73)
3	2	2	8	0.55373	2.5	2.842139	0.55795	2.5	2.766257	(1.93, -7.41)
3	3	3	3	0.53508	2.5	2.963033	0.53816	2.5	2.902267	(1.97, -6,16)
3	3	3	5	0.55919	2.5	2.986704	0.56324	2.5	2.913987	(2.25, -7.24)
3	3	3	8	0.57336	2.5	2.994600	0.57800	2.5	2.915846	(2.33, -7.72)
8	2	2	3	0.55206	2.5	2.971734	0.55636	2.5	2.887827	(1.94, -7.75)
8	2	2	5	0.57223	2.5	3.022436	0.57814	2.5	2.915278	(2.45, -9.80)
8	2	2	8	0.58430	2.5	3.054844	0.59132	2.5	2.932937	(2.77, -11.08)
8	3	3	3	0.63419	2.5	3.682498	0.64124	2.5	3.566985	(4.83, -11.26)
8	3	3	5	0.65524	2.5	3.779566	0.66459	2.5	3.633744	(6.19, -14.06)
8	3	3	8	0.66833	2.5	3.842901	0.67916	2.5	3.678659	(7.04, -15.86)

Table 1. Performance indices for asymmetrical and symmetrical models

				OMC	SRG		DS		TLS		
Cli	maxR	maxQ	Prio	St.	St.	T.	St.	T.	St.	Peak	T.
3	3	2	3	16547	4533	1.59	1071	1.47	998	1160	0.14
3	3	2	5	69638	17708	9.38	1217	2.11	998	1727	0.20
3	3	2	8	269732	65268	48.75	1436	4.30	998	3185	0.50
8	4	2	3	529061	122771	61.88	24156	45.51	23457	25221	0.77
8	4	2	5	3591686	759796	592.02	26884	69.93	24787	32725	1.72
8	4	2	8	21835811	4358051	5143.14	28981	101.66	24787	48601	5.59
8	5	3	3	2241462	496618	282.09	94593	187.98	92600	96088	3.04
8	5	3	5	23568689	4788499	4206.34	134553	492.49	110376	159104	13.98
8	5	3	8	–	–	–	193401	4321.32	–	–	–
8	5	4	3	2241462	496618	282.09	94593	187.98	92600	96088	3.04
8	5	4	5	23656834	4838244	4226.15	153180	649.24	114442	178690	21.09
8	5	4	8	–	–	–	346649	39063.66	–	–	–
8	6	3	3	7168981	1543595	952.74	280296	574.6	276658	282250	7.31
8	6	3	5	111146711	21863821	21235.84	438194	1724.53	388521	483119	32.22
8	6	3	8	–	–	–	563975	10535.30	–	–	–

Table 2. Benchmark model showing the efficiency of the different methods.

6 Conclusion and future work

In this paper the DS and TLS methods have been presented: their goal is to generate a lumped CTMC from a partially symmetrical and almost symmetrical CTMC specification respectively. They can be efficiently applied only if the MCs on which they operate can be handled symbolically, exploiting the a priori known presence of symmetries: this happens when they are derived from a higher level model, such as SWNs, where the presence of similarly behaving components is made explicit. Implementation issues have also been discussed referring to SWNs. A realistic performance evaluation case study has been presented and the effectiveness of the two methods was illustrated through it.

In perspective, it is interesting to achieve a characterization of the type of models that can fully exploit the potential of the presented methods, possibly based on structural properties of the high level model. Another possible line of development is to complement these methods with the possibility of computing bounds on the performance indices by transforming the partially symmetrical MC into a symmetrical one, and using stochastic ordering arguments.

Finally the presentation of the methods in a general setting can be a good starting point to extend their application to other high level formalisms able to highlight the presence of similarly behaving components.

References

[1] S. Baarir, S. Haddad, and J.-M. Ilié. Exploiting Partial Symmetries in Well-formed nets for the Reachability and the Linear Time Model Checking Problems. In *Proc. of WODES'04*, Reims - France, 2004. Springer Verlag.

[2] Baarir S., Dutheillet C., Haddad S., and Ilié J-M. On the use of exact lumpability in partially symmetrical well-formed nets. In *Proc. of QEST'05*, pages 23–32, Torino - Italy, Sept. 2005. IEEE C.S. press.

[3] M. Beccuti, S. Baarir, G. Franceschinis, and J.-M. Iliè. Efficient lumpability check in partially symmetric systems. In *3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 211–221, Riverside, CA, USA, September 2006. IEEE Computer Society.

[4] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transactions on Computers*, 42(11):1343–1360, nov 1993.

[5] G. Chiola, G. Franceshinis, R. Gaeta, and M. Ribaudo. GreatSPN1.7: GRaphical Editor and Analyzer for Timed and Stochastic Petri Nets. *Performance Evaluation, North Holland Journal*, 24, 1997.

[6] E. Emerson and A. Prasad Sistla. Symmetry and Model Checking. *FMSD'96*, 9:307–309, 1996.

[7] E. A. Emerson and R. J. Trefler. From Asymmetry to Full Symmetry: New Techniques For Symmetry Reduction in Model Checking. In *Proc. of CHARME'99*, LNCS, pages 142–156, Bad Herrenalb - Germany, Sept. 1999. Springer Verlag.

[8] S. Haddad, J. Ilié, and K. Ajami. A model checking method for partially symmetric systems. In *Proceedings of FORTE/PSTV'00*, pages 121–136, Pisa, Italy, Oct. 2000. Kluwer Academic Publishers.

[9] S. Haddad, J. Ilié, M. Taghelit, and B. Zouari. Symbolic Reachability Graph and Partial Symmetries. In *Proc. of the* 16^{th} *ICATPN*, volume 935 of *LNCS*, pages 238–257, Turin, Italy, June 1995. Springer Verlag.

[10] Huber P., J. Jepsen L.O., and Jensen K. Towards Reachability Trees for High Level Petri Nets. In *Proc. of EWATPN*, Aarhus, Denmark, June 1984.

[11] C. N. Ip and D. L. Dill. Better verification through symmetry. *FMSD*, 9(1/2):41–75, 1996.

[12] J. Kemeny and J. Snell. Finite Markov chains. New York, NY, 1960. D. Van Nostrand-Reinhold.

[13] J. Ledoux. Weak lumpability of finite markov chains and positive invariance of cones. Technical report, IRISA, 1996.

[14] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM J.* Comput., 16 (6)(6):973–989, 1987.

[15] S. Derisavi, H. Hermanns, and W. H. Sanders. Optimal State-Space Lumping in Markov Chains. *Information Processing Letters*, 87 n.6(6):309–315, 2003.

[16] W. Sanders and J. Meyer. Reduced Base Model Construction Methods for Stochastic Activity Networks. In *Proc. International Conference on Petri Nets and Performance Models*, pages 74–84, Kyoto,Japan, December 1989.

[17] P. J. Schweitzer. Aggregation methods for large Markov chains. In *Proc. of IWCPR*, pages 275–286. North-Holland, 1984.