From partially to fully lumped Markov chains in Stochastic Well Formed Petri Nets

S. Baarir
LIP6
UPMC Paris
Souheib.Baarir@lip6.fr

M. Beccuti
Dip. di Informatica
Univ. di Torino
beccuti@di.unito.it

C. Dutheillet
LIP6
UPMC Paris
claude.dutheillet@lip6.fr

G. Franceschinis
Dip. di Informatica
Univ. Piemonte Orientale
giuliana@mfn.unipmn.it

ABSTRACT

This paper presents a generic framework for building quotient graphs for Stochastic Well-formed Net models by exploiting the symmetries implicitly defined in the model. Two instantiations are presented, one based on static symmetries and the other one based on dynamic symmetries. The second method can usually deal with partially symmetric systems in a more effective way than the first one. However, in some cases it may result in a larger graph. We present here a new approach that overcomes the weakness of these methods. All these techniques can be used for efficient performance analysis of systems.

1. INTRODUCTION

ICT systems have become pervasive and their complexity is growing, also because most systems tend to be interconnected either locally with their environment, or globally through telecommunication networks. Performance, reliability, security, energy efficiency, are examples of issues that need to be faced since the early stages of their design.

Model-based analysis and simulation can support the verification and evaluation of the system under design. Often the analysis methods are based on the generation of the state space: these methods must cope with the combinatorial explosion of the number of states. Several approaches have been proposed in the literature to manage this problem: decomposition, approximations, bounding, and the use of very efficient data structure (namely Decision Diagrams). In this paper we concentrate on symmetry-based methods, exploiting the presence of similarly behaving components to aggregate the states and state transitions into equivalence classes, hence generating a more abstract and compact state space (a quotient graph).

This idea has been applied in the literature to both verification [6, 8, 9] and performance evaluation (through the notion of lumpability [5, 13, 14]). Such methods have also been proposed in the specific context of the high-level Petri net formalism called Stochastic Well-formed Nets (SWN) [3, 4, 5] for both qualitative (i.e. model checking) and quantitative (i.e. performability) analysis: exploiting the structured syntax of the formalism, a quotient graph is directly derived from the model and used for verification purposes, moreover a Continuous Time Markov Chain (CTMC) can be derived from the quotient graph: either the quotient graph already satisfies a markovian lumpability condition, or a refinement algorithm is applied. The resulting CTMC can be solved to compute performance/reliability measures. These methods have been implemented in the GreatSPN tool [1, 15].

In ICT systems, symmetries can be global, i.e., symmetric components *always* behave in a similar way, or local, meaning that they *almost always* behave in a similar way but at some point, they perform different actions. Among existing quotient graphs, the Symbolic Reachability Graph (SRG) and the Dynamic SRG (DSRG) satisfy the exact lumpability condition by construction. The SRG has proved very efficient for representing globally symmetric systems but the reduction is poor when considering local symmetries. The DSRG attempts to efficiently deal with such systems. Indeed the representation of "state aggregates" in the DSRG is more flexible than in the SRG, allowing to capture symmetries that change during the model evolution. Many experiments show that the DSRG can improve significantly the size of the representation with respect to the RG. However, in some cases, its size is greater than that of the RG.

In this paper, a unifying framework for building quotient graphs for SWNs is presented. We show how it can be applied to the SRG and the DSRG constructions. We use this framework for proposing an alternative construction, the Partially Lumped DSRG (*PLDSRG*), that makes it possible to efficiently represent locally symmetric systems by partially relaxing the lumpability constraint on the DSRG construction. A lumped Markov chain can still be derived from the *PLDSRG* by refinement.

The paper is organized as follows: Sec. 2 introduces the SWN formalism and the notions of symbolic marking and firing, Sec. 3 presents a generic algorithm that can produce a quotient graph for SWNs, and how it is instantiated to produce the SRG or the DSRG. In Sec. 4, a new method is proposed with the aim of overcoming the weak points

VALUETOOLS 2009, October 20-22, Pisa, Italy

DOI 10.4108/ICST.VALUETOOLS2009.7733

of the SRG and DSRG. Two examples are then presented in Sec. 5, showing the effectiveness of the proposed new method in practice, also comparing it with the Extended SRG method. Conclusions and future work directions are presented in Sec. 6.

2. PRELIMINARIES

2.1 Lumpability in Markov chains

Lumping of (finite) MCs is a useful method for dealing with large chains [10]. 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. We focus here on *exact* lumpability. More details can be found in [3].

DEFINITION 1 (CONTINUOUS TIME MARKOV CHAIN). *A CTMC $\mathcal{C} = \langle S, Q, \pi_0 \rangle$ is defined by a state space S, an infinitesimal generator Q, that is an $S \times S$ 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.*

Exact lumpability is defined by:

DEFINITION 2. *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. $\{S_i\}_{i \in I}$ 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 proposition holds:

PROPOSITION 3. *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:*
- $\forall i, j \in I, \forall s \in S_j,\ Q^{lp}(i,j) = (|S_j|/|S_i|) \times (\sum_{s' \in S_i} Q(s', s))$
- *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 $\forall i \in I, \forall s, s' \in S_i, \pi(s) = \pi(s')$.*

As states within an aggregate are equiprobable, exact lumpability makes it possible to retrieve original state probabilities, provided that the cardinality of aggregates is known. As a consequence any performance index that can be computed on the MC can also be obtained from the exactly lumped one.

2.2 The Well-formed Petri Net model

In this section we formally define the SWN syntax and the notions required for building a quotient reachability graph.

DEFINITION 4 (STOCHASTIC WELL-FORMED NETS). *An SWN is a nine-tuple:*

$$\mathcal{N} = \langle P, T, \mathcal{C}, Cd, \mathbf{Pre}, \mathbf{Post}, \mathbf{Inh}, \mathbf{pri}, \mathbf{w} \rangle$$

- *P and T are the finite, disjoint, non empty sets of places and transitions, modeling the state and the possible state changes respectively.*
- *$\mathcal{C} = \{C_1, \ldots, C_n\}$ is a family of finite and disjoint* basic color classes. *By convention, classes with index up to h are not ordered, while classes with higher index are (circularly) ordered (a successor function is defined on their elements); basic color class C_i may be partitioned into* static subclasses $\{C_{ij}\}_j$. *The global partition of $\mathcal{C}$ is $\mathcal{P}art_a = \{\{C_{ij}\}_j\}_i$.*
- *Cd defines the* ***color domain*** *of each place and transition; place color domains are specified as Cartesian products of color classes (with repetitions), transition color domains define their* parameters *and the corresponding types (in $\mathcal{C}$); each parameter is associated with a* variable *appearing in expressions labeling some arc connected to the transition; $Var(t)$ ($Var_i(t)$) denotes the variables of t (of type C_i). The* instances *of a transition (i.e., the possible values assigned to its parameters) can be restricted by means of a* guard *pred, specified through a* standard predicate, *that is a Boolean expression of* basic predicates. *Basic predicates are: $x = y$, $x = {!y}$, $d(x) = C_{ij}$, $d(x) = d(y)$, where $x, y \in Var_i(t)$ have the same type, $!y$ is the successor of y (if y belongs to an ordered class), and $d(x)$ denotes the static subclass x belongs to.*
- $\mathbf{Pre}[p,t], \mathbf{Post}[p,t] : Cd(t) \rightarrow \mathrm{Bag}(Cd(p))$ *are the* pre- *and* post- incidence matrices, *whose elements are* arc expressions.
- $\mathbf{Inh}[p,t] : Cd(t) \rightarrow \mathrm{Bag}(Cd(p))$ *is the matrix defining the inhibitor arcs and associated arc expressions.*
- ***Arc expressions*** *are weighted (and possibly guarded) sums of tuples; the tuple elements in turn are weighted sums of* basic functions*:*

$$f_i = \sum_{q=1}^{ns_i} \alpha_{i,q}.S_{Ci_q} + \sum_{x \in Var_i(t)} (\beta_x.x + \gamma_x.!x)$$

  where $\alpha_{i,q}$, β_x and γ_x are integers. The multiset returned by a tuple of basic functions is the Cartesian product of the multisets returned by its elements.
- *The domain of basic functions is a class $C_i \in \mathcal{C}$, their codomain is the set of multisets over C_i ($Bag(C_i)$). There are three types of basic functions: the* projection, *denoted x, where $x \in Var(t)$, and selecting the value of x from transition instance (t,c); the* successor, *denoted $!x$, returning the successor of x; and the* diffusion / synchronization, *denoted S_{C_i} (or $S_{C_{ij}}$), which is constant and returns the whole set of colors of C_i (of $C_{ij} \subseteq C_i$).*
- *Function* $\mathbf{pri} : T \rightarrow \mathbb{N}$ *is a function associating a priority with each transition: priority zero is reserved to timed transition (with random firing time, exponentially distributed), the other values to immediate transitions (with zero firing time).* $\mathbf{w}$ *is a T indexed vector of functions that assigns rates to timed transitions and weights to immediate transitions (used for the probabilistic characterization of conflicts resolution): it can be both color and marking dependent, however, we consider here the simpler case where* $\mathbf{w}$ *is color and marking independent.*

It is worth noting that arc expressions of SWN prevent objects belonging to the same static subclass from having different behaviors: whatever action an object can perform,

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

any other one will be able to perform the same at some point of its evolution.

Let us illustrate the above definition on the example SWN of Fig. 1: it comprises two color classes, each partitioned into two static subclasses. The elements of a basic color class represent (identities of) objects of the same nature; a color class may be partitioned into *static subclasses*: elements in different static subclasses represent objects of the same nature but with different behavior.

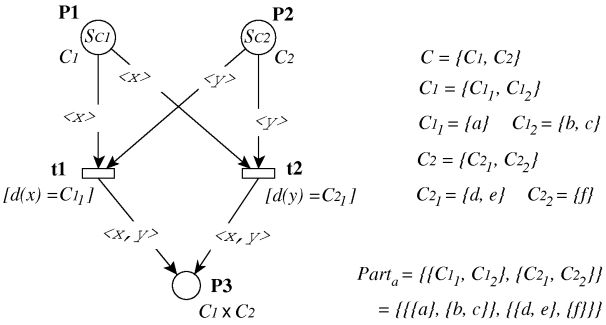

Figure 1: A simple example of SWN model

The color domain of a place defines the multi-field data structure associated with its tokens. In the SWN of Fig.1 $Cd(P3) = C_1 \times C_2$, i.e., its tokens contain two fields of type C_1 and C_2 respectively.

The transition color domains define their ($\mathcal{C}$-typed) parameters and a guard: in the SWN of Fig.1 transition $t2$ has two parameters, x and y, of type C_1 and C_2 respectively; the guard $d(y) = C2_1$ requires y to take values in $C2_1$. All arc expressions in the example are simply tuples of projections: for instance transition $t2$ has an input arc with expression $\langle y \rangle$ and an output arc with expression $\langle x, y \rangle$: when an instance of $t2$ fires, a token of the color associated with y is withdrawn from $P2$ while a token with the first field equal to the value of x and the second field equal to the value of y is put into $P3$.

The priority of transitions is specified through a natural number: in the example all transitions have priority 0, i.e. they are all timed (immediate transitions are depicted as black bars and labeled with a priority number if it is greater than 1). $\mathbf{w}$ defines a firing rate for each transition (in this example it is not color nor marking dependent).

An *ordinary marking* m is a function mapping each place p into a multiset on $Cd(p)$ (denoted as a weighted sum of colors, represented by tuples). m_0 is the *initial marking*.

We call *instance* of t, denoted (t, c) a binding c of the variables of $Var(t)$ to elements in the appropriate color class.

Given a marking m, an instance (t, c) *has concession* in m iff: (1) $pred(t)(c)$ holds true; (2) $\forall p$: $\mathbf{Pre}[p, t](c) \leq m(p)$; (3) $\forall p, \forall c' \in \mathcal{C}(p) : \mathbf{Inh}[p, t](c)(c') = 0$ or $\mathbf{Inh}[p, t](c)(c') > m(p)(c')$, An instance is *enabled* in m *iff* no instance of a higher priority transition has concession in m. An enabled instance (t, c) may fire, producing a new marking m' ($m[t, c\rangle m'$): $\forall p, m'(p) = m(p) - \mathbf{Pre}[p, t](c) + \mathbf{Post}[p, t](c)$. A *path* is a sequence $m_0[t_1, c_1\rangle m_1[t_2, c_2\rangle m_2 \ldots m_{n-1}[t_n, c_n\rangle m_n$. The set of all markings reachable from m is denoted by $[m\rangle$ ($[m_0\rangle$ is called *Reachability Set*, RS). The *Reachability Graph (RG)* structure includes all possible paths from initial marking m_0 and models the behaviour of the system.

2.3 Symbolic Marking and Symbolic Firing

The interest in SWN is due to the *Symbolic Marking* and *Symbolic Firing* notions that allow an (automatic) construction of reduced representations of the RG.

Symbolic marking: a symbolic marking is a compact representation of a set of ordinary markings, which are equivalent up to a permutation/rotation of colors within static subclasses of non ordered/ordered basic classes; a symbolic marking can also be interpreted as a more abstract representation of the system state. It can be formally defined through the group of permutations/rotations operating on $C_i \in \mathcal{C}$ (the group of permutations operating on a set E is the group of all the bijections of E on itself; rotations are the permutations preserving the successor relation). The need to partition the basic classes into finer subsets (i.e. the static subclasses) requires to consider the subgroup of permutations that exchange only elements within the same static subclass (with each static subclass corresponding to the so called "orbit" of some color element with respect to the subgroup). The groups of permutations operating on basic classes can be combined to obtain a group operating on the markings.

An equivalence relation can thus be established between markings: the equivalence classes induced by such relation are the so called *symbolic markings*. Let us define a *a symbolic representation* for each equivalence class: it relies on a *set-based* notation where objects (colors) used for the description of symbolic markings are replaced by *set-variables*. The set of possible instances of those variables gives *exactly* the set of ordinary markings in the symbolic marking.

Notice that the average number of ordinary markings into a symbolic marking depends on the partition into static subclasses, namely $\mathcal{P}art_a$: the finer the partition, the more restrictive the set of permutations, the less efficient the grouping of ordinary states; the maximum grouping is achieved when $\mathcal{P}art_a = \mathcal{P}art_s$, where $\mathcal{P}art_s$ is the degenerate partition with only one static subclass for each class. For the net of Fig.1, $\mathcal{P}art_s = \{\{\{a, b, c\}\}, \{\{d, e, f\}\}\}$.

Definition 5 (Symbolic partition of $\mathcal{C}$ and Cd). *Let $\mathcal{P}art = \{\mathcal{P}art^i\}_{i=1,\ldots,|\mathcal{C}|}$ be a partition of $\mathcal{C}$ into subclasses. We define a symbolic partition of $\mathcal{C}$ into dynamic subclasses as $\mathcal{S}ymb = \{\mathcal{S}ymb_i\}_{i=1,\ldots,n}$ where $\mathcal{S}ymb_i = \langle \mathcal{D}yn_i, card_i, d_i \rangle$ is the symbolic partition of class C_i, and:*

- *$\mathcal{D}yn_i = \{Z_i^k\}_{k=1,\ldots,k_i}$ is a representation of a set of partitions of C_i. Each element Z_i^k is called dynamic subclass.*
- *$card_i : \{1, \ldots, k_i\} \to \mathbb{N}$, is a function that associates with each Z_i^k its cardinality (also denoted $|Z_i^k|$), i.e., the number of objects (basic colors) that it represents.*
- *$d_i : \{1, \ldots, k_i\} \to \{1, \ldots, |\mathcal{P}art^i|\}$, such that:*
  1. *$\mathcal{P}art^{i,d_i(k)}$ is the partition element in which the colors represented by Z_i^k are instantiated,*
  2. *$\sum_{k|d_i(k)=j} card_i(k) = |\mathcal{P}art^{i,j}|$.*

The extension to color domain $D = \prod_{i=1}^{n} C_i^{e_i}$ is straightforward:

$$\mathcal{S}ymb_D = \prod_{i=1}^{n} \prod_{j=1}^{e_i} (\mathcal{D}yn_i) \stackrel{def}{=} \prod_{i=1}^{n} (\mathcal{D}yn_i)^{e_i}$$

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

Consider again the model in Fig. 1. A possible symbolic partition associated with $Part_a$ is given by:

$\mathcal{D}yn_1 = \{Z_1^1, Z_1^2, Z_1^3\}$ $\quad$ $\mathcal{D}yn_2 = \{Z_2^1, Z_2^2\}$
$card_1(1) = card_1(2) = card_1(3) = 1$
$card_2(1) = 1$ $\quad$ $card_2(2) = 2$
$d_1(1) = 2 \quad d_1(2) = 1 \quad d_1(3) = 2 \quad d_2(1) = 2 \quad d_2(2) = 1$

This means for instance that dynamic subclass Z_1^3 represents any object chosen in C_{12} ($d_1(3) = 2$). Hence, the above symbolic partition represents two actual partitions of $\mathcal{C}$:
$Z_1^1 \to \{b\}, Z_1^2 \to \{a\}, Z_1^3 \to \{c\}, Z_2^1 \to \{f\}, Z_2^2 \to \{d, e\}$
$Z_1^1 \to \{c\}, Z_1^2 \to \{a\}, Z_1^3 \to \{b\}, Z_2^1 \to \{f\}, Z_2^2 \to \{d, e\}$.

Now, we can formally define a symbolic marking.

Definition 6 (Symbolic Marking). *A symbolic marking is a tuple $s = \langle \widehat{m}, \mathcal{P}art \rangle$ where $\widehat{m} = \langle mark, \mathcal{S}ymb \rangle$ and $\widehat{m}.\mathcal{S}ymb$ is a symbolic partition of $\mathcal{C}$ w.r.t. $\mathcal{P}art$ while $\widehat{m}.mark$ is a function that associates with each place $p \in P$ a multiset on $\mathcal{S}ymb_{Cd(p)}$: $\widehat{m}.mark(p) \in Bag(\mathcal{S}ymb_{Cd(p)})$. The equivalence class represented by s is denoted $[s]$.*

As there are several ways to symbolically represent the same set of ordinary markings, a canonical representation is defined so that the indexes assigned to each dynamic subclass satisfy a given (arbitrary) ordering condition that ensures the unicity of representation [5].

The syntax of SWN ensures that for a symbolic marking where $\mathcal{P}art = \mathcal{P}art_a$, the ordinary markings within this SM reach sets of successors that are equivalent, up to a permutation of the elements in their instances. To compute the set of successors at the symbolic level, we define a symbolic firing rule which *is valid only when* $\mathcal{P}art = \mathcal{P}art_a$.

Symbolic firing: our symbolic firing rule is an adaptation of the ordinary firing rule to symbolic markings. The *symbolic instance* of a transition requires the introduction of two additional functions, λ and μ: $\lambda(x)$ identifies the dynamic subclass assigned to variable x. If two variables x and x' in $Var_i(t)$ are assigned to the same dynamic subclass, function μ is used to determine whether the chosen objects are the same or not. If C_i is ordered, μ gives the position of the selected object in the dynamic subclass that has been chosen for the instance. For this definition let's assume an arbitrary ordering (e.g. a lexicographical one) among the variables of t in $Var_i(t)$.

Definition 7 (Symbolic instance). *Let t be a transition such that $Cd(t) = \prod_{i=1}^{n} C_i^{e_i}$. Let $\langle \widehat{m}, \mathcal{P}art_a \rangle$ be a symbolic marking. Let $\lambda = \{\lambda_i : Var_i(t) \to \{1, \ldots, |\mathcal{D}yn_i|\}$, $\mu = \{\mu_i : Var_i(t) \to \mathbb{N}\}$, $inst = (t, (\lambda, \mu))$ is a symbolic instance of t referred to $\widehat{m}$ iff $\forall i \in \{1, \ldots, n\}, \forall x \in Var_i(t)$*
• $\mu_i(x) \leq card_i(\lambda_i(x))$,
• If $i < h$ then $\forall 0 < l < \mu_i(x), \exists x' < x$ such that $\lambda_i(x') = \lambda_i(x) \wedge \mu_i(x') = l$.
If $Var_i(t) = \emptyset$ then μ_i et λ_i are not defined.
The equivalence class represented by inst is denoted $[inst]$.

The symbolic firing of symbolic instance $(t, (\lambda, \mu))$ from a symbolic marking $\langle \widehat{m}, \mathcal{P}art_a \rangle$ is composed of three steps: splitting, firing and canonization of the reached symbolic marking. We briefly describe these three steps, the detail of which can be found in [5]:

1. In order to test the enabling of a symbolic instance in a symbolic marking, we need to isolate the symbolic objects selected by functions λ and μ: a dynamic subclass of cardinality 1 is created for every object involved in the firing (observe that this operation is a refinement of the symbolic partition). The resulting representation is called *split symbolic marking*.

2. The usual firing rule can directly apply on the split representation. The only difference is that dynamic subclasses of cardinality 1 substitute colors in the transition instance.

3. After the symbolic firing, a canonical representation is computed that is minimal and unique for every symbolic marking.

A symbolic transition instance represents several ordinary transition instances departing from any ordinary marking in the symbolic marking. Assigning a dynamic subclass to a transition parameter means assigning *any* element in that subclass to the parameter: it can be shown that this is correct because all the represented ordinary firings lead to the same destination symbolic marking.

3. SYMBOLIC QUOTIENT GRAPH

A symbolic quotient graph is a graph whose nodes are symbolic markings and whose edges represent symbolic firings among these nodes. We denote by *SM* (resp. *SE*) the set of symbolic markings (resp. firings).

3.1 Generic Symbolic Quotient Graph

Algorithm 1 $\mathcal{SQG}(\mathcal{N}, s_0)$

```
1: NewSt : 2^SM
2: Succ, Succ', RefSt : 2^SM
3: Edges, Edges' : 2^SE
4: Inst : Set of Symbolic Firing Instances
5: Graph.Nodes = {s0}; Graph.Edges = ∅
6: NewSt.push(s0)
7: while NewSt ≠ ∅ do
8:     s1 = NewSt.Pick()
9:     RefSt = CompSymbRef(s1, Part_a)
10:    Succ = ∅; Edges = ∅
11:    for s1k ∈ RefSt do
12:        for t ∈ N.T do
13:            Inst = CompSymbInst(s1k, t)
14:            for i ∈ Inst do
15:                s2 = CompSymbSucc(s1k, i)
16:                Succ ⊎ = {s2}
17:                Edges ⊎ = {s1 --lab(s1k,i)--> s2}
18:            end for
19:        end for
20:    end for
21:    ⟨Succ', Edges'⟩ = Optimize(Graph, Succ, Edges)
22:    NewSt.Add(Succ')
23:    Graph.Nodes ⊎ = Succ'
24:    Graph.Edges ⊎ = Edges'
25: end while
26: return Graph
```

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

Algorithm 1 presents a generic reachability graph construction for SWN models. It aims at giving a homogeneous overview of the different symbolic structures (quotient graphs) that can be obtained from an SWN. As such, it is not optimal since it cannot take into account optimizations that would be suitable for only a subset of structures.

As standard reachability construction algorithms, this one takes, as inputs, a net $\mathcal{N}$ and an initial marking s_0, and iteratively applies the firing rule on each new reachable state until saturation (i.e., until no new state is constructed). However, unlike standard algorithms, this algorithm handles *symbolic* states and firings.

In order to explain how it affects the construction, let us first recall that each symbolic state s constructed by Algorithm 1 is a pair $s = \langle \hat{m}, \mathcal{P}art \rangle$, whose elements are denoted by $s.\widehat{m}$ and $s.\mathcal{P}art$.

The symbolic firing rule (of section 2.3) operates on symbolic states whose associated partition is $\mathcal{P}art_a$. Using this partition makes it possible to build a graph on which we can check properties we are interested in. Hence, we will never handle symbolic states whose associated partition is finer than $\mathcal{P}art_a$: the representation would be less efficient without bringing any new valuable information. Yet, we may be interested in using coarser partitions and have a more compact representation of the state space while preserving properties under consideration.

In this case, to apply our symbolic firing rule, we need to get back to $\mathcal{P}art_a$. This is the goal of function $CompSymbRef$ on line 9: it rewrites a symbolic marking s_1 according to a finer partition, $\mathcal{P}art_a$ in this case. This operation is called *symbolic refinement* and generates a set of symbolic markings representing a partition of $[s_1]$:

$$RefSt = \{s_{1k} = \langle \hat{m}_{1k}, \mathcal{P}art_a \rangle\}.$$

Each symbolic marking s_{1k} of $RefSt$ fulfills the condition $s_{1k}.\mathcal{P}art = \mathcal{P}art_a$, and the classical symbolic firing rule, with the computation of successors, can be applied to it. Lines 11-20 perform this computation for each symbolic state s_{1k}. Function $lab(s_{1k}, i)$ of Algo. 1 returns the following value:

$$lab(s_{1k}, i) = \mid [s_{1k}] \mid . \mid [i] \mid .\mathbf{w}_t(i)$$

where $\mathbf{w}_t$ is the (constant) rate function of transition t. As we will see later, this label is used to generate a CTMC of the studied system.

As we handle symbolic markings with possibly different associated partitions, the problem of *non-empty intersection* among the represented classes occurs. Several strategies can be considered, depending on the property that we want to check/preserve on the resulting structure. For instance, for checking the reachability property, we can consider the possibility of removing some existing nodes if the set of states that they represent is included in the set of states represented by a new symbolic marking, while the lumpabiltiy property can be affected by this operation. The application of the chosen strategy is assigned to function *Optimize* (line 21).

Section 3.2 discusses the instantiations of function *Optimize* for the SRG and the DSRG.

3.2 Instantiations of the generic algorithm

We discuss here on the existing instantiations of the generic algorithm. They correspond to different trade-offs between the size of the generated quotient graph and the needed effort for the satisfaction/preservation of exact lumpability.

SRG: the construction starts from a symbolic marking whose associated partition is $\mathcal{P}art_a$. The symbolic firing rule applies directly and does not affect the partition. Thus, we choose $\mathcal{P}art_a$ as the associated partition for every symbolic marking. As a consequence, strict inclusion is not possible between any two symbolic markings, only equality must be considered. The edge resulting from a symbolic firing is added to the graph, together with the head node.

The advantage of this construction is its simplicity : neither function *CompSymbRef*, nor function *Optimize* is needed. Exact lumpability is directly guaranteed, provided that the performance parameters of the system are defined at the static subclass level rather than the object level. Unfortunately, when $\mathcal{P}art_a$ is partitioned into many subsets of small size, a symbolic marking represents only a small number of markings and the efficiency of the method is poor [12, 7].

However, it may not be necessary to distinguish objects belonging to different elements of $\mathcal{P}art_a$ throughout the construction of the graph: it is often the case that in several states of the system, the element of $\mathcal{P}art_a$ to which an object belongs is not relevant. The DSRG exploits the possibility of defining equivalence among objects on a dynamic basis, i.e., the equivalence relation between states is reconsidered after every firing.

DSRG: the partition associated with the initial symbolic marking is $\mathcal{P}art_s$. Before applying the symbolic firing rule, a symbolic marking is rewritten as a set of symbolic markings whose associated partition is $\mathcal{P}art_a$. This rewriting is done by function *CompSymbRef*. For each of the resulting symbolic markings, a set of enabled symbolic instances is computed (function *CompSymbInst* at line 13) and the corresponding symbolic firings are executed.

However, the symbolic refinement may break symmetries that actually remain valid after the firings. To optimize the size of the graph, we must retrieve these symmetries.

The idea is to consider the elements in *Succ* and see if some of them can be merged, so that we end up with fewer markings and associated partitions coarser than $\mathcal{P}art_a$.

Let us now give an example of this operation. Consider the net of Fig. 3, with a single color class $C = \{c_1, c_2, \ldots, c_5\}$. The guard associated with transition lcs makes a comparison between two objects of C. The only way this comparison can be expressed in the syntax of SWN is by defining a static subclass per object of C and an implicit order on static subclasses derived from the enumeration order of the elements of C. If C_i is the static subclass that contains c_i, then the SWN syntax of the guard is:
$[(d(p) = C_2 \wedge d(q) = C_1) \vee \ldots \vee (d(p) = C_5 \wedge (d(q) = C_1 \vee \ldots \vee d(q) = C_4))]$

We focus on the symbolic marking $\langle \hat{m}, \mathcal{P}art_s \rangle$ for which two objects are in place ID, one is in place RQ, and the remaining two in place GS. As the static partition is $\mathcal{P}art_s$, function d of the symbolic partition is meaningless and we use a notation for the symbolic marking in which only the cardinality of the dynamic subclass representing the objects in each place is given: $\langle \hat{m}, \mathcal{P}art_s \rangle = ID(2)+RQ(1)+GS(2)$.

A significant subset of the refinement of this marking on $\mathcal{P}art_a$ and the corresponding firings of transition lcs are represented in the shaded part of Fig. 2. When trying to group the newly obtained markings, we notice that whichever pair

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

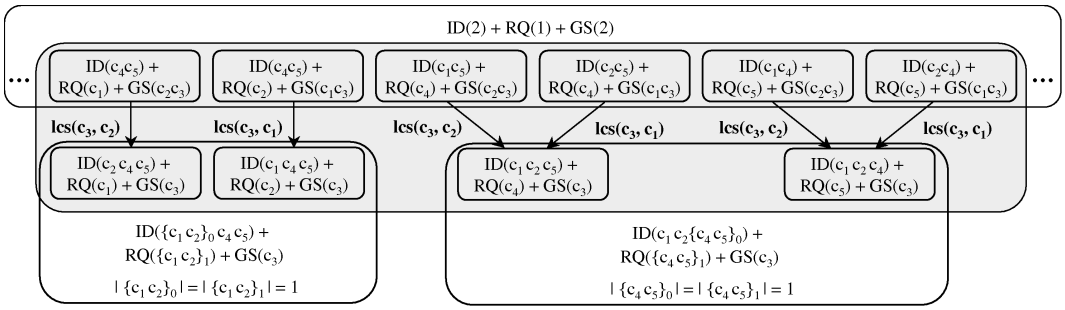

Figure 2: Refinement and grouping

of markings we consider, there exists a permutation between them. However, for grouping states, we need that also the exact lumpability condition be satisfied. It means that the states must be reached from markings belonging to the same symbolic marking (which is true), with the same input rate. The two markings on the left are reached with a rate $\mathbf{w}_{lcs}$, while the two on the right have an input rate $2.\mathbf{w}_{lcs}$, thus restraining the possibilities of grouping. We end up with two symbolic markings, one with an associated partition $\mathcal{P}art = \{\{c_1, c_2\}, \{c_3\}, \{c_4\}, \{c_5\}\}$, the other one with $\mathcal{P}art = \{\{c_1\}, \{c_2\}, \{c_3\}, \{c_4, c_5\}\}$. The information in the shaded part is then removed, and only what is in the white portion is actually stored in the graph.

The notation $\{c_i, c_j\}_k$ represents here a dynamic subclass Z^k that is instantiated in $\{c_i, c_j\}$: in the left symbolic state, $d(Z^0) = d(Z^1) = 1$ (i.e., both are instantiated in the first element of $\mathcal{P}art$), $card(0) = card(1) = 1$, meaning that either c_1 or c_2 is in RQ, the other one is in place ID.

Formally, the recovering of lost symmetries is performed by function *Optimize_DSRG* (Algorithm 2) that instantiates function *Optimize* of the generic algorithm. It calls function *CompSymbLump* that

- examines $Succ$ to find a set $\{s_{2k}\}_k$ which can be grouped from a qualitative point of view :

$$\exists \langle \hat{m}, \mathcal{P}art \rangle, [\langle \hat{m}, \mathcal{P}art \rangle] = \bigcup_k [s_{2k}] \tag{1}$$

- with the additional constraint that all s_{2k} must have identical input rates from states of $RefSt$. The rate from s_1 to s' is computed using the following formula:

$$rate(s_1, s') = \frac{|\,[s']\,|}{|\,[s_1]\,| \,.\, |\,[s'']\,|} \cdot \sum_{s_1 \xrightarrow{lab(s_{1k},i)} s'' \in Edges} lab(s_{1k}, i) \tag{2}$$

  where s'' is any of the successors of s_1 that have been grouped to obtain s'.

The result is a set $Succ'$ of nodes that will be added to the graph. Input arcs of elements that have been grouped are redirected on the corresponding aggregate.

The effect of this local optimization on the size of the graph is hard to predict. In fact, such a *symbolic grouping* may generate a phenomenon of *non-empty intersections* between symbolic states, leading to several occurrences of a marking in the final graph.

For example, starting from three markings $\{s_1, s_2, s_3\}$, we can end up with four symbolic markings: a separated grouping of $Set_0 = \{s_1, s_2, s_3\}$, $Set_1 = \{s_1, s_2\}$, $Set_2 = \{s_1, s_3\}$ and $Set_3 = \{s_2, s_3\}$, reached by different firings, may end up with the construction of four symbolic states S_0,S_1,S_2 and S_3, which are not disjoint.

Algorithm 2 *Optimize_DSRG*$(Graph, Succ, Edges)$

```
1: Succ' : 2^SM;
2: Edges' : 2^SE;
3: Edges' = Edges;
4: Succ' = CompSymbLump(Succ, Edges);
5: for s' ∈ Succ' do
6:     Ed = {s -*-> s'' ∈ Edges' | [s''] ⊆ [s']};
7:     Edges' = (Edges' \ Ed) ∪ {s -rate(s,s')-> s'};
8:     if s' ∈ Graph.Nodes then
9:         Succ' = Succ' \ {s'};
10:    end if
11: end for
12: return ⟨Succ', Edges'⟩;
```

Although we are not yet able to predict the efficiency of the approach on a given example, many experiments show that it can improve significantly the size of the representation with respect to the RG. However, in some cases, the size of the resulting symbolic graph is **greater** than the size of the underlying RG.

By not considering the lumpability condition when trying to regroup states after a set of firings, i.e., by weakening the condition used by function *Optimize_DSRG*, we may dramatically reduce the size of the final symbolic graph. Actually, lumpability prevents us from considering strict inclusion between sets of states, because if two sets independently satisfy a lumpability condition and one is strictly included in the other, then their superposition cannot satisfy it.

Hence, the idea is to allow taking into account inclusion in function *Optimize* so as to reduce the size of the resulting structure, and then apply a refinement algorithm to obtain a (lumped) Markov chain.

A such, two-step, approach has already been used in the so-called Extended SRG (ESRG) construction for deriving a lumped Markov chain starting from a quotient reachability graph. However, this approach uses only partitions $\mathcal{P}art_a$ and $\mathcal{P}art_s$, resulting in a domino effect in the refinement and a peak in memory usage [4].

4. THE PROPOSED APPROACH

The *Partially Lumped DSRG (PLDSRG)* is a preliminary

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

structure on which a refinement algorithm can be applied to obtain a fully lumped CTMC. The *PLDSRG* is said to be partially lumped because it may happen that some of its states do not satisfy the lumpability condition.

Algorithm 3 *Optimize_PLDSRG*$(Graph, Succ, Edges)$

1: $Succ = CompSymbLump(Succ, Edges)$
2: **for** $s' \in Succ$ **do**
3: $\quad Edges = (Edges \setminus \{sr \xrightarrow{*} s'' \mid [s''] \subseteq [s']\}) \cup \{sr \xrightarrow{rate(sr,s')} s'\}$
4: $\quad$ **if** $\exists v \in Graph.Nodes$ s.t. $[s'] \subseteq [v]$ **then**
5: $\quad\quad Succ = Succ \setminus \{s'\}$
6: $\quad\quad$ **if** $[s'] == [v]$ **then**
7: $\quad\quad\quad Edges = (Edges \setminus \{sr \xrightarrow{rate(sr,s')} s'\}) \cup \{sr \xrightarrow{rate(sr,s')} v\}$
8: $\quad\quad$ **else**
9: $\quad\quad\quad Edges = (Edges \setminus \{sr \xrightarrow{*} s'\}) \cup \{sr \overset{*}{\rightsquigarrow} v\};$
10: $\quad\quad$ **end if**
11: $\quad$ **else**
12: $\quad\quad Set = \{v \in Graph.Nodes \mid [v] \subset [s']\}$
13: $\quad\quad$ **if** $Set \neq \emptyset$ **then**
14: $\quad\quad\quad Ed_{in} = \{s \xrightarrow{*} v \mid v \in Set\}$
15: $\quad\quad\quad Ed_{out} = \{v \xrightarrow{*} s \mid v \in Set\}$
16: $\quad\quad\quad Set_{in} = \{s \mid s \xrightarrow{*} v \in Ed_{in}\}$
17: $\quad\quad\quad Set_{out} = \{s \mid v \xrightarrow{*} s \in Ed_{out}\}$
18: $\quad\quad\quad Graph.Nodes \setminus = Set$
19: $\quad\quad\quad Graph.Edges \setminus = (Ed_{in} \cup Ed_{out})$
20: $\quad\quad\quad Edges = Edges \cup \{s' \xrightarrow{l} s \mid v \xrightarrow{l} s \in Ed_{out}\}$
21: $\quad\quad\quad Edges = Edges \cup \{s \overset{*}{\rightsquigarrow} s' \mid s \in Set_{in}\}$
22: $\quad\quad$ **end if**
23: $\quad$ **end if**
24: **end for**
25: **return** $\langle Succ, Edges \rangle$

The generation of a *PLDSRG* requires the introduction of a new *Optmize* function, called *Optimize_PLDSRG* and depicted in Algo. 3. Like *Optimize_DSRG*, this function uses *CompSymbLump* to group symbolic markings (line 1). If one of the obtained states, namely s', had been already visited (lines 4-10), then only its input rate is added to the graph (line 7), in case $[s'] == [v]$. If $[s'] \subsetneq [v]$ then a specially annotated arc, $\overset{*}{\rightsquigarrow}$, is used to notify that there is an inclusion relation (line 9). The case where a s' is itself including a set of states of the graph is treated in lines 11-21: The whole set of states Set, s.t. $\bigcup_{s \in Set}[s] \subset [s']$, is replaced by s' and, as consequence, the input/output edges are updated. Here also, it appears an inclusion relation that must be notified on the resulting structure (line 21).

Trivially, all those states that are linked by inclusion arcs (arcs of the form $\rightsquigarrow$) do not satisfy the lumpability condition. Then, to obtain a (fully lumped) CTMC, the *PLDSRG* must be refined using these arcs as guides. We call F(ully) L(umped) DSRG the resulting structure.

4.1 The algorithm for checking exact lumpability

We describe here an extension of Paige and Tarjan's partition refinement algorithm [11], for the exact lumpability check of state aggregations induced by the *PLDSRG*. With respect to Paige and Tarjan's algorithm this extension uses a different aggregation condition (the exact lumpability one) and works using the information contained in the *PLDSRG*.

Algorithm 4 Algorithm for the exact lumpability check

1: $B, D : 2^{SM}$
2: $A, X : 2^{2^{SM}}$
3: $Succ : 2^{(SM \times \mathbb{R} \times \mathbb{N})}$
4: $PartSucc : 2^{(2^{SM} \times \mathbb{R} \times \mathbb{N})}$
5: $X.Create(PLDSRG)$
6: $A = X.PreSplit(\mathcal{P}art_a)$
7: **while** $X \neq A$ **do**
8: $\quad D = X.Remove()$ s.t. $\forall A_i \in A, A_i \neq D$
9: $\quad B = A.Pick(D)$ s.t. $B \subset D \wedge \forall A_i \subset D, |B| \geqslant |A_i|$
10: $\quad X.Add(\{B, D \setminus B\})$
11: $\quad Succ = CompAllSymbSucc(B, \mathcal{P}art_a)$
12: $\quad PartSucc = PartWrtRateA(Succ)$
13: $\quad A.Split(PartSucc)$
14: **end while**
15: **return** A

Algorithm 5 Algorithm of the *Split* function

1: $Set, A_j : 2^{SM}$
2: **for** $\langle S, rate, j \rangle \in PartSucc$ **do**
3: $\quad Set = \emptyset$
4: $\quad A_j = GetElement(j)$
5: $\quad Set = \biguplus_{s \in A_j} CompSymbRef(s, \mathcal{P}art_a)$
6: $\quad Set = Set \setminus S$
7: $\quad Set = CompSymbGroup(Set)$
8: $\quad S = CompSymbGroup(S)$
9: $\quad A_j = Set$
10: $\quad Add(S)$
11: $\quad UpdateEdges(S)$
12: $\quad UpdateEdges(A_i)$
13: **end for**

The stability condition of Paige and Tarjan's algorithm is weaker than the exact lumpability one, and is implied by the latter. In fact, the exact lumpability condition does not only check that all elements in each aggregate are reached by the same source aggregates, but it also requires that they are reached with the same rate.

The algorithm presented in this section exploits the aggregations suggested by the *PLDSRG*, rather than blindly applying state aggregation to the RG. This has two advantages, in the initialization of X and A, and in the total memory usage.

Algorithm 4 depicts the pseudo-code of the algorithm. It has two main phases: the initialization (lines 5-6) and the iterative refinement phase (lines 7-13).

The initial phase. The *PLDSRG* nodes are partitioned according to the following invariant:

$$\begin{array}{c} \forall X_i \in X, \forall s_1, s_2, \quad s_1 \in X_i \wedge s_2 \in X_i \Leftrightarrow \\ \exists s, \; s.\mathcal{P}art = \mathcal{P}art_s \wedge [s_1] \subseteq [s] \wedge [s_2] \subseteq [s] \end{array} \tag{3}$$

In other words, every element X_i contains all the symbolic states that could be represented by s. Note that s may not be a node of *PLDSRG*. Function *Create* operates this partitioning (line 5).

PreSplit (line 6) returns a refinement A of X, such that each element $A_i \in A$ satisfies the exact lumpability condi-

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

tion w.r.t. each element $X_j \in X$:

$$\forall s_1, s_2 \in A_i, \sum_{s_k \in X_j} rate(s_k, s_1) = \sum_{s_k \in X_j} rate(s_k, s_2) \quad (4)$$

This splitting is performed by dividing those sets of X that are reached by some *inclusion arcs*. This is sufficient to ensure condition 4, because blocks without any inclusion arcs already satisfy condition 4. The splitting requires to symbolically refine on $\mathcal{P}art_a$[1] the elements of the considered X_i and compute their input rates. Then in A, X_i is split in a set of sets $A_k : X_i = \biguplus_k A_k$ (line 6).

The iterative refinement phase. The algorithm core consists in repeating a *refinement step* until X converges to A ($X = A$). This step is performed as follows: in X, an element D that has been refined in a previous step is selected (line 8), then the largest (in terms of number of contained items) element $B \in A$ s.t. $B \subseteq D$ is chosen (line 9). Finally, X is updated by replacing D with the set $\{B, D \setminus B\}$ (line 10).

All successors of B are computed. This requires to refine on $\mathcal{P}art_a$ all the elements included in B and to store the following information in $Succ$ (line 11): the successor, the rate with which it is reached and the index of the A-element that contains it. Then, function $PartWrtRateA$ performs a partitioning $PartSucc$ of $Succ$ by grouping the tuples with the same second and third element.

At this point, A must be refined according to the new partition represented by $PartSucc$, as described in Algo 5: for each element $\langle S, rate, j\rangle \in PartSucc$, we refine all symbolic markings of A_j on $\mathcal{P}art_a$ and store them in Set (line 5). Then, we remove elements of S from Set. Finally the obtained sets Set and S are symbolically grouped (lines 7-8): function $CompSymbGroup$ checks whether subsets of its parameter satisfy condition (1).

Observe that function $CompSymbGroup$ is crucial for preventing the fragmentation of the representation of symbolic markings. Any symbolic marking that is affected by Algorithm 4 is split in a set of symbolic markings with associated partition $\mathcal{P}art_a$. In models where symmetries are not global, static subclasses are often reduced to singletons. Hence, without function $CompSymbGroup$, the size of the $FLDSRG$ tends to the size of the RG.

5. CASE STUDY

In this section, a distributed critical section (DCS) model and a client-sever (CS) algorithm are studied. These two models are locally symmetric: in both cases the colored elements distinguish their behavior only when some specific conflict situations must be solved.

A distributed critical section model: The SWN of Fig. 3 models a system where a finite set of processes, whose identifiers are represented by color class C, are in competition to access a critical section. The color domain of all the places of the net is C. As there is a single class, the constant function representing the set of all processes is simply denoted by S.

Initially, all processes are idle (place ID), meaning that they do not request the critical section. The firing of transition rcs represents a process requesting the critical section. As soon as a request is accepted and reaches the selection phase (place GS), no new request can be accepted: permissions for applying are removed from place PR by the firing of fr (with priority over other transitions).

If there are several candidates, i.e., the number of tokens in places RQ and GS is greater than one, a selection is performed based on the identifiers: successive firings of transition lcs eliminate all candidates but the one with the highest identifier. Actually, the function labelling the arc from place FDR to transition wcs prevents a process from entering the critical section (firing of wcs) until all the tokens representing other processes are in place FDR, meaning that either the corresponding process did not apply, or its request was discarded because at least one process with higher identity applied too.

When a process releases the critical section (firing of transition ecs), all processes become idle again and a new round can start.

A client/server model: the system represented by the SWN of Fig. 4 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 client can join the wave anymore (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

[1]The symbolic refinement used in Algo. 1.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

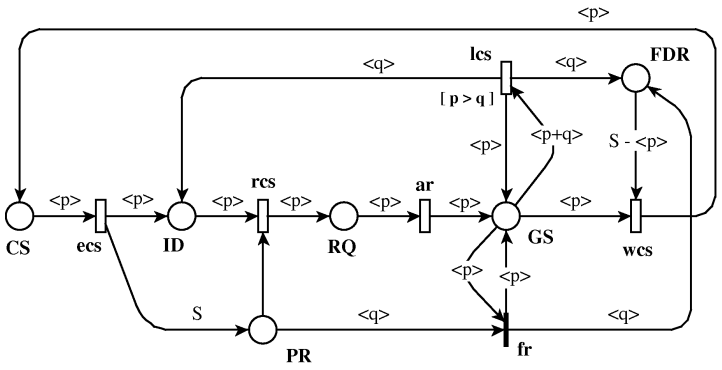

Figure 3: SWN of the Distributed Critical Section

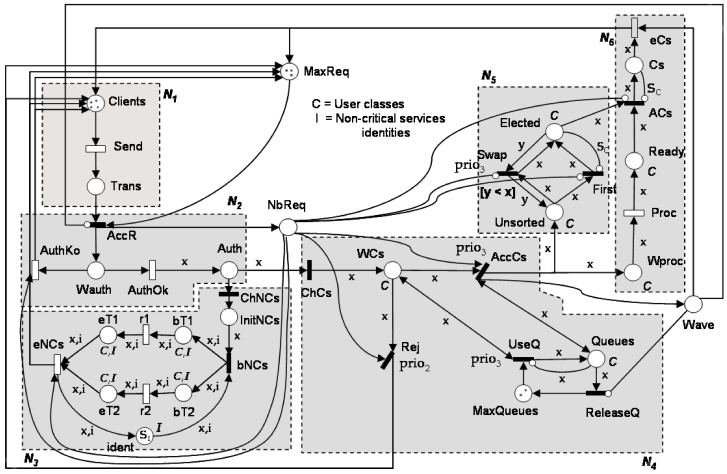

Figure 4: SWN of the Client/Server System.

subnet N_5). If there is no priority among 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 asymmetric *guarded* transition $Swap$ (observe that $[y < x]$ is again not standard SWN syntax, but this is only syntactic sugar). 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 (which is denoted $prio_3$).

5.1 Experiment results

To experiment our methods, we have implemented our algorithms on top of GreatSPN's kernel. In our tool, one can specify an SWN, and use different options to obtain the results for the described constructions. The machine used for our tests is a PC/Linux of 3.2 GHz and 3 Gb of RAM.

Tables 1 and 2 summarize the results, in terms of size of the lumped Markov chain, obtained on the two examples for different approaches: SRG, Refined ESRG (RESRG), DSRG, PLDSRG and FLDSRG. The RESRG is the CTMC obtained from the ESRG.

Column P represents the used parameter for each model. For the DCS model, this corresponds to the number of processes involved in the algorithm. For the CS model, and for sake of simplicity, all the parameters of the algorithm are set to the same value P.

Among the graphs that satisfy a lumpability condition, these examples show that the approach introduced in this paper, i.e., FLDSRG, improves the size of the resulting graph. For the DCS example, the reduction w.r.t. the DSRG is not very significant. Actually, there is a strong synchronization among processes at the end of the critical section, as they all return to idle state. Hence a few nodes with non-null intersections are constructed.

In the Client/Server system, the asymmetric part is represented by subnet N_5. Each firing of transition $Swap$ may partition the processes in a different way: this partitioning is propagated along every future action in the net. Hence, many groups of states with non-null intersections are created and stored in the DSRG, while the FLDSRG can group them.

It is worth noticing that the intermediate structure, i.e., PLDSRG, can be dramatically small. On the DCS exam-

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733

P	SRG	RESRG	DSRG	PLDSRG	FLDSRG
6	2.001	688	444	44	424
8	23.041	6.343	4.150	74	3.748
10	250.769	58.082	40.320	112	34.004
12	2.632.641	527.425	394.500	158	316.084
14	27.655.686	4.757.825	3.893.311	166	3.113.024

Table 1: Results for the DCS model.

P	SRG	RESRG	DSRG	PLDSRG	FLDSRG
2	265	147	147	144	147
4	82.978	9.020	7.922	7.070	7.856
6	26.501.875	368.571	317.996	215.954	275.564
8	-	-	3.497.956	1.501.268	2.074.648
10	-	-	11.956.692	4.811.931	6.326.812

Table 2: Results for the Client/Server model.

ple, this is due to the fact that asymmetries affect a limited part of the graph, i.e., the selection for entering the critical section. As soon as a process leaves the critical section, all processes come back to idle state. Although only intermediate in our approach, PLDSRG is a useful structure that can be used to check reachability or the existence of dead states.

6. CONCLUSION

In this paper, we presented an approach for automatically deriving a lumped Markov chain from a Stochastic Well-formed Net. Compared with existing approaches, this one looks more efficient when the system under consideration is locally symmetric, meaning that some of its components almost always behave in a similar way.

The gain in efficiency depends on the way the asymmetric and symmetric parts of the system are connected. We presented an example of distributed critical section where processes "synchronize" at the end of a critical section (they all end up in the same idle state), thus blocking the propagation of asymmetries in the construction of the graph. In such a situation, the gain is not significant compared with some existing approaches. However, when asymmetries are likely to hold in many situations, which is usually the case in real systems, the FLDSRG construction compares favourably to other approaches.

Among the different approaches that exist for building a lumped Markov chain from an SWN model, we are not yet able to predict which one will be the most efficient. However, it is possible to run all of them in parallel and gather the results from the first that succeeds.

We plan now to investigate the possibility of detecting at the SWN model structural level whether, for some given system, one of these approaches is likely to give better results than the others. A discussion on this topic can be found in [2]: this is based on four case studies that can be considered as *model patterns*, representative of typical locally symmetric models.

7. REFERENCES

[1] S. Baarir, M. Beccuti, D. Cerotti, M. De Pierro, S. Donatelli, and G. Franceschinis. The greatspn tool: recent enhancements. *SIGMETRICS Perform. Eval. Rev.*, 36(4):4–9, 2009.

[2] S. Baarir, M. Beccuti, C. Dutheillet, G. Franceschinis, and S. Haddad. Performance analysis of partially symmetric SWNs: efficiency characterization through some case studies. Technical Report TR-INF-2009-07-06-UNIPMN, Dip. di Informatica, Univ. del Piemonte Orientale, 2009. http://www.di.unipmn.it/index.php/technical_report.php.

[3] S. Baarir, C. Dutheillet, S. Haddad, and J.-M. Ilié. On the use of exact lumpability in partially symmetrical Well-formed Nets. In *Proc. of the 2nd Int. Conference on the Quantitative Evaluation of Systems*, pages 23–32, Torino - Italy, September 2005.

[4] M. Beccuti, S. Baarir, G. Franceschinis, and J.-M. Ilié. Efficient lumpability check in partially symmetric systems. In *Int. Conf. on Quantitative Evaluation of Systems*, Riverside, CA, USA, Sept. 2006.

[5] 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, November 1993.

[6] E. A. Emerson and A. P. Sistla. Symmetry and Model Checking. *Formal Methods and System Design*, 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 *CHARME*, pages 142–156, 1999.

[8] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In *LNCS 188*, pages 215–233. Springer-Verlag, 1985.

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

[10] J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. VanNostrand, 1960.

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

[12] S.Haddad, J.M Ilié, M. Taghelit, and B. Zouari. Symbolic Reachability Graph and Partial Symmetries. *16th Int. Conference on Application and Theory of Petri Nets. Turin, Italy.*, 935:238–251, June 1995.

[13] W.D. Obal II and W.H. Sanders. Measure-adaptive state-space construction. *Performance Evaluation*, 44(1-4):237–258, 2001.

[14] W.H. Sanders and J.F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25–36, 1991.

[15] GreatSPN tool's web page. http://www.di.unito.it/∼greatspn.

Digital Object Identifier: 10.4108/ICST.VALUETOOLS2009.7733
http://dx.doi.org/10.4108/ICST.VALUETOOLS2009.7733