Efficient lumpability check in partially symmetric systems

M. Beccuti  
G. Franceschinis
Dip.Informatica, Univ. del Piemonte Orientale  
Alessandria, Italy
giuliana.franceschinis@mfn.unipmn.it

S. Baarir  
J-M. Ilié
LIP6, Univ. Pierre et Marie Curie  
Paris, France
jean-michel.ilie@lip6.fr

Abstract

State space based performance analysis of stochastic models may be impaired by the state space explosion but such problem can be mitigated in symmetrical behaving systems by aggregating equivalent states and transitions.

An effective way of exploiting symmetries when the system is modeled using the Stochastic Well-Formed Net (SWN) formalism, is to generate the Symbolic Reachability Graph (SRG) and automatically derive a lumped Continuous Time Markov Chain (CTMC) of the same size as the SRG from it. For partially symmetric systems, the Extended SRG (ESRG) can be used instead, but the derivation of a lumped CTMC in this case is not as direct as in the SRG case: in fact the ESRG structure might need a refinement to satisfy the lumpability conditions.

In this paper a new efficient algorithm to derive a lumped CTMC from the ESRG is presented, and the results obtained by experimenting its implementation within the GreatSPN environment are discussed. The algorithm combines the Paige and Tarjan’s partition refinement algorithm (extended to work with weighted arcs) and a previously proposed lumpability check algorithm (built specifically for the use with the ESRG) and outperforms both of them. The implementation of the algorithm within the GreatSPN environment will allow the several users that have chosen this package to apply the proposed technique.

1 Introduction

The behavior in time of a model expressed through the Stochastic High Level Petri Net formalism called Stochastic Well-formed Nets (SWN) [1] can be described by means of a Continuous Time Markov Chain (CTMC) isomorphic to its reachability graph. The SWN formalism imposes some constraints on the syntax of the color domains, arc expressions and transition guards, leading to the possibility of automatically discovering and exploiting behavioral symmetries to obtain a reduced Reachability Graph called Symbolic RG (SRG) [2] and the corresponding (lumped) CTMC [1]; the aggregate states of the SRG are called Symbolic Markings (SM).

The interesting feature of the SRG is that it allows to obtained most qualitative properties as well as performance measures derivable from the original RG and CTMC.

The Extended SRG (ESRG) [3] structure has been introduced to take advantage of partial symmetries, a possibility that is not available with the SRG. In systems with mostly symmetric behavior and occasional, local asymmetric behavior, the state aggregation induced by the ESRG can be significantly higher than that induced by the SRG: in fact the ESRG groups into Extended SMs (ESM) sets of (partially) similar SMs, moreover when the behavior is locally symmetric the set of SMs captured in an ESM can be kept implicit and represented by means of a unique Symmetric Representation (SR). Even when the ESM groups SMs that must be explicitly represented because of local asymmetry, the common SR can be factorized, still allowing a space saving.

The derivation of a CTMC from the ESRG is not straightforward: in fact the aggregation of SMs suggested by the ESRG does not always satisfy the strong lumpability condition. Hence an algorithm must be devised to find the coarsest ESRG refinement that satisfies the condition. One such algorithm was proposed in [4]. In this paper a new algorithm is proposed, based on the Paige and Tarjan’s partition refinement algorithm and exploiting the information contained in the ESRG: in [5] it has been proved that it is possible to extend the Paige and Tarjan algorithm to CTMC lumping keeping the same complexity of $O(m \log n)$. The modification of the algorithm based on the ESRG allows us to save time, because fewer steps are performed, and also

*This work has been partially supported by Italian MIUR under grant FIRB-Perf.
to save space in all those models where a significant number of ESM can be described by their SR only. Even if the single steps of the ESRG-based refinement may require the "instantiation" of the SMs grouped in an ESM for performing the lumpability check, this additional cost is needed at most once for each ESM, and it is a cost that would be paid earlier (during the SRG construction process) if the ESRG was not adopted.

The paper is organized as follows: in Sec. 2 the basic definitions and notation used in the next sections are introduced, Sec. 3 presents the algorithms for lumpability check and for the lumped CTMC generation. Sec. 4 discusses the algorithm implementation and presents the results obtained on some examples, trying to characterize the class of systems that can benefit from this approach. Sec. 5 compares the approach proposed in this paper with those presented in the related literature, and provides some information on its implementation. Finally Sec. 6 concludes the paper and proposes possible future works.

2 Basic Definition and Notation

In this section, we recall the basic concepts of the SWN formalism and the corresponding approach to the SRG construction, then we discuss how the level of states aggregation can be improved by introducing the ESRG, and the relation between the ESRG and a lumped CTMC of size less than or equal to the SRG. We assume that the reader is familiar with high level Petri nets in general and with the SWN formalism in particular, which can be seen as a special form of Colored Petri Nets. For the sake of simplicity and conciseness, a single color class is considered, however this is not a limitation of the formalism, but it simplifies the presentation. For a more formal presentation of the SWN and the SRG, the reader can refer to [1, 2].

2.1 The symbolic approach of SWN

The symbolic approach used for building the SRG or the ESRG is based on the systematic and automatic exploitation of symmetries, starting from an SWN model. This is achieved by lifting the detail level of the state description (ordinary marking) to a more abstract one, called Symbolic Marking (SM). The basic idea is to replace token colors with symbols which stand for any color in a given color class. The resulting representation can be interpreted as a pattern common to all ordinary markings represented by that SM. A more efficient representation can be obtained when a set of \( k \) different symbols in an SM appear with the same multiplicity in all places: in fact in this case these symbols can be substituted (in all places) by a unique symbol \( Z^j \), representing the \( j^{th} \) set of symbols, of cardinality \( |Z^j| = k \).

Let’s refer to the SWN depicted in Fig. 1: this net models a Distributed Critical Section (DCS) algorithm. A color class \( C = \{pr_1, pr_2, pr_3\} \) is introduced to represent a finite set of processes and is used to specify the color domain of each place. Idle processes, initially represented by the corresponding color in place \( ID \), may independently issue a request in \( RQ \) for accessing the critical section \( CS \). The solution of multi-access conflicts is managed in \( GS \) so that a single request is accepted (by transition \( t_3 \)) while the others are all rejected (this task is accomplished by transition \( t_4 \)), while the immediate transition \( t_6 \) is used to prevent any other new request submission. One example of SM for this net is \( \hat{m} = ID(Z^1) + GS(Z^2) + FDR(Z^3) \), where \( |Z^1| = 2, |Z^2| = 1 \). It represents the set of three ordinary markings such that two (arbitrarily chosen) processes are in place \( ID \), and the other one in \( GS \); the SM represents several ordinary markings, e.g. \( m = ID(pr_1 + pr_3) + GS(pr_2) + FDR(pr_1 + pr_3) \). The symbols \( Z^j \) just introduced are called dynamic subclasses, and stand for any subset of colors (of cardinality \( |Z^j| \)) in \( C \). The SM representation is based on a symbolic partition of the color classes in dynamic subclasses, and it is a very compact way of representing a set of equivalent ordinary markings. In this framework two ordinary markings are equivalent if they can be obtained from each other through the permutation of colors in the same class.

The construction of the SRG is based on the SM definition and on a symbolic firing rule directly applicable to a SM representation and a symbolic transition instance. It is similar to the ordinary one except that the variables associated with the transition are bound to dynamic subclasses rather than specific colors. For example from the above SM \( \hat{m} \) where \( \langle Z^2 \rangle \) marks place \( GS \) with \( |Z^2| = 1 \), transition \( t_3 \) is enabled for the symbolic instance: \( p = Z^2 \), hence, \( \{t_3, \langle Z^2 \rangle\} \) can be fired yielding the new SM: \( ID(Z^1) + CS(Z^2) \) (the interpretation being that whatever the identity of the requesting process, it can move on to the \( CS \)).
Most properties that can be checked on the ordinary RG can also be checked on the SRG:

**Property 1** All the ordinary markings represented by a reachable SM are reachable. Moreover, if two SMs are connected in the SRG, all the ordinary markings of the source SM reach the same number of ordinary markings of the target SM in the RG.

The SRG technique is very efficient if the modeled system has a symmetric behavior, i.e. if the entities represented by colors in the same class behave all in the same way; if instead the entities represented by colors in the same class do not always behave in the same way, asymmetry arises. In the SWN formalism, any asymmetric behavior must be expressed by partitioning the color classes into subsets called static subclasses, so that objects with different behaviors belong to different subclasses. At the model level the static subclasses are mainly used to restrict the enabling of transitions (e.g. some variable can be bound only to colors belonging to a given static subclass) or to restrict the domain of color functions on arcs.

A transition is called asymmetric if its guard or its arc functions explicitly refer to some static subclass, otherwise it is called symmetric. In our example, the CS access conflict is solved by means of a privilege given to the process of higher identity among the requesters. In the figure, this is expressed by associating a guard \( [p > q] \) with transition \( t_4 \). To express this in the syntax of SWN, the class \( C \) must be split in three static subclasses of cardinality one: \( C_i = \{ p_{i1} \} \), \( i = 1, 2, 3 \), and transition \( t_4 \) has an associated guard in the form: \( \text{OR}_{i>j} ([p \in C_i] \land [q \in C_j]) \). When static subclasses are introduced, the dynamic subclasses of SMs express a (dynamic) partition of colors in static subclasses. This prevents “confusion” of colors belonging to different static subclasses. In this case two ordinary markings are equivalent if they can be obtained from each other by permutation of elements within the same static subclass. Let us explain through an example what is meant by “confusion of colors”; the symbolic marking \( RQ(Z^1) + GS(Z^2) \), where \( |Z^1| = 1, |Z^2| = 2 \), apparently enables (an instance of) transition \( t_4 \), however it is impossible to establish if the transition guard is true if we do not know the static subclass to which the two elements in \( Z^2 \) belong. Hence, the SM representation in this case must be refined by taking static subclasses into account. In this particular example where the static subclasses are elementary, we can have only one dynamic subclass \( (Z_{C_i}) \) of cardinality 1 per static subclass \( C_i \). For example, the SM above must be refined into three different SMs: \( \tilde{m}_1 = RQ(Z^1_{C_1}) + GS(Z^1_{C_2} + Z^1_{C_3}) \), \( \tilde{m}_2 = RQ(Z^1_{C_2}) + GS(Z^1_{C_1} + Z^1_{C_3}) \), \( \tilde{m}_3 = RQ(Z^1_{C_3}) + GS(Z^1_{C_1} + Z^1_{C_2}) \), where \( Z^1_{C_i} \) is the \( j \)th dynamic subclass of \( C_i \). For each of these markings we can now establish which instance of transition \( t_4 \) is enabled without ambiguity.

In conclusion, the presence of static subclasses affects the efficiency of the standard SRG method since asymmetries must always be taken into account, even if the asymmetric behavior of the system is local. Back to the DCS example, the SRG has the same size as the ordinary RG, because static subclasses are elementary and the ‘identity’ is the only allowed permutation between equivalent ordinary markings. The next section shows how to overcome this problem by extending the standard symbolic approach.

In the rest of the paper, we call asymmetric symmetric SM, the symbolic markings developed with/without consideration of the model static subclasses.

### 2.2 The ESRG

Fig. 2 depicts the ESRG of the DCS application (with 3 processors). A node of an ESRG, called Extended SM (ESM), still represents a set of markings, however, its representation is extended in order to take partial symmetries in consideration.

Every ESM (\( \tilde{m} \)) includes a two-level description. It always contains a symbolic marking representation defined without any reference to the static subclasses (first level description): this part is called the Symmetric Representation (SR) of the ESM (depicted in Grey). It optionally contains a set of asymmetric symbolic markings (second level description) which refine the SR by taking into account the static subclasses. The second level description, is expressed in a compact way through the so-called eventualities of the SR.

An eventuality is a set of assignments of static subclasses to each dynamic subclass of the SR. From the SR and the eventualities it is possible to reconstruct the explicit representation of all SMs included in an ESM. Back to our example, the eventualities \( e_1 \), \( e_2 \) and \( e_3 \) of ESM \( \tilde{m}_{10} \) in...
Fig. 2, represent the asymmetric SMs: \( \widehat{m}_1, \widehat{m}_2 \) and \( \widehat{m}_3 \), mentioned in the previous subsection. As a special case, a uniform ESM includes only one SM. This happens when the colors of the class have all the same distribution on places and, in practice, no confusion is introduced by using the SR representation instead of the SM representation to denote it. For instance, the initial ESM \( \widehat{m}_0, \widehat{m}_1, \widehat{m}_3 \) and \( \widehat{m}_13 \) are uniform. The two level representation of an ESM also leads to different types of arcs: the instantiated arcs depart from an eventuality of an ESM (thin arcs) and the generic arcs (thick arcs) depart from the SR part of an ESM. Both types correspond to transition instance firings, but only the instantiated arcs take into account the static subclasses. In the ESRG of Fig. 2, transition instance \((t_5, \langle Z^2 \rangle))\) enabled in \( \widehat{m}_{11} \) is generic. Instead, the instance \((t_4, \langle Z^1_{C_2}, Z^1_{C_1} \rangle))\), enabled in eventuality \(e_3\) of ESM \( \widehat{m}_{10}\), is instantiated since it refers to the refined dynamic subclasses of the eventuality. We denote \( \langle t, \widehat{c} \rangle \) and \( \langle t, \widehat{\widehat{c}} \rangle \) the instances of transition \( t \) whose enabling depends or does not depend on the static subclasses respectively.

Starting from an SWN, the construction of its ESRG looks like a standard reachability graph construction, but it is based on the SM representations and the symbolic firing rule. The key point of the construction process relies on the fact that the representation of eventualities is only required in two cases: (1) if the ESM enables an asymmetric transition and is not uniform; (2) if the ESM is not saturated, i.e. if some of its eventualities are not reachable. The presence of saturated symmetric ESMs in an ESRG leads to space saving since all their eventualities are reachable and as a consequence they could be left implicit (represented only by the SR). The saturated asymmetric ESMs instead require that at least some eventualities (those enabling some asymmetric transition) are explicitly represented. All the ESMs in Fig. 2 are saturated. Among these ESMs, \( \widehat{m}_{10}\) contains three explicit eventualities due to the enabling of the asymmetrical transition \( t_4 \) (it is saturated asymmetric), while the six eventualities of \( \widehat{m}_9 \) are not represented (it is saturated symmetric). Observe that the saturated asymmetric ESM, like \( \widehat{m}_{10} \) in Fig. 2, may have both generic and instantiated arcs departing from them: the saturation property allows to use the SR part of the ESM as a representative for all the eventualities, but it can be used only to fire the symbolic firing instances (generic arcs).

2.3 Performance evaluation of SWNs through the ESRG

The SWN formalism allows to have timed and immediate transitions: the former have an associated delay which is an exponentially distributed random variable, while the latter fire in zero time, and have an associated weight, used to probabilistically solve the possible conflicts.

Whenever all transitions are timed, the SRG is isomorphic to a CTMC from which it is possible to obtain the state probabilities at time \( t \) or, if the chain is ergodic, the steady-state probabilities. This result can be extended to deal with immediate transitions: in this case the embedded Markov Chain approach can be used, and a discrete-time process derived. The state probabilities can then be computed in a similar way as in the continuous-time case.

Lumping of CTMCs is a possible method for dealing with very large chains: it consists in replacing the chain by an equivalent one where each state is an equivalence class of states (called aggregate) of the original one [6]. Different prerequisites for lumping can be formally defined. Here, we refer to the strong lumpability condition: for every aggregate, all the states contained in it, must reach any target aggregate with the same rate.

The SRG built from an SWN is known to be isomorphic to a CTMC that lumps the CTMC associated with the RG [1], in particular the lumped CTMC corresponds to a graph whose nodes are the SMs and whose arcs are weighted with the output rate of the transitions enabled in the SMs.

Now the question is whether it is possible to have a similar result for the ESRG as well. In the DCS model, if all instances of transition \( t_4 \) have the same firing rate, then the lumped CTMC whose nodes are the ESMs fulfills the lumpability condition. If instead transition \( t_4 \) is asymmetric also from a quantitative point of view, i.e. its firing instances have different rates, then some ESMs like \( \widehat{m}_{10} \) do not have the same output rates for all the reachable eventualities. Hence, the represented set of SMs must be split in several sub-aggregates. This could induce some border-effect on the adjacent nodes and cause cascading splitting in sub-aggregates: the algorithm presented in the next section performs this refinement efficiently.

3 Algorithm Description

In this section, we describe an extension of Paige and Tarjan’s partition refinement algorithm [7], for the strong lumpability check of the (SRG) state aggregation induced by the ESRG. With respect to Paige and Tarjan’s algorithm this extension uses a different aggregation condition (the strong lumpability one) and works using the information contained in the ESRG.

The stability condition of Paige and Tarjan’s algorithm is weaker than the strong lumpability one, and is implied by the latter. In fact the strong lumpability condition does not only check that all elements in each aggregate reach the same destination aggregates, but it also checks the transi-
...tion rates from each element in a source aggregate and the destination aggregates.

The algorithm presented in this section works being aware of the aggregations suggested by the ESRG, rather than blindly applying the states aggregation check to the SRG: this allows us to obtain two advantages (that will be explained in details in the following sections), one in the initialization of $X$ and $Q$, which allows us to reduce the number of algorithm steps and the second in the total memory use.

Like the Paige and Tarjan’s Partition refinement algorithm, our extension uses several data structures, namely $Q$, $X$, $C$ described hereafter.

- $Q$ is a double-linked list and represents the current partition of symbolic markings\(^1\); every element of the list is called block. A single block contains a set of elements of type \textit{Node}. A \textit{Node} element can be either an $SR$ or an eventuality. For every \textit{Node} element, the following information is stored:
  - \texttt{type}: it is set to ’instance’ if the \textit{Node} element represents an eventuality of an ESM, to ’macro’ if it represents a saturated symmetric ESM (only the $SR$ part is needed in this case).
  - \texttt{in\_generic}, \texttt{in\_inst}: are the lists of all input generic/instantiated transition firings which reach this \textit{Node}. Every element in these lists contains a pointer to the source \textit{Node} and the transition rate (weight).
  - \texttt{out\_generic}, \texttt{out\_inst}: are the lists of all output generic (instantiated) transition firings enabled in this \textit{Node}. Every element in these lists contains a pointer to the destination \textit{Node} and the transition rate (weight).

- $X$ is a double-linked list and represents another possible partition of the symbolic markings such that $Q$ is a refinement of $X$ and $Q$ satisfies the lumpability condition with respect to every block of $X$. Every block in $X$ is described as a list of one or more $Q$ blocks (in the former case it is called \textit{simple block}, in the latter it is called \textit{compound block}). The \textit{Node} elements contained in each $X$ block can be derived as the union of the \textit{Node} elements of the $Q$ blocks contained in $X$.

- $C$ is a list; it is used to maintain pointers to the compound blocks of $X$ (to retrieve them more efficiently).

The pseudo-code of the algorithm is given in Fig.1 and it can be divided into two phases: the initialization [lines 1-2] and the iterative refinement [lines 3-25].

In the first phase [lines 1-2], the data structures are initialized in this way:

\textbf{[line 1]} Creates the initial $Q$ list using the ESRG: for each saturated symmetric $ESM$, we shall insert a new block into the $Q$ list which will contain only one element of type ’macro’. For each saturated asymmetric $ESM$, we shall insert a new block into the $Q$ list containing as many elements of type ’instance’ as the eventualities of this $ESM$. [line 2] Creates the lists $X$ and $C$ and pre-splits the aggregates of $Q$. The function $\text{Create}_{X,C}(Q)$ will split a $Q$ block iff one or more asymmetric (instantiated) transitions are enabled in it, and the splitting is performed considering only the weights, transitions and destination ESMs of the outgoing asymmetric firings. The list $X$ will contain $n$ blocks.\(^2\) For every block of $Q$, that is not split in this step, a simple $X$ block containing it is inserted. A compound $X$ block is inserted for every split block of $Q$, and it will contain all the new $Q$ blocks generated by its splitting. For every compound $X$ block, one block is inserted in $C$.

After this pre-split phase, we have obtained the two par-

\begin{algorithm}[H]
\caption{Algorithm for the ESRG lumpability check}
\begin{algorithmic}[1]
\STATE 1: $\text{Create}_Q(ESRG)$
\STATE 2: $C=\text{Create}_{X,C}(Q)$
\WHILE{$(C!=\text{NULL})$}
\STATE 3: $S=C.\text{top}()$; \COMMENT{remove a compound aggregate $S$ from $C$}
\STATE 4: $B=S.\text{find\_new \_block}()$; \COMMENT{select a $Q$ block contained in $S$}
\STATE 5: $B=S.\text{compute } E^{-1}(B)$; \COMMENT{$e_B$ = set of elements that reach $B$}
\STATE 6: \IF{$(S'.\text{is\_compound}())$} \STATE 7: $C.\text{push}(S')$; \ENDIF
\STATE 8: \FOR{$(\text{sublist}_k \in \text{list})$} \STATE 9: $n\text{block}=\text{new \_block}$ \STATE 10: \FOR{$(x_j \in \text{sublist}_k)$} \STATE 11: $D_i=x_j.\text{find\_set}()$; \COMMENT{$D_i$ is the block containing $x_j$}
\STATE 12: \ENDFOR
\STATE 13: $D_i.\text{remove}(x_j)$;
\STATE 14: \ENDFOR
\STATE 15: \FOR{$(x_j \in \text{sublist}_k)$} \STATE 16: $\text{block}=\text{new \_block}$.\text{create}$(D_i)$; \COMMENT{create the block corresponding to $D_i$ in new \_block$}
\STATE 17: $\text{block}.\text{insert}(x_j)$;
\STATE 18: \ENDFOR
\STATE 19: \WHILE{$(\text{nblock}=\text{new \_block}.\text{top}()!=$\text{NULL})$}
\STATE 20: $\text{block}=Q.\text{insert}(\text{nblock})$; \COMMENT{return a pointer to the inserted block}
\STATE 21: $X\text{block}=\text{block}.\text{find\_block}(X)$;
\STATE 22: $X\text{block}.\text{update}(\text{block})$;
\STATE 23: \IF{$(X\text{block}.\text{is\_compound}())$} \STATE 24: $C.\text{push}(X\text{block})$;
\STATE 25: \ENDIF
\STATE 26: \RETURN{$Q$};
\end{algorithmic}
\end{algorithm}

\(^1\)It will be clarified later how the initial partition is chosen and how the iterated refinement steps leading to each successive refinement work.

\(^2\)where $n$ is the number of $Q$ blocks before the initial pre-split.
tions $Q, X$, so that the $Q$ satisfies the lumpability condition with respect to every block of $X$.

The second phase [lines 3-25] is the algorithm core and consists of repeating the refinement step until $C = \emptyset$. The refinement step is performed as follows:

**[lines 4-5]** Remove a block $S$ from $C$ and select a $Q$ block $B$ contained in this $X$ block.

**[lines 6-7]** Divide the block $S$ in $S' = B$ and $S'' = S - B$; if $S''$ is compound, put $S''$ into $C$.

**[lines 8-9]** For every Node element in $B$, put in the list $e_B$ the Node elements which reach it. These Node elements are found scanning for every Node element in $B$ its lists $in_{\text{inst}}$ and $in_{\text{generic}}$. They are inserted in $e_B$, if they have not yet been added, otherwise only the corresponding rate is updated. In particular all macroc elements found, before inserting in $e_B$, will have to be instantiated, so that the macroc elements will be substituted by all the eventualities they represent. The function $\text{prepare split}(\cdot)$ builds the list of lists $l_{\text{list}}$ where every sublist $\text{sublist}_k$ contains all the elements that reach $B$ with the same rate.

**[lines 10-25]** Split the $Q$ blocks according to the $l_{\text{list}}$. A generic $Q$ block is split if it contains one or more Node elements stored into $l_{\text{list}}$. All these Node elements are deleted from this block and are separated into one or more new $Q$ blocks according to their rates. A separate block for each outgoing rate category is created. Then the $X$ block containing the split $Q$ block is updated, this $Q$ block must be replaced with the new $Q$ blocks. If the $X$ block was simple then it becomes compound. For every new compound block in $X$, a new block is inserted in $C$.

**A simple example:** in this paragraph we describe two steps of our algorithm applied on the ESRG of Fig.4 obtained by the SWN model in Fig.3 where all the transitions have rate 1. In Fig. 5, it is shown that in this case, the refined ESRG is equal to the SRG of model.

The initial $Q$ list elements are:

$$Q = \{ q_1(\hat{m}_0), q_2(\hat{m}_1), q_3(\hat{m}_2), q_4(e_1, e_2) \}$$

where $\hat{m}_0, \hat{m}_1, \hat{m}_2$ are macroc, and $e_1, e_2$ are the eventualities corresponding to the saturated asymmetric $\hat{m}_3$.

In the pre-split phase only the block $q_4$ is split. After this phase the elements of the $Q$ list are:

$$Q = \{ q_1(\hat{m}_0), q_2(\hat{m}_1), q_3(\hat{m}_2), q_4(e_1 = \hat{m}_6), q_5(\hat{m}_7) \}$$

and those of the $X$ list are: $X = \{ x_1(q_1), x_2(q_2), x_3(q_3), x_4(q_4, q_5) \}$. The $C$ list contains only one element: $c_1(\{x_3\})$.

The first step refinement selects $x_4$ as $S$. After that, $q_4$ becomes $B$. During the $l_{\text{list}}$ computation the $\hat{m}_2$ is instantiated in $\hat{m}_3, \hat{m}_4, \hat{m}_5$. The $l_{\text{list}}$ contains only one element: $c_1(\{x_4\})$.

3In this case every sublist contains only one element.

3.1 Lumped Markov chain generation

In this section, it is explained how to generate the lumped CTMC from the refined ESRG. We can distinguish two different cases:

**[Rule 1 :]** $Ev_i \xrightarrow{\nu} Ev_j$ is the case corresponding to the instantiated firings departing from an eventuality: the rate is defined as follows:

$$\mu(t, \hat{c}) = |\hat{m}_4 \xrightarrow{(t, \hat{c})} | w(t)(\hat{c})$$

Figure 3. A simple SWN

Figure 4. ESRG of model in Fig.3
where $\tilde{m}_i$ is the SM corresponding to $Ev_i$, $\tilde{m}_i \xrightarrow{(t, \hat{c})} \tilde{m}_j$ is the firing corresponding to the instantiated arc connecting $Ev_i$ to $Ev_j$, finally $w[t](\hat{c})$ is the rate associated with $t$ (which may depend on the static subclasses of the color instance).

**Rule 2:** $SR_i \xrightarrow{G} SR_j$ is the case corresponding to a generic symmetric firing from an ESM to another ESM: the rate is defined as follows:

$$\mu(t, \hat{c}) = |\tilde{m}_i \xrightarrow{(t, \hat{c})} \tilde{m}_j| \cdot w[t]$$

where $\tilde{m}_i$ is the ESM corresponding to $SR_i$, $\tilde{m}_i \xrightarrow{(t, \hat{c})} \tilde{m}_j$ is the firing corresponding to the generic arc connecting $SR_i$ to $SR_j$, finally $w[t]$ is the rate associated with $t$.

Observe that if the arc is generic, then the rate of transition $t$ does not depend on static subclasses.

These two rules cover all the possible types of arcs connecting the Node elements. We can create a new ESRG (= final $Q$) from the ESM by the CTMC computing its rates using these rules. The pseudo-code of the algorithm is given in Fig.2. This algorithm represents the CTMC as a list, where every element is a 3-tuple $<Q block_i, Q block_j, rate>$.

4 Implementation and Results

In this section we are going to show and to compare the performance of this algorithm over a few examples with respect to the algorithms that may work at SRG level that can be found in the literature (e.g., like those in [5, 8, 9, 10]). A direct comparison with the performance of the previous ESRG-based algorithm proposed in [4] is not possible, because no implementation of such algorithm is available.

The results obtained with the new algorithm implementation will be compared with those that would be obtained by directly applying the Paige and Tarjan’s algorithm on the SRG without any knowledge of the ESRG: the comparison will be made in terms of number of steps (which gives a measure of the time saving) and of the number of not instantiated ESMs (which gives a measure of the space saving): this is a fair comparison, even if in our algorithm the single step may require the instantiation of the SMs not explicitly represented in an ESM to be checked for lumpability. The reasons why the comparison is fair are the following: 1) if the ESM instantiation does not need to be performed, we are splitting a block containing only eventualities, then the split step in our algorithm is equivalent to that of the other algorithms; 2) in the other case our algorithm calls a function that instantiates the SMs not yet explicitly represented in the ESM: since this function is called only one time for every ESM and its (worst case) cost is constant, we can consider again the split steps comparable, in terms of order of magnitude complexity. Finally, the cost of the instantiation function, would anyway be paid in the SRG generation phase in case the ESRG was not used: the examples presented in...
this section show that it may well be the case that the SRG cannot be generated because it is too large, while the corresponding ESRG fits the available memory, and does not need so much refinement to satisfy lumpability.

The algorithm used for comparison purposes, is a modified version of the same algorithm presented in this section, where $X$ initially contains only one block comprising all blocks of $Q$ and where all the eventualities are instantiated. This modified algorithm in fact mimics the behavior of other algorithms based on Paige and Tarjan’s one, and differs from those algorithms only because it starts with an initial refinement that prevents to reach an aggregation that is coarser than that induced by the ESRG, so that the number of refinement steps of the modified algorithm is a lower bound with respect to these other algorithms ([5, 8, 9, 10]). Although in some (rare) cases it may happen that a coarser partition than that induced by the ESRG could be reached by the other algorithms, this is not desirable, because it may prevent the computation of relevant performance indices.

Figure 6. The second simple SWN

The models analyzed in this section are: two simple SWNs (Fig.3,6), the model of a DCS algorithm (Fig.2) and a net that is representative of a class of partially symmetric systems (Fig.7) which we call benchmark net.

In the first asymmetric model depicted in Fig.3, and introduced in [11], the strong lumpability condition induces no splitting in the ESRG (ESRG = refined ESRG).

Our algorithm, applied on this net, tests the strong lumpability condition in a single step (only the initial split in the function Create $X,C()$ is computed) preserving all the macroc elements (ESM). The other version uses eight steps. We obtain a double advantage comparing the two implementations: a smaller number of steps and a smaller memory requirement (no ESM is instantiated).

The application of the algorithm to the second asymmetric model [11], generates a splitting propagation on all ESMs. This is the worst case because all the macroc elements have to be instantiated and split, so that the advantage in this case is in the smaller number of steps. The results of these two SWN nets are summarized in Table 1, where in the first column is reported the name of nets, the second contains the number of states of the SRG and ESRG, and the third contains the number of states of the refined ESRG; the fourth and the fifth columns contain the number of algorithm steps with $X=\{\text{one block with all SRG states}\}$ or with $X=\text{ESRG}$, initially. The sixth column contains the number of non split ESMs (macroc) in the refined ESRG, and also includes the number of corresponding (never instantiated) eventualities.

The third and fourth model represent examples of intermediate cases, in fact running the algorithm on these models, we can observe a propagation of the splitting, which divides some ESMs into sub-aggregates grouping lumpable eventualities. Therefore the refined ESRG is composed by some ESMs and some sub-aggregates.

The results obtained by running the algorithm on the DCS example are summarized in Table 2: the first column shows the number of cardinality one static subclasses.

In this case the memory gain (measured in non split ESMs) is not very relevant, because the ESM number in the refined ESRG and the number of eventualities represented by them is not high, as shown in Table 2. This gain becomes more relevant in all those systems where the asymmetric behavior and the symmetric behavior can be separated clearly in two phases, that may correspond to two sub-models. The passage from one sub-model to the other happens after a synchronization step of all the system (color) objects, so that the splitting of the asymmetric ESMs do not propagate to the ESMs belonging to the symmetric behav-

---

4In other words there are no macroc elements, hence the required memory space is greater than that of our algorithm.
Table 1. Results for the two simple SWNs

<table>
<thead>
<tr>
<th>#static subcl.</th>
<th>#SRG-#ESRG</th>
<th>#Ref. ESRG</th>
<th>#Steps X=ESRG</th>
<th>#Steps X=SRG</th>
<th>#Non split ESM</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>45-14</td>
<td>21</td>
<td>8</td>
<td>16</td>
<td>2(9)</td>
</tr>
<tr>
<td>4</td>
<td>161-21</td>
<td>41</td>
<td>29</td>
<td>33</td>
<td>3(28)</td>
</tr>
<tr>
<td>5</td>
<td>573-32</td>
<td>68</td>
<td>40</td>
<td>59</td>
<td>4(75)</td>
</tr>
<tr>
<td>6</td>
<td>2001-44</td>
<td>102</td>
<td>61</td>
<td>87</td>
<td>5(128)</td>
</tr>
<tr>
<td>7</td>
<td>6849-58</td>
<td>143</td>
<td>95</td>
<td>127</td>
<td>6(400)</td>
</tr>
</tbody>
</table>

Table 2. Results for the DCS model

<table>
<thead>
<tr>
<th>#elem subcl.</th>
<th>#SRG-#ESRG</th>
<th>#Ref. ESRG</th>
<th>#Steps X=ESRG</th>
<th>#Steps X=SRG</th>
<th>#Non split ESM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,2,2</td>
<td>18423-504</td>
<td>543</td>
<td>40</td>
<td>533</td>
<td>494</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(18374)</td>
</tr>
<tr>
<td>3,3,3</td>
<td>171600-1377</td>
<td>1465</td>
<td>80</td>
<td>1452</td>
<td>1364</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(171499)</td>
</tr>
<tr>
<td>5,5,5</td>
<td>5556894-6279</td>
<td>4862</td>
<td>227</td>
<td>6260</td>
<td>5984</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(5556599)</td>
</tr>
<tr>
<td>3,3,3</td>
<td>22864948-10647</td>
<td>11074</td>
<td>428</td>
<td>11052</td>
<td>10625</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(22868999)</td>
</tr>
<tr>
<td>3,3,3</td>
<td>34304004-12672</td>
<td>13162</td>
<td>491</td>
<td>13111</td>
<td>12649</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(34303491)</td>
</tr>
</tbody>
</table>

Table 3. Results for the benchmark net

5 In order to give an example we have chosen the fourth model, a benchmark net whose behavior falls in this class. The asymmetric part is due to the three transitions with guards between places P4 and P3: the only non symmetric markings are those enabling these transitions. Since there is a synchronization after the firing of these transitions that can be followed by a repetition of the asymmetric behavior or a transition to the symmetric one, the splitting propagation is limited. The results are shown in Table 3: the first column shows the number of elements in each static subclass. It is important to observe that in the last three cases it is not possible to directly generate the SRG; its number of states is obtained indirectly from the ESRG by computing the size of eventualities represented by each ESM.

5 Examples of systems behaving in this way can be found in the distributed algorithms field, where several entities behave similarly and only occasionally one of them starts a global protocol, e.g. for synchronization or recovery purposes, in which it plays a role of master, to then come back to the normal behavior. For example, the Time Warp distributed simulation algorithm is a member of this class, occasionally one process starts a global clock synchronization protocol involving all the processes.

5 Discussion

The efficiency of the approach presented in this paper relies on two main factors: the fact that it is based on Paige and Tarjan’s algorithm and that it exploits the ESRG as a starting point of the performance study. Let us discuss the advantages of the approach with respect to the other approaches that can be found in the literature.

The use of Paige and Tarjan’s algorithm has allowed us to build an algorithm with complexity $O(m \log(n))$, (where $n$ is the number of SRG nodes, and $m$ is the number of transitions) which is better than the $O(n^m)$ complexity of the algorithm proposed in [4], also based on the ESRG. In practice however the number of actually tested arcs and nodes can be much smaller than $m$ and $n$, this happens when several saturated symmetric ESMs are not instantiated by the algorithm. Of course in the worst case all ESMs are instantiated, but we have shown through some examples that there are classes of models where the savings in terms of the reduction in nodes and arcs to be tested can be very relevant. This makes our approach more efficient than other approaches (e.g. some of the lumping algorithms that have been devised in the context of Stochastic Process Algebras) which are also based on the Paige and Tarjan’s one.

Another paper which is related with the present one, is [12], however a direct comparison in this case is not possible since our algorithm is based on strong lumpability while the other one is based on exact lumpability. In general it is not possible to decide when strong lumpability allows to aggregate more than exact lumpability or vice-versa (see [11]): it depends on the specific model. However, the algorithm presented in this paper can be rather easily extended to check exact lumpability instead of strong lumpability: this can be obtained by checking the rates of arcs entering a particular SM in an ESM from a given aggregate rather than checking the arcs going from a particular SM in an ESM towards another aggregate (arcs in\_generic, in\_inst are considered instead of out\_generic, out\_inst). We have a preliminary implementation of the exact lumpability check that has been compared on a few examples with the algorithm in [12]: there are some relevant differences between the two approaches, since the latter is based on the computation of the so-called Dynamic SRG (DSRG) which satisfies the exact lumpability condition by construction. In the DSRG based approach it may happen that some markings (corresponding to the SMs) are replicated in the final lumpable structure, hence it may contain more aggregates than the refined ESRG, however it could also be the case that even if the number of aggregates in the refined, exactly lumpable ESRG is smaller than that obtained in the DSRG, the number of instantiated eventualities along the lumpability check process exceeds the final number of aggregates, potentially requiring more
memory. It would be interesting to find a characterization of those classes of models (pretty much in the line of the benchmark net) that can gain more from one type of lumpability criteria or from a specific aggregation algorithm.

An advantage of the ESRG-based approach with respect to the DSRG-based one is that in case transition rates are modified in the model, it is sufficient to run the lumpability check reusing the previously constructed ESRG, while in the DSRG based approach any variation in the model requires to start the construction of the DSRG all over again.

The ESRG and the lumpability check algorithm implementation have been integrated in GreatSPN. The implementation is structured in layers: it benefits from the symbolic manager kernel which provides the symbolic markings representation and the symbolic firing. The next layer called DYSY is common to every tool which needs to manage static-subclasses on-the-fly. It provides an interface to higher level tools like the one proposed here for performance evaluation purposes (it offers the access to the eventuality manager), or the one used for a symbolic LTL model checker in [13].

6 Conclusions and Future Work

In this paper we have presented a new algorithm for performing the strong lumpability check on the ESRG of an SWN model and generate the (coarsest) corresponding lumped Markov Chain. It is a variation on the theme of Paige and Tarjan’s partition refinement algorithm, but it also exploits the indications on how to aggregate, that can be derived from the ESRG: this allows to perform less refinement steps than other algorithms not driven by the ESRG. The algorithm has been implemented and the first encouraging results on a few representative models have been shown.

The future work on this line consists in extending the algorithm to work not only with strong lumpability but also with exact lumpability. Another topic for future work is to integrate this technique with the use of special data structures, DDD (Data Decision Diagram) that allow to exploit symmetries in the marking structure to dramatically reduce the amount of memory needed to keep the ESRG while executing the lumpability check algorithm.

References


