Electronic Notes in Theoretical Computer Science 39 No. 3 (2000)
URL: http://www.elsevier.nl/locate/entcs/volume39.html 27 pages

Exploiting Partial Symmetries for Markov Chain Aggregation

L. Capra[a] C. Dutheillet[b] G. Franceschinis[c,1] J-M. Ilié[b]

[a] *Dip. di Informatica, Universitá di Torino, Italy,* lorenzoc@di.unito.it

[b] *LIP6, Université Paris VI, France,* Claude.Dutheillet@lip6.fr

[c] *DISTA, Univ. del Piemonte Orientale, Alessandria, Italy,* giuliana@di.unito.it

Abstract

The technique presented in this paper allows the automatic construction of a lumped Markov chain for almost symmetrical Stochastic Well-formed Net (SWN) models. The starting point is the Extended Symbolic Reachability Graph (ESRG), which is a reduced representation of a SWN model reachability graph (RG), based on the aggregation of states into classes. These classes may be used as aggregates for lumping the Continuous Time Markov Chain (CTMC) isomorphic to the model RG: however it is not always true that the lumpability condition is verified by this partition of states. In the paper we propose an algorithm that progressively refines the ESRG classes until a lumped Markov chain is obtained.

Key words: Lumpable Markov Chains, Extended Symbolic Reachability Graph, High-level Petri Nets, Symmetries.

1 Introduction

Very few analysis techniques make it possible to obtain detailed information on the behaviour of a system without representing its state transition graph. As the complexity of systems increases, fighting the combinatorial explosion of the state space becomes more and more critical.

A possible approach to attack this problem is the exploitation of behavioural symmetries, from which quotient graphs can be computed. Such graphs offer a compact representation of the state space as their nodes are no longer states but classes of states of the system. They can be used both for

[1] This work was partially supported by the Italian MURST 60% and by the EEC project n. 28620 TIRAN.

quantitative (e.g. performance evaluation [3]) and qualitative (e.g., as a basis for model checking [6]) evaluation of systems.

Well-formed Nets (WN) are a Petri net based formalism from which a quotient graph, called Symbolic Reachability Graph (SRG), can be computed automatically. The SRG nodes are sets of equivalent (ordinary) markings. The SRG has proved useful for analysing basic qualitative properties of the system, e.g. deadlock-freeness or reversibility [4], and temporal logic properties [7].

The presence of behavioural symmetry in a model can be exploited also for performance evaluation purposes: in particular the performance analysis of Stochastic Well-formed Net (SWN) models can be performed by generating a Markov chain of the same size as the Reachability Graph (RG) of the model. However a *lumped* CTMC can be directly derived from the SRG, and any performance result that could be obtained by solving the (usually much larger) complete CTMC can also be obtained from the lumped one [5].

Unfortunately, the gain can be relevant only if the models are highly symmetric. In practice it is often the case that a system behaves in a symmetric way in most situations, while in exceptional situations asymmetries arise. The SRG approach to symmetry exploitation is such that an exceptional asymmetry may destroy any possibility of exploiting symmetries. For this reason an extension of the SRG technique (Extended SRG -ESRG) was proposed in [8].

The ESRG allows to exploit the symmetries whenever possible, and to deal with asymmetries only when they arise. This approach can lead to remarkably better results (in terms of state space reduction) than the SRG approach, especially in those models where the asymmetric behaviour is not so frequent.

The drawback of the proposed technique, in its original formulation, is a loss of information that in some cases makes it impossible to prove some properties and to directly build a lumped Markov chain for performance analysis purposes. In [2] we made a first step to overcome the problem of using the ESRG for performance analysis purposes: in fact we presented an incremental approach to the refinement of the information contained in the ESRG to check the *ergodicity* of the underlying stochastic process. In this paper we make one step further by showing how a lumped MC can be derived from the ESRG, avoiding as much as possible its refinement.

The paper is organized as follows. Section 2 is a primer on the symbolic reachability graph and Section 3 introduces through an example the basic concepts of the extended symbolic reachability graph. The main definitions and properties related to the derivation of a lumped Markov chain from the SRG are presented in Section 4. The extension of some of these properties to the ESRG is also given in this section. The algorithm proposed in Section 5 transforms the ESRG by recursively splitting some extended symbolic markings, so as to ensure that the new aggregates will satisfy the lumpability condition. Section 6 explains how to compute the rates of the lumped Markov chain obtained with our algorithm. Section 7 concludes and gives some perspectives to this work.

2 A primer on the symbolic reachability graph

Stochastic Well-formed Nets (SWN) belong to the class of high-level (stochastic) Petri nets, where tokens carry information taken from a finite set of colours. In SWN, colour domains and colour functions must respect a simple, rigorous syntax, in order to give the possibility of directly building a Symbolic Reachability Graph (SRG).

The Appendix briefly presents the SWN formalism for the readers' convenience; refer to [3,4] for the complete definition of SWNs and of the Symbolic Reachability Graph algorithm.

SWN tokens carry a composite information expressed as a tuple of colours (called objects), taken from possibly ordered *basic colour classes*.

Each colour class represents system components of a given kind (e.g. the process class, the processor class,...), and can be partitioned into static subclasses: objects in the same static subclass represent entities that always behave in a symmetric (homogeneous) way, while colours belonging to different static subclasses may occasionally exhibit a different behaviour. A colour class partitioned into static subclasses is called *distinguished* class, and a SWN model with distinguished classes is said *partially symmetrical*.

The SRG approach is based on the systematic and automatic exploitation of *symmetries*, which consists in *lifting* the detail level of the state description (ordinary marking) to a *more abstract* one, called *symbolic marking* (SM). For the sake of presentation simplicity, let us imagine that there is a unique basic colour class C.

Intuitively, in a SM representation, objects are replaced with *symbols* which stand for *any object* in class C: the resulting representation can be interpreted as a pattern common to all ordinary markings represented by that SM. If C is partitioned into static subclasses, each symbol appearing in the SM representation must be assigned to a static subclass (C_i) of C, and it will represent an arbitrarily chosen object of C_i. Different symbols represent different objects.

For efficiency, when a set of k different symbols appear with the same multiplicity in all places of a given SM, they can be substituted by a new symbol, denoted Z^j, characterized by its *cardinality* (k). For example let us consider the symbolic representation $p1(x_1 + x_2 + x_3 + 2\ x_4 + 2\ x_5) + p2(3x_1 + 3\ x_2 + 3\ x_3)$: we can substitute set $\{x_1, x_2, x_3\}$ with the new symbol Z^1, such that $|Z^1| = 3$, and set $\{x_4, x_5\}$ with the new symbol Z^2, such that $|Z^2| = 2$, so that the new, more compact SM representation is $p1(Z^1 + 2\ Z^2) + p2(3\ Z^1)$. The symbol Z^j just introduced is called *dynamic subclass*, and stands for any subset of $|Z^j|$ objects in C. Again if C is partitioned into static subclasses each dynamic subclass Z^j must be explicitly associated with a static subclass C_i (through a function $d()$), and in this case it represents an arbitrarily chosen subset of C_i of cardinality $|Z^j|$. Obviously it must hold $\forall C_i\ : \sum_{Z^j\ : d(Z^j) = C_i} |Z^j| = |C_i|$.

SMs represent in a very compact manner sets of *equivalent* ordinary markings, where equivalent means that they can be obtained from each other by permutations on colour classes preserving static subclasses. Another important feature of SMs is that all ordinary markings grouped in the same SM enable equivalent sets of transition instances. Moreover, if we fire equivalent transition instances from equivalent markings, the reached markings are still equivalent: this leads to the notion of *symbolic transition instance*, which is a symbolic representation of equivalent instances.

Since in SMs symbols replace objects, then a symbolic instance enabled in SMs will associate symbols to the transition variables, instead of objects. Through a structured symbolic firing rule [3], it is thus possible to directly build the SRG, starting from a symbolic initial marking.

Since the aggregation performed by SMs must preserve static subclasses, the effectiveness of the SRG may be dramatically reduced for partially symmetrical models. In the extreme case of basic classes all partitioned into cardinality one static subclasses, the SRG corresponds to the ordinary reachability graph (RG), as explained in the Appendix.

So it was proposed to *ignore* the partition into static subclasses when not necessary, allowing a new symbolic representation, called *Extended Symbolic Marking* (ESM), even more abstract than the one provided by the SM definition. A ESM groups together similar SMs, namely SMs which have the same representation, when the partition into static subclasses is ignored. Of course the extension is not straightforward because if class C is split into static subclasses it is likely that sooner or later (perhaps seldom) this distinction will be needed to infer the possible model behaviour.

The basic concepts and notations of the ESRG [8] are presented in the next section and illustrated on a SWN model representing a distributed critical section algorithm.

3 The distributed critical section example

In this section we informally introduce the ESRG through an example. We also point out its possible use for performance evaluation purposes.

The SWN model in Fig. 1 represents a distributed critical section algorithm. Idle processes may independently issue requests for the critical section CS (firing of t_1). As soon as a process enters the state (place GS) where a selection among issued requests is performed (firing of t_2), the authorizations for issuing other requests (place PR) are temporary removed from all processes that are still in idle state (firing of immediate transition t_6), until a request has been served. If several requests have been sent, a selection is performed depending on the identities of the processes: all the requests, except the one with greatest sender identity, are discarded (firing of t_4). Forbidden and discarded requests are collected in place FDR. A process can enter the critical section (place CS), once place FDR contains the identities of all remaining

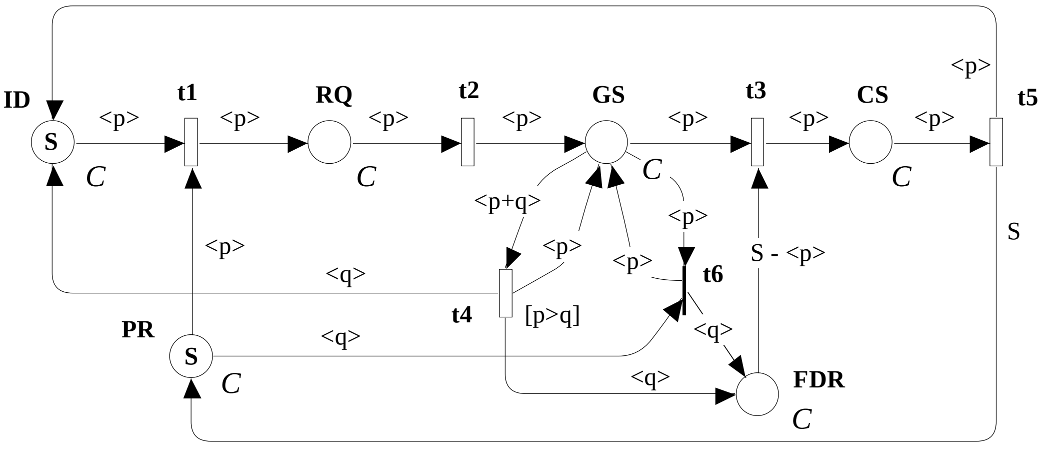


Fig. 1. Petri net model of the Distributed Critical Section algorithm.

processes (firing of t_3). After some time, the process leaves CS (firing of t_5).

In the example a single class is used, the class of processes $C = \{pr_1, pr_2, pr_3\}$.

Let us consider the guard of transition t_4, which actually is not a standard SWN guard. As the order defined on SWN colour classes is circular, it cannot be used for totally ordering the elements of C. For that, we need to partition the class in three static subclasses of cardinality one, namely $C_1 = \{pr_1\}$, $C_2 = \{pr_2\}$, $C_3 = \{pr_3\}$. Using the SWN syntax, the guard $p > q$ becomes:

$$(p \in C_2 \textit{ and } q \in C_1) \textit{ or } (p \in C_3 \textit{ and } (q \in C_1 \textit{ or } q \in C_2))$$

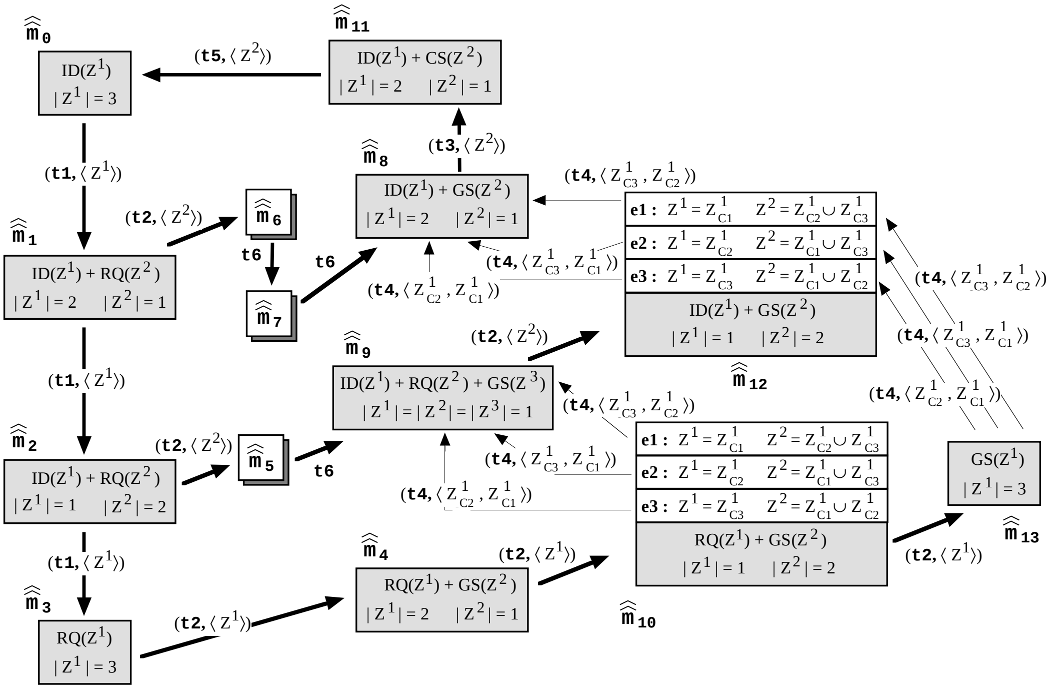
Here, all static subclasses have cardinality one. Hence we are in the situation where the SRG does not reduce the RG, since each SM corresponds to an ordinary marking.

However, only the firing/enabling of transition t_4 depends on static subclasses. This kind of transitions are called *asymmetrical*. As long as t_4 is not enabled, the net behaves as if class C had not been split, and it is not necessary to relate dynamic subclasses to static subclasses.

For instance, the ESM symbolic representation $ID(Z^1) + RQ(Z^2)$, where $|Z^1| = 2, |Z^2| = 1$, represents the class of markings with two (arbitrarily chosen) processes in place ID, and the other in RQ, e.g. $m = ID(pr_1 + pr_3) + RQ(pr_2)$. We use $\widehat{\widehat{m}}_k$ to denote ESMs, while the standard SM representation is denoted $\widehat{m}_k$.

When the refinement of state description becomes unavoidable, typically because a transition becomes enabled whose behaviour is static subclass dependent, then the ESM representation is developed, i.e., the set of SMs grouped into that ESM is generated: these SMs are called *eventualities* of the ESM and are obtained by instantiating dynamic subclasses with static subclasses.

In our examples, eventualities are represented by a new, more refined definition of dynamic subclasses to ensure that each dynamic subclass represents



$\widehat{\widehat{m}}_0$
$ID(Z^1)$
$|Z^1| = 3$
$(t5, \langle Z^2 \rangle)$
$\widehat{\widehat{m}}_{11}$
$ID(Z^1) + CS(Z^2)$
$|Z^1| = 2 \quad |Z^2| = 1$
$(t3, \langle Z^2 \rangle)$
$(t1, \langle Z^1 \rangle)$
$(t2, \langle Z^2 \rangle)$
$\widehat{\widehat{m}}_6$
t6
$\widehat{\widehat{m}}_7$
t6
$\widehat{\widehat{m}}_8$
$ID(Z^1) + GS(Z^2)$
$|Z^1| = 2 \quad |Z^2| = 1$
$(t4, \langle Z^1_{C3}, Z^1_{C2} \rangle)$
$(t4, \langle Z^1_{C3}, Z^1_{C1} \rangle)$
$(t4, \langle Z^1_{C2}, Z^1_{C1} \rangle)$
$\widehat{\widehat{m}}_1$
$ID(Z^1) + RQ(Z^2)$
$|Z^1| = 2 \quad |Z^2| = 1$
e1 : $Z^1 = Z^1_{C1} \quad Z^2 = Z^1_{C2} \cup Z^1_{C3}$
e2 : $Z^1 = Z^1_{C2} \quad Z^2 = Z^1_{C1} \cup Z^1_{C3}$
e3 : $Z^1 = Z^1_{C3} \quad Z^2 = Z^1_{C1} \cup Z^1_{C2}$
$ID(Z^1) + GS(Z^2)$
$|Z^1| = 1 \quad |Z^2| = 2$
$\widehat{\widehat{m}}_{12}$
$(t4, \langle Z^1_{C3}, Z^1_{C2} \rangle)$
$(t4, \langle Z^1_{C3}, Z^1_{C1} \rangle)$
$(t4, \langle Z^1_{C2}, Z^1_{C1} \rangle)$
$(t1, \langle Z^1 \rangle)$
$\widehat{\widehat{m}}_9$
$(t2, \langle Z^2 \rangle)$
$ID(Z^1) + RQ(Z^2) + GS(Z^3)$
$|Z^1| = |Z^2| = |Z^3| = 1$
$(t4, \langle Z^1_{C3}, Z^1_{C2} \rangle)$
$\widehat{\widehat{m}}_2$
$(t2, \langle Z^2 \rangle)$
$\widehat{\widehat{m}}_5$
t6
$ID(Z^1) + RQ(Z^2)$
$|Z^1| = 1 \quad |Z^2| = 2$
$(t4, \langle Z^1_{C3}, Z^1_{C1} \rangle)$
$(t4, \langle Z^1_{C2}, Z^1_{C1} \rangle)$
e1 : $Z^1 = Z^1_{C1} \quad Z^2 = Z^1_{C2} \cup Z^1_{C3}$
e2 : $Z^1 = Z^1_{C2} \quad Z^2 = Z^1_{C1} \cup Z^1_{C3}$
e3 : $Z^1 = Z^1_{C3} \quad Z^2 = Z^1_{C1} \cup Z^1_{C2}$
$RQ(Z^1) + GS(Z^2)$
$|Z^1| = 1 \quad |Z^2| = 2$
$GS(Z^1)$
$|Z^1| = 3$
$\widehat{\widehat{m}}_{13}$
$(t2, \langle Z^1 \rangle)$
$(t1, \langle Z^1 \rangle)$
$\widehat{\widehat{m}}_4$
$(t2, \langle Z^1 \rangle)$
$\widehat{\widehat{m}}_3$
$RQ(Z^1)$
$|Z^1| = 3$
$(t2, \langle Z^1 \rangle)$
$RQ(Z^1) + GS(Z^2)$
$|Z^1| = 2 \quad |Z^2| = 1$
$\widehat{\widehat{m}}_{10}$

Fig. 2. ESRG of the Petri net model of the DCS algorithm.

only elements of one static subclass. The new dynamic subclasses shall be denoted $Z^k_{C_i}$ where C_i indicates the static subclass of the elements of $Z^k_{C_i}$. Hereafter we shall use function $d()$ defined as $d(Z^k_{C_i}) = C_i$ to associate each dynamic subclass with a static subclass. The eventuality representation is in the form: $Z^k = Z^h_{C_i} \cup Z^m_{C_l} \cup \ldots$ which shows how the dynamic subclasses Z^k appearing in the non-refined symbolic representation are refined into dynamic subclasses $Z^j_{C_i}$ which take into account the partition into static subclasses.

The ESRG of the distributed CS model is depicted in Fig. 2. Vanishing markings are shadowed, and their representation is not reported for space reasons (anyway they do not correspond to any state in the CTMC). The dynamic subclass distribution in places PR and FDR is omitted, since it can be derived from that of other places: if CS is marked, PR and FDR are empty. If GS is marked, PR is empty and FDR has the same marking as ID. If both GS and CS are empty, FDR is empty and PR has the same marking as ID.

In this ESRG, eventualities are not represented until a class of markings is reached, where t_4 is enabled. At this point, eventualities may induce different behaviours, hence they must be considered separately. For instance, $\widehat{\widehat{m}}_{10}$ contains three explicit eventualities, while the six eventualities of $\widehat{\widehat{m}}_9$ are not represented.

The ESRG generation algorithm, explicitly represents eventualities only in two cases : if the ESM enables an asymmetrical transition or if some eventualities do not correspond to *reachable* SMs. In all the other cases, equivalent transition instances are enabled in every eventuality and the set of reachable eventualities is the set of possible instantiations of dynamic subclasses by static subclasses.

As we already mentioned, a possible consequence of asymmetry is that some eventualities of an ESM may not correspond to *reachable* SMs. An ESM $\widehat{\widehat{m}}_k$ is thus completely defined by a couple $\langle SR_k, E_k \rangle$, where SR_k stands for the ESM symbolic representation and E_k for the set of *reachable* eventualities of the ESM. A formal definition of ESMs is given in Appendix.

A specific reachable eventuality of $\widehat{\widehat{m}}_k$ is denoted $\langle SR_k, e_l \rangle$, or simply e_l, when the ESM is clearly identified by the context. With the same notation we will also indicate the corresponding SM, which is easily obtained by instantiating SR_k dynamic subclasses as defined by e_l.

ESMs whose eventualities all correspond to *reachable* SMs are called *saturated.* All the ESMs in Fig. 2 are saturated. Saturated ESMs with only one eventuality, e.g. $\widehat{\widehat{m}}_0$, are called *uniform.* ESMs that enable only symmetric transitions (all the ESMs in Fig. 2 but $\widehat{\widehat{m}}_{10}$ and $\widehat{\widehat{m}}_{12}$) are called *symmetric.*

When dealing with the ESRG, two levels of detail have to be handled: the level of the ESMs and the level of the eventualities. This also leads to different types of transition instances: we can have *generic* ones (denoted by thick arcs), which refer to the more abstract symbolic ESM representation, and the *instantiated* ones (denoted by thin arcs), which refer to the detail level of the eventualities. Both types of ESRG transition instances are performed using a symbolic firing rule directly derived from the one used for building the SRG. We postpone in Section 4.1 a more detailed description of ESRG firings.

The ESRG in Fig. 2 verifies one of the *sufficient* conditions for the ergodicity (of the underlying SRG) defined in [2], hence it can be used for steady state performance analysis.

We anticipate that, if all instances of transition t_4 (in the CS model) have the same firing rate, and the *strong lumpability* condition is considered (all the states in any aggregate must have the same output rates towards other aggregates), the obtained lumped CTMC is isomorphic to the ESRG (considering the ESMs as the nodes of the graph). If instead transition t_4 is asymmetrical also from a quantitative point of view, i.e. its firing instances have different rates, the states (SMs) in ESM $\widehat{\widehat{m}}_{10}$ do not have the same output rates, hence the stochastic behaviour of the system cannot be studied taking the ESM as an aggregate.

The technique that we present allows a lumped CTMC to be derived from an ergodic ESRG, which in the worst case coincides with the lumped CTMC isomorphic to the SRG. It is based on the analysis of the ESRG arcs to check the lumpability condition, considering ESMs as the initial aggregates of states (SMs). A splitting of the aggregates is carried out whenever the lumpability condition does not hold.

4 Definitions, Terminology and Notation

4.1 Basic definitions

Let us recall some basic definitions, concerning the construction of the SRG, and the isomorphic lumped CTMC, that are needed to understand the technique for deriving a lumped CTMC from the ESRG.

In a *symbolic firing instance*, dynamic subclasses are assigned to the transition variables instead of objects. The meaning is that any object in the dynamic subclass can be assigned to the variable. When several type Ci variables are instantiated in the same dynamic subclass Z_i^j (representing a set of objects arbitrarily chosen within the static subclass of Ci of index $d(Z_i^j)$), we also need to specify whether the variables are instantiated to the same object or to different objects of Z_i^j.

Let $Var(t) = \{x, y, \ldots\}$, denote the set of variables inscribing the arcs adjacent to t; we denote $\mathit{type}(x)$ the basic colour class (type) of x.

Definition 4.1 [Symbolic instance]
A symbolic instance of t in SM $\widehat{m}$, denoted $(t, \hat{c})$, is a function associating to each $x \in Var(t)$, where $\mathit{type}(x) = Ci$, a pair of integers $\langle j, k\rangle$, such that:

- $1 \le j \le nd_i$, where nd_i is the number of dynamic subclasses of Ci in $\widehat{m}$,
- $1 \le k \le |Z_i^j|$,
- $\forall 0 < l < k, \exists y \in Var(t) : \mathit{type}(y) = Ci$ and $(t, \hat{c})(y) = \langle j, l\rangle$.

More intuitively, the instance of $x \in Var(t)$ is specified by a dynamic subclass $Z_i^{j,k}$, meaning that the variable represents the k-th (arbitrarily chosen) element of Z_i^j in $\widehat{m}$.

The notion of symbolic instance does not change when considering the ESRG. The only difference is that it may apply to either one of two different levels: the eventuality level or the more abstract ESM (symbolic representation) level. In our examples, where one basic colour class C is used (C_k denotes the k-th static subclass of C), symbolic instances are denoted by pairs $(t, \langle Z_{C_n}^{j,k}, Z_{C_s}^{m,l}, \ldots\rangle)$, or $(t, \langle Z^{j,k}, Z^{m,l}, \ldots\rangle)$, depending on whether the relationship between dynamic and static subclasses is indicated or not (see also Sect. 4.2). The notation $(t, \hat{\hat{c}})$ will be also used to indicate an instance of the second type.

Indices $k, l \ldots$, denoting an element in a dynamic subclass, are omitted when the dynamic subclass cardinality is one. Indices $j, m \ldots$ are omitted if there is only one dynamic subclass.

The *cardinality of a symbolic instance*, denoted $|\widehat{m}\xrightarrow{(t,\hat{c})}|$, is the number of ordinary firings from each ordinary marking belonging to $\widehat{m}$ represented by the symbolic firing. Denoting with $\hat{Z}(\hat{c})$ the set of dynamic subclasses of $\widehat{m}$ which are assigned to some variable in $\hat{c}$, and with $inst_i^j$ (≥ 0) the number of *different* elements of dynamic subclass Z_i^j associated with some variable in $\hat{c}$:

$$|\widehat{m}\xrightarrow{(t,\hat{c})}| = \prod_{Z_i^j \in \hat{Z}(\hat{c})} \frac{|Z_i^j|!}{(|Z_i^j| - inst_i^j)!}$$

The cardinality of a symbolic instance enabled in a symbolic ESM representation SR, denoted $|SR\xrightarrow{(t,\hat{c})}|$, is computed in the same way, considering the dynamic subclasses appearing in SR.

Let $\mathbf{w}[t](c)(m)$ denote the SWN *rate/weight function*. The symmetry property of SWNs implies that $\forall m_1, m_2 \in \widehat{m}$, $\forall$ instances $(t, c_1), (t, c_2)$ represented by the same symbolic instance $(t, \hat{c})$: $\mathbf{w}[t](c_1)(m_1) = \mathbf{w}[t](c_2)(m_2)$. Hence, the function can be extended to symbolic instances in a straightforward way:

$\mathbf{w}[t](\hat{c})(\widehat{m}) = \mathbf{w}[t](c)(m_k)$, where $m_k \in \widehat{m}$ and (t, c) is any instance represented by $(t, \hat{c})$.

If we assume that the rate/weight function only depends on the assignment of transition variables to static subclasses, we can use the notation $\mathbf{w}[t](static(\hat{c}))$, where $static(\hat{c})$ defines the static subclasses to which dynamic subclasses involved in the firing belong.

The transition rate of symbolic instance $(t, \hat{c})$ (in $\widehat{m}$) is thus defined by:

Definition 4.2 [Rate of a Symbolic Instance]

$$\mu(t, \hat{c}) = |\widehat{m}\xrightarrow{(t,\hat{c})}| \ \ \mathbf{w}[t](static(\hat{c})).$$

Let us now recall the strong lumpability condition.

Definition 4.3 Let $\{A_1, \ldots, A_n\}$ be a partition of the state space of a CTMC M. The strong lumpability condition holds for any A_i of A iff

$$\forall A_j, j \neq i, \forall s_k, s_m \in A_i : rate[s_k, A_j] = rate[s_m, A_j]$$

where $rate[s_k, A_j] = \sum_{s_l \in A_j} rate[s_k, s_l]$.

If the strong lumpability condition holds for all A_i of A, then a CTMC M' can be obtained from M replacing each set of states $s_k \in A_i$ with a single state (aggregate), denoted A_i, and setting $\forall A_j, j \neq i : rate[A_i, A_j] = rate[s_k, A_j]$. Then, denoting with π and π' the stationary probability functions of M and M', the following property holds:

$$\sum_{s_k \in A_i} \pi(s_k) = \pi'(A_i)$$

It has been proven [3] that the strong lumpability condition holds for the CTMC isomorphic to the RG of any SWN, with respect to the aggregation of ordinary markings into symbolic markings. The output rate from $\widehat{m}_i$ to

$\widehat{m}_j$ (the element $[i,j]$ of the infinitesimal generator Q, of size $|SRG|$), may be directly computed on the SRG:

$$rate[\widehat{m}_i, \widehat{m}_j] = \sum_{(t,\hat{c}):\, \widehat{m}_i \xrightarrow{(t,\hat{c})} \widehat{m}_j} \mu(t, \hat{c}).$$

4.2 ESRG arcs classification

An important concept in the ESRG is the distinction between asymmetric and symmetric firings. It is possible to *syntactically* recognize the *symmetric* transitions, i.e. those transitions whose enabling condition and firing rate does not depend on static subclasses (all transitions of SWN in Fig.1, but t_4) from *asymmetric* transitions (e.g. t_4 in the same SWN).

Definition 4.4 [Symmetric transition]
t is symmetric if:

a) no static subclass, and no clause of type $d(x) = d(y)$ or its negation appears in t inscriptions or predicate (structural condition).

b) $\forall \widehat{\widehat{m}} = \langle SR, E\rangle, \forall (t, \hat{c}_1), (t, \hat{c}_2)$ represented by $(t, \hat{\hat{c}})$ in SR:

$$\mathbf{w}[t](\hat{c}_1) = \mathbf{w}[t](\hat{c}_2) \quad \text{(firing rate condition).}$$

If t is symmetric, the rate/weight of a symbolic instance, $\mathbf{w}[t](\hat{\hat{c}})$, is set equal to $\mathbf{w}[t](\hat{c}_1)$, where $(t, \hat{c}_1)$ is any instance of $(t, \hat{\hat{c}})$.

In the ESRG there are two types of arcs (transition firings), that are classified as follows:

generic arc : it corresponds to the firing of a symbolic instance of a *symmetric* transition from the symbolic representation SR_i of a *saturated* ESM, to the symbolic representation SR_j of an ESM (which as a consequence is also *saturated* [2]).

instantiated arc : it corresponds to the firing of a symbolic instance of a transition from an eventuality of the source ESM. Very often instantiated arcs correspond to asymmetric transition firings, however if the source ESM is *not saturated*, the transition may be symmetric. The term *instantiated* means that the symbolic instance takes into consideration the partitions of classes into static subclasses. The destination may be the symbolic representation of a saturated symmetric ESM, or an eventuality.

For instance, in the ESRG of Fig. 2, the arcs corresponding to the firings of asymmetric transition $t4$ are instantiated. All remaining arcs are generic.

4.3 Properties of the ESRG and lumpability

We state some properties of the ESRG which will be employed in different steps of the algorithm we are going to present, which allows a lumped CTMC to be derived from an ergodic ESRG. It is based on extracting information from the ESRG arcs, which is exploited to check the lumpability condition, considering ESMs as the initial aggregates of states (SMs).

Property 1 *The strong lumpability condition holds for all the ESMs with only one eventuality (called elementary).*

The next property relates the cardinality of the ESRG generic arcs to the cardinality of the SRG arcs they represent.

Property 2 *Let t be a symmetric transition. Let $\widehat{\widehat{m}}_1 = \langle SR_1, E_1 \rangle$, $\widehat{\widehat{m}}_2 = \langle SR_2, E_2 \rangle$. Then, $\forall e_k \in E_1$:*
$\sum_{\hat{c}, e_m \colon \langle SR_1, e_k \rangle \xrightarrow{(t,\hat{c})} \langle SR_2, e_m \rangle} |\langle SR_1, e_k \rangle \xrightarrow{(t,\hat{c})}| = \sum_{\hat{\hat{c}} \colon SR_1 \xrightarrow{(t,\hat{\hat{c}})} SR_2} |SR_1 \xrightarrow{(t,\hat{\hat{c}})}|.$
where the symbolic instances $(t, \hat{c})$ appearing in the left summation are those represented by the generic transition instance(s) $(t, \hat{\hat{c}})$ appearing in the summation on the right.

A first direct consequence of Property 2 is that the lumpability condition is *initially* satisfied by symmetric ESMs since only symmetric firings (either generic or instantiated) depart from them: in fact each eventuality of a symmetric ESM reaches the destination ESM with the same rate, which is equal to the sum of rates of the generic firings connecting the two ESMs. A second direct consequence is that the rates of generic firings may be computed without the need of developing all the SRG arcs represented by them.

A similar property does not hold instead for non symmetric ESMs. For this reason an algorithm is needed that *refines* the partition into aggregates, by splitting those aggregates that do not satisfy the lumpability condition. This may start a *domino effect* of aggregates splitting. The following property will be used during the refinement process.

Property 3 *Let $ag_1 \longrightarrow ag_2$ satisfy the lumpability condition. Then each arc resulting from a splitting of source ag_1 still satisfies the condition.*

In the last properties, we have used the term *lumpability condition* with respect to single (symmetric) arcs: this forces a little bit the meaning of lumpability condition, however the justification is that in our algorithm we shall check the global lumpability by checking the condition locally for each pair $\langle ag_i, ag_j \rangle$ of connected aggregates, moreover for a given pair of aggregates we shall perform the local check avoiding to analyse those symmetric arcs that surely cannot cause an unbalance in the rate from the elements of the source aggregate and the destination aggregate. Note that instead if $ag_1 \longrightarrow ag_2$ and ag_2 gets split into k smaller aggregates $ag_{2,1}, \ldots, ag_{2,k}$, the lumpability condition may not hold any more for the k pairs $ag_1 - ag_{2,j}$.

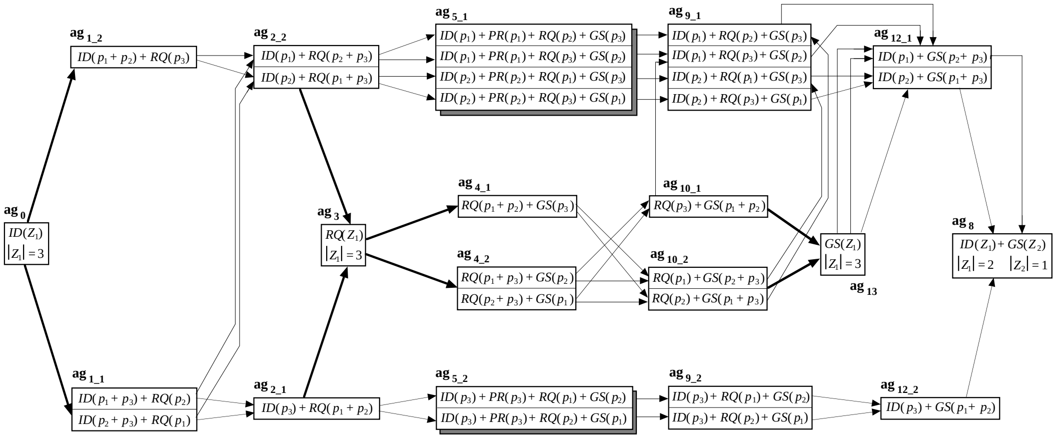

$\mathbf{ag}_{0}$
$ID(Z_1)$
$|Z_1| = 3$
$\mathbf{ag}_{1\_2}$
$ID(p_1 + p_2) + RQ(p_3)$
$\mathbf{ag}_{1\_1}$
$ID(p_1 + p_3) + RQ(p_2)$
$ID(p_2 + p_3) + RQ(p_1)$
$\mathbf{ag}_{2\_2}$
$ID(p_1) + RQ(p_2 + p_3)$
$ID(p_2) + RQ(p_1 + p_3)$
$\mathbf{ag}_{2\_1}$
$ID(p_3) + RQ(p_1 + p_2)$
$\mathbf{ag}_{3}$
$RQ(Z_1)$
$|Z_1| = 3$
$\mathbf{ag}_{5\_1}$
$ID(p_1) + PR(p_1) + RQ(p_2) + GS(p_3)$
$ID(p_1) + PR(p_1) + RQ(p_3) + GS(p_2)$
$ID(p_2) + PR(p_2) + RQ(p_1) + GS(p_3)$
$ID(p_2) + PR(p_2) + RQ(p_3) + GS(p_1)$
$\mathbf{ag}_{4\_1}$
$RQ(p_1 + p_2) + GS(p_3)$
$\mathbf{ag}_{4\_2}$
$RQ(p_1 + p_3) + GS(p_2)$
$RQ(p_2 + p_3) + GS(p_1)$
$\mathbf{ag}_{5\_2}$
$ID(p_3) + PR(p_3) + RQ(p_1) + GS(p_2)$
$ID(p_3) + PR(p_3) + RQ(p_2) + GS(p_1)$
$\mathbf{ag}_{9\_1}$
$ID(p_1) + RQ(p_2) + GS(p_3)$
$ID(p_1) + RQ(p_3) + GS(p_2)$
$ID(p_2) + RQ(p_1) + GS(p_3)$
$ID(p_2) + RQ(p_3) + GS(p_1)$
$\mathbf{ag}_{10\_1}$
$RQ(p_3) + GS(p_1 + p_2)$
$\mathbf{ag}_{10\_2}$
$RQ(p_1) + GS(p_2 + p_3)$
$RQ(p_2) + GS(p_1 + p_3)$
$\mathbf{ag}_{9\_2}$
$ID(p_3) + RQ(p_1) + GS(p_2)$
$ID(p_3) + RQ(p_2) + GS(p_1)$
$GS(Z_1)$
$|Z_1| = 3$
$\mathbf{ag}_{13}$
$\mathbf{ag}_{12\_1}$
$ID(p_1) + GS(p_2 + p_3)$
$ID(p_2) + GS(p_1 + p_3)$
$\mathbf{ag}_{8}$
$ID(Z_1) + GS(Z_2)$
$|Z_1| = 2 \quad |Z_2| = 1$
$\mathbf{ag}_{12\_2}$
$ID(p_3) + GS(p_1 + p_2)$

Fig. 3. Aggregates modified by the application of the lumpability algorithm

5 The algorithm for lumpability check

In this section we first sketch the algorithm for lumpability check, then we state it more formally.

The input to the algorithm is an initial partition of SMs into aggregates corresponding to ESMs. The goal of the algorithm is to check whether the aggregates satisfy the lumpability condition, and in case the condition is not satisfied, partition the aggregates in smaller ones that satisfy the condition.

The part of the ESRG of the DCS example that is modified by the algorithm is illustrated in Fig. 3: this refinement of aggregates refers to the case of a static subclass dependent rate for transition t_4 (which may take one of two possible values, qA_4 or qB_4). For the sake of clarity, in Fig. 3 the symbolic representations SR of ESMs are not reported (only aggregations of eventualities are highlighted). The meaning of the notation is as follows : ag_i is the aggregate corresponding to ESM $\hat{\hat{m}}_i$, ag_{i_j} is the j^{th} subaggregate of ESM $\hat{\hat{m}}_i$ in case it gets split.

The task of checking lumpability and of refining the aggregates so that the condition holds has been already dealt with in the literature (see for example [10,9]). The novelty of the proposed algorithm is the fact that it works using the information contained in the ESRG, so that the CTMC to be checked for lumpability is not explicitly given.

The *objects* manipulated by the algorithm are mainly the aggregates, but we also use the ESRG information and structure. In particular, the ESM eventualities correspond to the SMs which belong to an aggregate. A delicate point is that the arcs between aggregates are never explicitly represented, instead they are derived from the ESRG structure. When an ESM contains the explicit representation of its eventualities, and all the arcs departing from it are instantiated, the arcs connecting aggregates can be easily derived from the instantiated arcs departing from eventualities. On the other hand, a generic arc between two ESMs which have been refined by the algorithm, can represent a set of arcs connecting the subaggregates of the source ESM to *some* subaggregates (not necessarily all) of the destination ESM: as we shall see, when this happens the generic arcs of the ESRG are *unfolded* into the instantiated arcs they represent, to allow a correct information on the connection between aggregates. A simpler case arises when the generic arc connects two ESMs and only the source ESM has been refined into subaggregates. In this situation, each subaggregate of the source ESM surely reaches the destination ESM, moreover the lumpability condition surely holds with respect to this arc, whatever the partition of the source ESM is (see Properties 2 and 3), hence this arc does not need to be unfolded.

During the execution of the algorithm, the connections between aggregates will be classified as *checked* or *to-be-checked* with respect to a local checking of the lumpability condition. In particular, set TBC denotes the pairs $\langle ag, ag' \rangle)$ of *to-be-checked* connections.

The algorithm initialization step marks as *to-be-checked* all the connections for which the lumpability condition does not trivially hold, i.e., the pairs $\langle ag, ag' \rangle$ connected by an asymmetric connection (this happens when the ESM corresponding to ag and ag' are linked by an asymmetric firing). An exception to this rule arises when ag contains only one eventuality, since a source ESM with one eventuality trivially satisfies the lumpability condition. For instance, if we consider the ESRG of our DCS example, the initialization step yields $TBC = \{\langle ag_{12}, ag_8 \rangle, \langle ag_{10}, ag_9 \rangle\}$ (a direct correspondence to set $\{\ \langle \widehat{\widehat{m}}_{12}, \widehat{\widehat{m}}_8 \rangle, \langle \widehat{\widehat{m}}_{10}, \widehat{\widehat{m}}_9 \rangle\ \}$). Observe that the connection $\langle ag_{13}, ag_{12} \rangle$ is not *to-be-checked* even if $\widehat{\widehat{m}}_{13}$ and $\widehat{\widehat{m}}_{12}$ are connected through asymmetric arcs (firing of t_4), because ag_{13} is uniform (it has only one eventuality). Moreover, the connection $\langle ag_{10}, ag_{13} \rangle$ is not in TBC, because it corresponds to a generic arc between $\widehat{\widehat{m}}_{10}$ and $\widehat{\widehat{m}}_{13}$.

If the lumpability condition is discovered not to hold for some pair $\langle ag, ag' \rangle$ in TBC, then ag must be refined (i.e., split) into *finer grain* aggregates. The splitting of an aggregate ag may break the lumpability condition for the predecessors of ag (i.e., for the aggregates that reach ag). As a consequence the algorithm must add into TBC a pair $\langle ag', ag \rangle$ for each predecessor ag' of ag. On the other hand, after the splitting all the connections $\langle ag, . \rangle$ must be removed from TBC.

Turning to our example, when the lumpability check is performed for aggregate ag_{12}, comprising the three eventualities of $\widehat{\widehat{m}}_{12}$ each one enabling an instance of t_4, if t_4 has a static subclass independent rate then the lumpability condition is satisfied and there is no need to split ag_{12}. If instead t_4 has a static subclass dependent rate, and in particular if we assume the following firing rates, qA_4 for the instance departing from eventuality $e3$ and qB_4 for the other two instances, then, ag_{12} must be split into subaggregates ag_{12_1} and ag_{12_2} (see Fig. 3). This splitting may invalidate the lumpability condition for the predecessors of ag_{12}, hence the pairs $\langle ag_9, ag_{12\_1} \rangle, \langle ag_9, ag_{12\_2} \rangle$ are added into TBC. After the above operations, $\langle ag_{12}, ag_8 \rangle$ is removed from TBC.

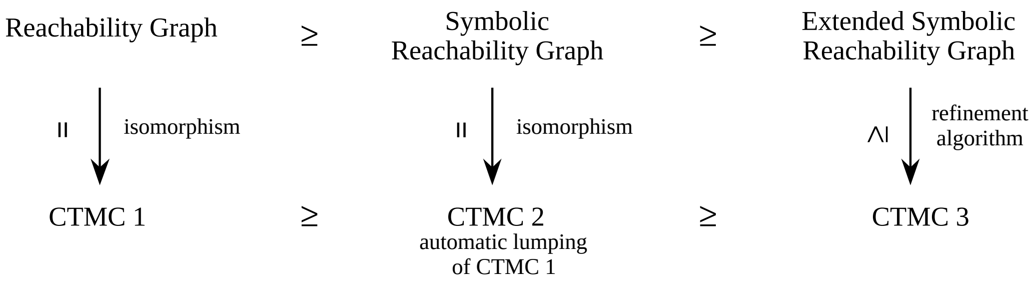

Reachability Graph
≥
Symbolic Reachability Graph
≥
Extended Symbolic Reachability Graph
=
isomorphism
=
isomorphism
≥
refinement algorithm
CTMC 1
≥
CTMC 2
automatic lumping of CTMC 1
≥
CTMC 3

Fig. 4. Comparison of the CTMCs obtained from a SWN model.

The procedure described above must be reiterated until set TBC becomes empty. For the sake of efficiency, all the pairs which relate to the same source aggregate are analysed together. This is the case for example for pairs $\langle ag_9, ag_{12\_1}\rangle$ and $\langle ag_9, ag_{12\_2}\rangle$, added to TBC upon splitting of ag_{12}. Observe that the lumpability check from ag_9 ($\widehat{\widehat{m}}_9$) requires not only to develop $\widehat{\widehat{m}}_9$ in eventualities but also to *unfold* the generic arc $\widehat{\widehat{m}}_9 \xrightarrow{\langle t_2, Z^2\rangle} \widehat{\widehat{m}}_{12}$ into the instantiated arcs connecting the eventualities of $\widehat{\widehat{m}}_9$ to the eventualities of $\widehat{\widehat{m}}_{12}$. After the unfolding, it becomes apparent that ag_9 must be split in two subaggregates ag_{9_1} and ag_{9_2}. There are also ESMs that are not split by the algorithm: this is the case for aggregates ag_{13}, ag_3 and ag_0 which all correspond to uniform ESMs, and for aggregate ag_{11} ($\widehat{\widehat{m}}_{11}$) since it reaches only ag_0 through a generic arc. Furthermore, the algorithm does not always require the unfolding of generic arcs. For improving the readability of Fig. 3, generic arcs connecting split aggregates, are explicitly represented, while the algorithm derives them from the ESRG arcs (e.g. see the generic arcs exiting from aggregate ag_3 ($\widehat{\widehat{m}}_3$)).

Let us discuss the relation between the sizes of the CTMCs that can be obtained from a SWN model, summarized in Fig. 4. The CTMC obtained from the SRG has the same size as the SRG and is actually an automatic lumping of the CTMC obtained from the RG. It may happen that $CTMC2$ is identical to $CTMC1$, i.e., $CTMC1$ is not lumpable, and this is exactly what happens when all the basic colour classes are split into static subclasses of cardinality one: this condition however can be automatically detected from the structure of the model. In general it is not possible to derive a CTMC isomorphic to the ESRG. The refinement algorithm proposed in this paper, allows to automatically derive a Markov chain $CTMC3$ from the ESRG, which is a lumping of $CTMC2$ and which in general will have a number of states greater than the number of ESMs in the ESRG. The alternative to the proposed approach would be to blindly apply a partitioning algorithm to $CTMC2$ (or even worse, to $CTMC1$), but this approach in general would be less efficient because it would not take into account the information on the potential for aggregation contained into the ESRG (and the SRG), moreover it would require to first build completely $CTMC2$ (or even worse $CTMC1$), while we directly build the lumped Markov chain $CTMC3$.

Let us compare the sizes of the SRG, of the ESRG and of the lumped CTMC of our running example in two cases: in the first case transition t_4 has a rate which does not depend on the colour, while in the second it has a static subclass dependent rate.

The SRG (which in this case is of the same size as the RG) has 30 tangible and 15 vanishing SMs, the ESRG has 11 tangible and 3 vanishing ESMs. In the less favourable case, the final lumped CTMC is reduced by a factor 2 with respect to the SRG size, thanks to our technique. The reduction grows to a factor 3 when the rate of t_4 does not depend on the static subclasses: in fact in this case all the ESMs satisfy the lumpability condition (observe that in any case the reduction due to the ESRG algorithm cannot be more than $|Proc|! = 6$). In general, the reduction factor is hard to estimate, as it depends both on the degree of asymmetry of the net and the structure of the graph.

5.1 Formal description of the algorithm

In this section, we formally express how the aggregates are managed by the algorithm. A technical aspect which deserves some explanation concerns the management of aggregate representation: this is not trivial since aggregates are created and split dynamically; as a consequence the grouping of eventualities into aggregates must be systematically updated in such a way that it is always easy to retrieve the association between an aggregate identifier ag and the eventualities it comprises, even if ag has been refined into finer grain subaggregates at some point. In order to solve this problem, the algorithm uses two structures, besides the input data structure representing the ESRG:

- $AGGREG$ is a set of tree structures, comprising one tree for each ESM of the ESRG; it is needed to represent all the aggregates generated by the algorithm, and the relation between (*subaggregates of*) aggregates, due to the splitting of aggregates performed during the algorithm. For the sake of simplicity in the presentation, each aggregate has a unique identifier within this structure.
- TBC is a set of pairs of aggregate identifiers (from $AGGREG$). As anticipated in the informal presentation, each pair $\langle ag, ag_{dest} \rangle$ in TBC indicates that aggregate ag should be checked for lumpability, and in particular the pairs $\langle ag, - \rangle$ in TBC indicate which connections departing from ag may not satisfy the condition.

Let us now describe in detail these structures and the functions handling them. In $AGGREG$ there is one tree for each ESM $\widehat{\widehat{m}}$: the leaves of these tree represent the current partition in subaggregates of $\widehat{\widehat{m}}$ (each leaf corresponds to a subset of eventualities of $\widehat{\widehat{m}}$). Each branching node in the tree instead represents a subaggregate of $\widehat{\widehat{m}}$, created at some step of the algorithm: its successor nodes are the subaggregates that were directly created from it by the algorithm, which may in turn have been split at successive steps. To deal with the hierarchical structure of $AGGREG$, the *subAg(ag)* function is introduced: given an aggregate identifier ag, this function returns the set of aggregate identifiers corresponding to the leaves of the subtree in $AGGREG$, whose root is ag. The splitting of a leaf node is performed by function *splitAg(ag, AG)*: parameter AG is the new partition of the eventualities of the leaf node identified by ag. The *splitAg(ag, AG)* function modifies the $AGGREG$ structure so that node ag becomes a branching node and its immediate successors (new leaves) are created as required by the splitting operation represented by AG. Moreover, it returns the identifiers of the newly created nodes.

Finally function *createAGGREG*($\widehat{\widehat{m}}$) is used to initialize $AGGREG$ as follows: for each ESM $\widehat{\widehat{m}}$, a tree is created made of a single (root-leaf) node corresponding to all the reachable eventualities of $\widehat{\widehat{m}}$.

In order to keep track of the ESRG information and structure during the algorithm, we introduce the following notations, which refer to a given $\widehat{\widehat{m}} = \langle SR, E \rangle$ of the ESRG and to a given node ag in the tree of $AGGREG$ associated with $\widehat{\widehat{m}}$: $\widehat{\widehat{m}}.AG$ is the set of nodes of $AGGREG$ associated with $\widehat{\widehat{m}}$. Moreover, $ag.ESM$ denotes the symbolic representation SR of $\widehat{\widehat{m}}$ and $ag.EV$ denotes the subset of eventualities of $\widehat{\widehat{m}}$ corresponding to ag.

We now define the connections between two nodes ag and ag_d of $AGGREG$. $\exists ag \rightarrow ag_d$ iff one of the following conditions holds true :

- $\exists e \in ag.EV, e' \in ag_d.EV, \ e \rightarrow e'$
- $\exists e \in ag.EV, \ e \rightarrow ag_d.ESM$ (instantiated arcs)
- $\exists ag.ESM \rightarrow ag_d.ESM$ (generic arc)

At this point, it is worth noticing that when ESMs are refined according to the lumpability condition, the ESRG structure is updated: the eventualities of the split ESMs are developed and the generic arcs departing from the ESM are unfolded into the corresponding instantiated ones if needed. The labels attached to the arcs linking aggregates are inherited from the transition relation between ESMs, in particular $ag \xrightarrow{(t,\hat{\hat{c}})} ag_d$ ($ag \xrightarrow{(t,\hat{c})} ag_d$) means that one arc from aggregate ag to ag_d is due to the firing of the generic (instantiated) transition firing $(t, \hat{\hat{c}})((t, \hat{c}))$. We shall also use the notation $ag \xrightarrow{(a)} ag_d$ meaning that (some eventuality in) ag is connected to (some eventuality in) ag_d through an instantiated asymmetric arc. Finally, $\langle ., ag \rangle$ denotes the set of pairs of connected aggregates with destination ag.

Set TBC is the key structure of the algorithm since it is used to keep track of the elements to be checked for lumpability. Any pair $\langle ag, ag_{dest} \rangle$ in TBC means that aggregate ag may be not lumpable because of its connections $ag \rightarrow ag_{dest}$. Using the information contained in the ESRG, one can derive the arcs $ag.ESM \rightarrow ag_{dest}.ESM$ which may be a problem with respect to the lumpability condition, namely: (1) all instantiated asymmetric arcs from $ag.EV$ to $ag_{dest}.EV$ are *to be checked*; (2) all symmetric arcs from $ag.EV$

to $ag_{dest}.EV$ (instantiated) and from $ag.SR$ to $ag_{dest}.SR$ (generic) are *to be checked* whenever the ESM $ag_{dest}.ESM$ is split ($\mid ag_{dest}.ESM.AG \mid > 1$).

Set TBC is initialized with pairs of ESMs connected through asymmetric firings. The following set called TBC_{asym} captures both qualitative and quantitative aspects of asymmetries, in the initial partition into aggregates:

$$TBC_{asym} = \{\langle ag, ag_{dest}\rangle \mid \exists ag \xrightarrow{(a)} ag_{dest}\} \cup$$
$$\{\langle ag, ag_{dest}\rangle \mid \exists ag \xrightarrow{t} ag_{dest} \text{ s.t. } w[t] \text{ depends on st.subcl. }\}.$$

For any element $tbc = \langle ag, ag_d\rangle$ of TBC, we introduce the following notations: $tbc.src = ag$ and $tbc.dest = ag_d$. Moreover, $\langle ag, -\rangle \in TBC$ denotes the subsets of pairs of TBC, the first component of which is ag. The algorithm which derives a lumpable structure from the ESRG is the following:

Algorithm 1 (Computation of a lumpable structure)

```
foreach $\widehat{\widehat{m}} \in ESRG$ do createAGGREG($\widehat{\widehat{m}}$);
$TBC = TBC_{Asym} \setminus \{\langle ag, ag_{dest}\rangle$ s.t. $\mid ag.EV \mid = 1\}$;
while(($tbc = TBCchoose$ ()) != nil) do
    begin
    $TBCC = \{ag_d \mid \langle ag, ag'\rangle \in TBC \wedge ag_d \in subAg(ag')\}$;
    $AG := localLump(tbc.src, TBCC)$;
    if ($\mid AG \mid > 1$) then
        begin
        $AGR := splitAg(tbc.src, AG)$;
        $TBC = TBC \backslash \langle tbc.src, -\rangle$; $TBC = TBC \cup \{\langle ., ag\rangle, ag \in AGR\}$;
        end
    end
endalgorithm
```

So, after the initialization of $AGGREG$ and TBC, the algorithm iterates, selecting at each step a pair *tbc* of TBC by means of function *TBCchoose()*.

For the subset $\langle tbc.src, -\rangle$ found in TBC, the algorithm computes the set of TBC destination leaf nodes, namely $TBCC$. At this point, one may observe that, although a pair $\langle ag, ag_{dest}\rangle$ concerns leaf nodes when it is added to TBC, it may be possible that ag_{dest} is split further later, hence ag_{dest} could represent a branching node of $AGGREG$ (this is why we need to use function $subAg()$ when building set $TBCC$) . In contrast, *tbc.src* is always a reference to a "leaf node" since the algorithm deals with all pairs of $\langle tbc.src, -\rangle$ simultaneously, checks *tbc.src* for lumpability, then removes $\langle tbc.src, -\rangle$ globally.

Then, the algorithm computes a partition AG of eventualities from *tbc.src* by a call to function *localLump(tbc.src,TBCC)*. This function analyzes the lumpability condition, locally to the aggregate named *tbc.src* and with respect to all destination aggregates contained in $TBCC$. It starts by developing the eventualities of the ESM corresponding to *tbc.src* (if needed). Also some generic arcs departing from this ESM may need to be unfolded into the corresponding instantiated arcs: they correspond to those reaching the ESMs $ag.ESM$, where $ag \in TBCC$. After the instantiation (which actually modifies the ESRG structure), a matrix of rates is computed whose rows correspond to the eventualities ev_j of the aggregate named *tbc.src*, and whose columns correspond to all aggregates referred in $TBCC$. Then $rate[ev_j, ag_i]$ is generated: the elements of this matrix are computed by simply applying the rate computation used for deriving a CTMC from the SRG. Actually, the eventualities are symbolic markings, and the arcs we are considering correspond to instantiated firings. Observe that the computation of matrix rate does not involve symmetric arcs such that the destination is not split since they contribute with equal rate values for all the source aggregates. After the *rate* matrix has been computed, if all its rows are equal, then the aggregate named *tbc.src* satisfies the lumpability condition and does not need further splitting (it is returned as a singleton of set AG). If instead not all the rows of *rate* are equal, then we must partition the eventualities referred by *tbc.src* in new (sub)aggregates, corresponding to the equivalence classes of eventualities obtained by using the following equivalence relation: $ev_i \sim ev_j$ iff $rate[ev_i, .] = rate[ev_j, .]$. In other words, the eventualities corresponding to equal rows in *rate* are grouped together in a new (sub)aggregate.

The correctness of the algorithm is strongly related to the fact that at each step, set $TBCC$ represents all *to-be-checked* destination aggregates w.r.t. some aggregate. This is initially true due to the initialization of TBC from (a convenient subset of) TBC_{asym}, then at each step of the algorithm TBC is updated, (1) by removing all the elements of $\langle ag, - \rangle$, and (2) by adding all the aggregates of $\langle ., ag \rangle$ if ag is split. The termination is ensured by the fact that the number of splittings is bounded by the number of eventualities of the ESM.

The efficiency of the algorithm depends on the order in which aggregates are considered: function $TBCchoose()$ is responsible for this. We want to avoid reconsidering the same aggregate over and over again due to successive splitting of the aggregates it reaches. Some heuristics can help in this sense, for example the aggregates enabling some asymmetric transitions should be considered first since they most probably will be split. Moreover, if ag_1 reaches ag_2 and the two aggregates are both *to-be-checked* (i.e., TBC contains both some pairs $\langle ag_1, . \rangle$ and some pairs $\langle ag_2, . \rangle$), then ag_2 should be considered first.

6 Generation of the Markov chain

In this section we define the rules for the lumped Markov chain generation given the (possibly refined) ESRG and the partition into aggregates computed by the algorithm presented in the previous section.

- **Rule a:** the arcs to which this rule applies correspond to instantiated firings departing from eventualities, i.e., from SMs. Therefore, the rate computation rule recalled in Definition 4.2 can be used again.

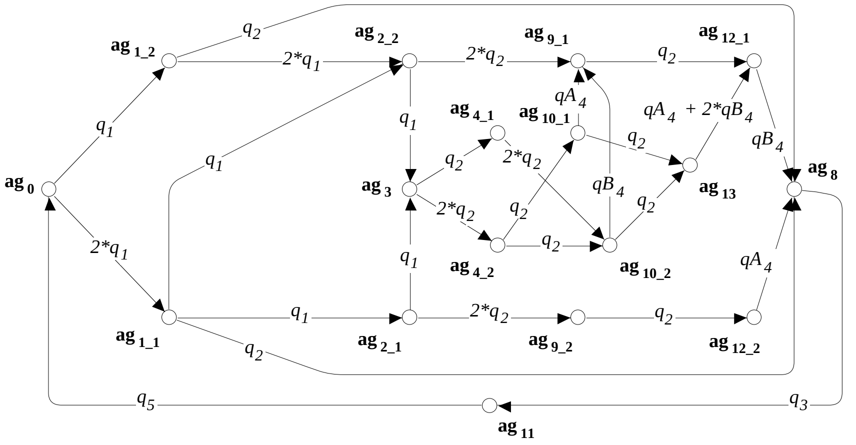


Fig. 5. complete CTMC of the DCS example

- **Rule b1:** the arcs to which this rule applies correspond to generic symmetric arcs departing from aggregates and reaching non split (saturated) ESMs. Again the usual rate computation rule can be applied taking care of the fact that the symbolic firing refers to the dynamic subclasses defined in the symbolic representation of the ESM. This is why we need to keep track of this representation also for the split subaggregates. Observe also that in this case the transition rate cannot depend on the static subclasses of the objects involved in the firing.
- **Rule b2:** the last rule applies to generic arcs connecting a uniform (source) ESM ag_u and the subaggregates ag_{d_j} of a split aggregate ag_d. Notice that the split aggregate surely originated from a saturated ESM (because saturation propagates through generic symmetric firings). Let $n1$ be the cardinality (number of eventualities) of ag_d, the ESM from which a subaggregate ag_{d_j}, originated, and let $n2$ be the cardinality (number of eventualities) of ag_{d_j}. The rate from ag_u to ag_{d_j} is obtained by computing the rate corresponding to the generic symmetric firing (using the usual rule of rate computation for symbolic firings) and multiplying it by the factor $\frac{n2}{n1}$.

It is easy to show that the above three rules cover all possible types of arcs connecting aggregates.

Fig. 5 shows the complete CTMC derived for our running example in the hypothesis that t_4 has a colour dependent rate (as in the examples of Section 5). In this specific example, the eventualities correspond to ordinary markings, due to the fact that the static subclasses of class C have all cardinality one, so that the instantiated firings boil down to ordinary firings and their rate computation is trivial. The symbolic firing rate computation rule (case $b1$) can instead be applied to all the generic symmetric firings: e.g., the arc connecting $\widehat{\widehat{m}}_8$ to $\widehat{\widehat{m}}_{11}$ corresponds to firing the symbolic transition instance (t_3, Z_2). Since in $\widehat{\widehat{m}}_8$ the cardinality of Z_2 is 1, the rate of this transition instance is the (static subclass independent) rate q_3 of transition t_3. Another example of application of this rule is the computation of the rates from ag_{10_i}

to ag_{13} $(= \widehat{\widehat{m}}_{13})$: observe that both arcs have necessarily the same rate given by the (static subclass independent) rate q_2 of t_2 multiplied by the cardinality of the symbolic arc that in this case is 1 because the unique dynamic subclass involved (Z_1) has cardinality 1.

Let us consider now the rate computation rule $b2$ that appears two times in our running example. The two generic arcs connecting $\widehat{\widehat{m}}_0$ to ag_{1_i} have a global rate of $3q_1$ where q_1 is the (static subclass independent) rate of t_1 while the factor 3 is computed as: $\frac{|Z_1|!}{(|Z_1|-1)}$ (in $\widehat{\widehat{m}}_0$, $|Z_1| = 3$). Now we have to compute which portion of this rate is directed towards each ag_{1_i}: this is directly related with the cardinality of the two aggregates, so that $\frac{2}{3}$ of it goes towards ag_{1_1} and $\frac{1}{3}$ of it goes towards ag_{1_2}.

7 Conclusions

In this paper, we have introduced a general method for deriving a lumped Markov chain from the SWN representation of a partially symmetrical system. The advantage of using SWN as a modelling formalism is that partial symmetries are obtained from the structure of the model, thus avoiding to compute them a-posteriori, and they can be automatically exploited during the state space construction. The algorithm that we propose exploits the ESRG both for deriving the structure of the Markov chain and for optimizing the computation of rates between states. In some cases, the size of the lumped Markov chain is close to the one obtained from a totally symmetrical model, despite the asymmetries.

A possible alternative to the strong lumpability condition that has been presented in this paper would be to look for other lumpability conditions, as for example the *exact lumpability*[1]. Choosing which one should be applied mainly depends on the performance criteria that must be computed. If probabilities of individual (symbolic) markings are needed, then strong lumpability cannot be used as it only gives the probabilities of aggregates. If instead performance criteria can be expressed at the level of aggregates, then both approaches can be used. Unfortunately, we do not have yet a criteria for deciding which method will give the smallest CTMC for a given SWN model.

The fact that for some systems the ESRG yields a too abstract view of the state space (which implies that the refinement algorithm has to go through several iterations to generate the final CTMC) leads us now to work on the state space construction algorithm. Our future work aims at finding a unifying framework for different approaches which deal with partial symmetries, among which the recent works of [6] and [7], without forgetting the need to extend the associated performance evaluation techniques accordingly.

References

[1] Buchholz, P., *Exact and ordinary lumpability in finite Markov chains*, Journal of Appl. Prob., **31**(1994), 59–75.

[2] Capra, L., C. Dutheillet, G. Franceschinis and J.M. Ilie, *Towards Performance Analysis with Partially Symmetrical SWN*, In Proc. 7th International Symposium on Modeling, Analysis and Simulation, College Park, MD, USA, October 1999.

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

[4] Chiola, G., C. Dutheillet, G. Franceschinis, and S. Haddad, *A Symbolic Reachability Graph for Coloured Petri Nets*, Theoretical Computer Science B (Logic, semantics and theory of programming), **176, n. 1&2**(1997), 39–65.

[5] Chiola, G., G. Franceschinis, R. Gaeta, and M. Ribaudo, *GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets*, Performance Evaluation, special issue on Performance Model ing Tools, **24/1&2**(1995), 47–68.

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

[7] Haddad, S., J-M. Ilie, and K. Ajami, *A Model Checking Method for Partially Symmetric Systems*, In Proc. of of FORTE XIII, Pisa, Italy, October 2000.

[8] Haddad, S., J-M. Ilie, M. Taghelit, and B. Zouari, *Symbolic marking graph and partial symmetries*, In Proc. of 16th Int. Conference on Application and Theory of Petri Nets, ICATPN '95, Torino, Italy, June 1995, 238–257.

[9] Hermanns, H., "Interactive Markov Chains", PhD Thesis, Erlangen-Nurnberg Friedrich-Alexander University, 1999.

[10] Paige, P., and R. Tarjan. *Three Partition Refinement Algorithms*, SIAM Journal of Computing, **16/6**(1987), 973–989.

Appendix

Among the different proposals of high level extensions of Generalized Stochastic Petri Nets (GSPNs) available in the literature, Stochastic Well-formed Nets (SWNs) offer the advantage of an *automatic* detection of the model symmetries, and of their exploitation in the model solution, through the concept of *symbolic marking*. In addition to the usual Coloured Petri Net (CPN) structure and annotations, SWN adopt a particular syntax to specify colour domains, arc functions and predicates. Nevertheless, SWNs and CPNs possess the same expressive power.

In Section A we give a description of the SWN formalism, and its semantics. In Section B we give some basic definitions of the (E)SRG approach.

A SWN definition

The starting point in the structured definition of the SWN colour syntax is the set of *basic colour classes* $\{C1, \ldots, Cn\}$. A basic colour class Ci is a nonempty, finite (possibly circularly *ordered*) set of colours; intuitively, a basic colour class can be defined as a set of colours identifying objects of the same nature. A basic colour class C is ordered if a *successor function* is defined on its elements (actually it is a successor modulo $|C|$), such that it induces a circular ordering on the class elements. Examples are the class of processors, the class of memories, the class of busses, etc. An example of ordered class is the class of processors connected in a ring topology. Basic colour classes are disjoint (i.e., $\forall i, j : i \neq j, Ci \quad \cap \quad Cj = \emptyset$), moreover, a class may be partitioned into several *static subclasses* ($Ci = Ci_1 \cup \ldots \cup Ci_{ns_i} \forall j, k : j \neq k, Ci_j \cap Ci_k = \emptyset$): colours belonging to different static subclasses represent objects of the same type but with different behaviour, for example the basic colour class of processors could be partitioned into two (disjoint) static subclasses, one containing the *fast* processors and the other containing the *slow* ones. In the example of Fig. 1 there is only one class, C, representing process identifiers. This class has cardinality three and is split into three static subclasses of cardinality one.

The place colour domains are defined by composition through the Cartesian product operator of basic colour classes. The colour domain of a place is similar to a C-language structure declaration, i.e., the information associated with tokens comprises one or more *fields*, each field in turn has a type selected from the set of basic colour classes $\{C1, \ldots, Cn\}$. The identification of the fields is positional (there is no name associated with a field). The colour domains of the places in Fig. 1 are very simple: all of them are simply C.

The transition colour domains are used to define the *variables* of transitions and their type; each variable has a type selected from the basic colour classes, moreover restrictions can be defined on the possible colour instances of a transition (i.e., on the possible values assigned to variables) by means of a *transition predicate*, or *guard*. Therefore, the definition of a transition colour domain comprises two parts: a list of typed variables, and the *guard*, defined as a Boolean expression of (a restricted set of) basic predicates on the variables. The variables of a transition are all the *variables* appearing in the arc functions of the input, output and inhibitor arcs of the transition [2] . We shall denote $Var_i(t)$ the subset of transition t variables of type Ci, and $Var(t)$ the whole set of transition t variables. Most transitions in the example of Fig. 1 have only one variable p of type C, and have a guard which is always *true* (in this case the guard just does not appear in the picture). Transitions t_6 and t_4 have two variables, p and q, both of type C, moreover t_4 has a guard which allows to select the admissible instances of t_4.

Definition A.1 [Standard Predicates] A standard predicate (or guard) associated

[2] Observe that the *scope* of a variable appearing on a given arc is the corresponding transition: instances of the same variable appearing on arcs of the same transition actually represent the same object, while different instances of the same variable associated with different transitions are independent.

with a transition t is a boolean expression of *basic predicates*. The allowed basic predicates are: $x = y$, $x =!y$, $d(x) = Ci_j$, $d(x) = d(y)$, where $x, y \in Var_i(t)$ are variables of t of the same type, $!y$ denotes the successor of y (assuming that the type of y is an ordered class), and $d(x)$ denotes the static subclass x belongs to.

As mentioned before, in our running example only one transition has a guard, namely t_4. Actually the syntax $[p > q]$ is not allowed in SWNs, instead the correct syntax for this guard, using SWN standard predicates is:

$$(p \in C_2 \textit{ and } q \in C_1) \textit{ or } (p \in C_3 \textit{ and } (q \in C_1 \textit{ or } q \in C_2))$$

where C_i is the static subclass of C containing only element pr_i.

Arc functions are defined as weighted (and possibly guarded) sums of tuples, the elements composing the tuples are in turn weighted sums of *basic functions*, defined on basic colour classes and returning multisets of colours in the same class. Given this definition, it is more appropriate to refer to the arc inscriptions as *arc expressions* instead of arc functions.

Definition A.2 [Arc expressions] An arc expression associated with an arc connecting place p and transition t has the following form:

$$\sum_k \delta_k.[pred_k]F_k$$

where δ_k is a positive integer, F_k is a function and $[pred_k]$ is a standard predicate. The value of function "$[pred]f$" is given by:

$$[pred]f(c) \quad = \quad \text{If } pred(c) \text{ then } f(c) \text{ else } 0.$$

Each $F_k : cd(t) \rightarrow Bag(cd(p))$ is a function of the form

$$F = \bigotimes_{Ci \in \ \mathcal{C}} \bigotimes_{j=1,\ldots,e_i} f_i^j = \langle f_1^1, \ldots, f_1^{e_1}, \ldots, f_n^1, \ldots, f_n^{e_n} \rangle$$

with e_i representing the number of occurrences of class Ci in colour domain of place p, i.e.,
$cd(p) = \bigotimes_{Ci \in \mathcal{C}} \bigotimes_{j=1,\ldots,e_i} Ci = \bigotimes_{Ci \in \mathcal{C}} Ci^{e_i}$.

Each function f_i^j in turn is defined as:

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

where S_{Ci_q}, x and $!x$ are basic functions (defined hereafter), $\alpha_{i,q}$, β_x and γ_x are natural numbers.

The multiset returned by a tuple of basic functions is obtained by Cartesian product composition of the multisets returned by the tuple elements. As it can be observed in the formal definition of arc expressions, there are three types of basic functions: the *projection* function, the *successor* function and the *diffusion/synchronization* function. The syntax used for the projection function is x, where x is one of the transition variables (it is called projection because it selects one element from the tuple of variable values defining the transition colour instance), the syntax used for the successor function is $!x$ where x is again one of the transition variables, it applies only to ordered classes and returns the successor of the colour assigned to x in the transition colour instance. Finally, the syntax for the diffusion/synchronization function is S_{Ci} (or S_{Ci_j}): it is a constant function that returns the whole set of colours of class Ci (of static subclass $Ci_j \subset Ci$). It is called synchronization when used on a transition input arc because it implements a synchronization among a set of coloured tokens contained into a place, while it is called diffusion when used on a transition output arc because it puts several tokens of different colours into a place.

Most arc functions in our running example are very simple (1-tuples using only the projection function, e.g. $\langle p \rangle$). The function $\langle p + q \rangle$ on the arc from GS to t_4 returns a set with two elements of C (the values assigned to variables p and q by the considered instance of t_4). Function $\langle S - p \rangle$ on the arc from FDR to t_3 returns a set of cardinality $|C| - 1$ comprising all the elements of C except the one assigned to variable p by the considered instance of t_3.

Definition A.3 [Stochastic Well-formed Nets]

A *Stochastic Well-formed Net* is a nine-tuple:

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

where:

(i) P and T are disjoint finite non empty sets (the places and transitions of $\mathcal{N}$),

(ii) $\mathcal{C} = \{C1, \ldots, Cn\}$ is the finite set of finite *basic* colour classes, (we use the convention that classes with index up to h are not ordered, while classes with higher index are ordered),

(iii) cd is a function defining the colour domain of each place and transition; for places it is expressed as Cartesian product of classes of $\mathcal{C}$ (repetitions of the same class are allowed), for transitions it is expressed as a pair $\langle$ variable types, guard $\rangle$ defining the possible values that can be assigned to transition variables in a transition instance; guards must be expressed in the form of *standard predicates*,

(iv) $\mathbf{Pre}[p,t], \mathbf{Post}[p,t] : cd(t) \rightarrow \mathrm{Bag}(cd(p))$ are the *pre-* and *post- incidence matrices*, expressed in the form of *arc expressions*,

(v) $\mathbf{Inh}[p,t] : cd(t) \rightarrow \mathrm{Bag}(cd(p))$ is the matrix defining the inhibitor arcs and associated arc expressions,

(vi) $\mathbf{pri} : T \rightarrow \mathbb{N}$ is the priority function,

(vii) $\mathbf{w}$ is a T indexed vector of functions that assigns rates and weights to transitions: $\mathbf{w}[t] : cd(t) \times (\bigotimes_{p \in P} Bag(cd(p))) \rightarrow \mathbb{R}^+$

Although the weight function $\mathbf{w}$ can be both colour and marking dependent, in practice the definition of $\mathbf{w}$ is often simplified so that the rate of a given instance depends only on the static subclass to which the elements assigned to the variables belong.

An ordinary marking m is a function mapping each place p into a multiset (bag) on $cd(p)$ (denoted as a weighted sum). Hence a place can contain more than one token of a given colour. The *initial marking* is denoted as m_0. Tokens are denoted by tuples of objects. In our running example the initial marking is $ID(pr_1+pr_2+pr_3)+PR(pr_1+pr_2+pr_3)$.

Given a marking m and a transition t, we call *instance* of t in m a binding c of the variables of $Var(t)$ to objects in the appropriate colour class. A transition instance is denoted (t,c).

An instance (t,c) such that:

- $pred(t)(c)$ holds true;
- $\forall p$: $\mathbf{Pre}(p,t)(c) \le m(p)$;
- $\forall p : \mathbf{Inh}(p,t)(c) = 0$ or $\mathbf{Inh}(p,t)(c) > m(p)$.

is said to *have concession* in m. An instance is *enabled* in m iff it does not exist any instance of a higher priority transition having concession in m.

An enabled instance (t,c) may fire, producing a new marking m' ($m[t,c > m'$) such that for each place p, $m'(p) = m(p) - \mathbf{Pre}(p,t)(c) + \mathbf{Post}(p,t)(c)$. A *path* is a sequence $m_1 \xrightarrow{(t_1,c_1)} m_2 \xrightarrow{(t_2,c_2)} m_3 \ldots m_n \xrightarrow{(t_n,c_n)} m_{n+1}$. The set of all markings reachable form m is denoted by $[m\rangle$ ($[m_0\rangle$ is called *reachability set*).
In the initial marking of the SWN in Fig. 1 there are three enabled instances of t_1, characterized by the assignment $p = pr_i, \quad i = 1,2,3$. After firing instance $(t, \langle pr_i \rangle)$, a token of colour pr_i is withdrawn from places ID and PR, and a token of the same colour is put into RQ, so that the instance $(t_2, \langle pr_i \rangle)$ becomes enabled.

The interest in SWN is due to the Symbolic Marking (SM) and Symbolic Firing notions that allow to build a reduced representation of the RG called Symbolic RG (SRG).

A symbolic marking (SM) representation $\widehat{m}$ comprises two parts: a part representing the distribution of coloured tokens into places, and a part specifying the so called *dynamic subclasses* ($Z^k_{Ci_j}$). Section 2 contains an informal introduction to the symbolic marking concept: let us show here an example of symbolic marking applied to our running example; in particular we would like to demonstrate the effect of the partition into static subclasses on the aggregation of ordinary markings into symbolic markings, which is also a bridge towards the Extended Symbolic Marking concept. Let us consider a variation of our running example in which t_4 has no guard and C is not partitioned into static subclasses. The initial marking of this SWN model can be represented in a symbolic form as follows: $\widehat{m}_0 = ID(Z^1_C) + PR(Z^1_C), \ |Z^1_C| = 3$. The dynamic subclass Z^1 represents (any) three objects (its cardinality is 3) of basic colour class C. Since $|C| = 3$, there is only one way of assigning three objects of C to Z^1_C, so that this symbolic marking represents only one ordinary marking. The three objects represented by Z^1 are present in place ID and in place PR.

Let us now consider the following symbolic marking, reachable from the initial one by firing transition t_1: $\widehat{m}_1 = ID(Z^1_C) + PR(Z^1_C) + RQ(Z^2_C), \ |Z^1_C| = 2, |Z^2_C| = 1$. It represents three ordinary markings, corresponding to the possible assignment of the objects of C to the two dynamic subclasses (1: $Z^1_C = \{pr_1, pr_2\}, Z^2_C = \{pr_3\}$, 2: $Z^1_C = \{pr_1, pr_3\}, Z^2_C = \{pr_2\}$, 3: $Z^1_C = \{pr_2, pr_3\}, Z^2_C = \{pr_1\}$). It can be interpreted as a *pattern* for a marking in which one of the three processes of C (represented by the symbol Z^2_C) has issued a request ($RQ(Z^2_C)$), while the remaining two processes (represented by the symbol Z_C^1) are idle $(ID(Z_C^1) + PR(Z_C^1))$. Observe that the three ordinary markings represented by this symbolic marking can be obtained from each other by applying a permutation on the elements of C.

Let us now discuss the symbolic firing rule, allowing us to build the symbolic reachability graph starting from a symbolic initial marking.

A *symbolic instance* of a transition t is defined by specifying an assignment of dynamic subclasses to the transition variables. For example, we may consider the following symbolic instance enabled in the initial symbolic marking of our running example: $(t_1, \langle Z_C^1 \rangle)$. The assignment of a dynamic subclass to a transition variable corresponds to the assignment of *any* element of that dynamic subclass to the variable so that this symbolic firing represents the three ordinary firings: $(t, \langle pr_i \rangle),\ i = 1, 2, 3$. The restricted set of basic functions defined in the SWN formalism can be easily extended to work on dynamic subclasses, so that the state change can be defined at the symbolic marking level. So it is possible to automatically obtain the symbolic marking $\widehat{m}_1$ from $\widehat{m}_0$ applying the symbolic firing rule (see [3] for the details).

Let us discuss what happens when $|C|$ is split into three static subclasses (this is needed to be able to express the guard associated with t_4). Since the elements grouped into a given dynamic subclass must belong to a unique static subclass, when C is split all symbolic markings will have exactly three dynamic subclasses, $Z_{C_i}^1, |Z_{C_i}^1| = 1,\ i = 1, 2, 3$. The new representation of the initial marking will be:

$$\widehat{m}_0' = ID(Z_{C_1}^1 + Z_{C_2}^1 + Z_{C_3}^1) + PR(Z_{C_1}^1 + Z_{C_2}^1 + Z_{C_3}^1), |Z_{C_1}^1| = 1, |Z_{C_2}^1| = 1, |Z_{C_3}^1| = 1$$

In this situation, there are three enabled symbolic transition instances, namely $(t_1, \langle Z_{C_1}^1 \rangle)$, $(t_1, \langle Z_{C_2}^1 \rangle)$, $(t_1, \langle Z_{C_3}^1 \rangle)$. The symbolic markings reached by firing these symbolic instances are:

$$\widehat{m}_1' = ID(Z_{C_1}^1 + Z_{C_2}^1) + PR(Z_{C_1}^1 + Z_{C_2}^1) + RQ(Z_{C_3}^1), |Z_{C_1}^1| = 1, |Z_{C_2}^1| = 1, |Z_{C_3}^1| = 1$$

$$\widehat{m}_1'' = ID(Z_{C_1}^1 + Z_{C_3}^1) + PR(Z_{C_1}^1 + Z_{C_3}^1) + RQ(Z_{C_2}^1), |Z_{C_1}^1| = 1, |Z_{C_2}^1| = 1, |Z_{C_3}^1| = 1$$

$$\widehat{m}_1''' = ID(Z_{C_2}^1 + Z_{C_3}^1) + PR(Z_{C_2}^1 + Z_{C_3}^1) + RQ(Z_{C_1}^1), |Z_{C_1}^1| = 1, |Z_{C_2}^1| = 1, |Z_{C_3}^1| = 1$$

Observe that when C was not split into static subclasses these markings were represented by only one symbolic marking: $\widehat{m}_1$. Actually, when all classes are split into cardinality one static subclasses, each symbolic marking represents only one ordinary marking and the SRG coincides with the RG. The reason is that the ordinary markings grouped into a symbolic marking must be obtained from each other by applying a permutation on the objects of basic colour classes that preserves the partition into static subclasses. Clearly if all static subclasses have cardinality one, the only admissible permutation is the identity.

B The Extended Symbolic RG

The ESRG marking idea consists of *disregarding* the partition into static subclasses unless it is really needed, thus allowing to further aggregate the state space.

Definition B.1 (Extended Symbolic Marking)
An ESM $\widehat{\widehat{m}}$ is a pair $\langle SR, E\rangle$: Component SR (called symbolic representation of $\widehat{\widehat{m}}$) is a standard symbolic marking, except for the fact that dynamic subclasses are defined disregarding the partition of distinguished classes into static subclasses; Component E is a set of reachable eventualities: each eventuality defines a refinement of SR dynamic subclasses with respect to the static subclass partition : $e \in E$ is a family of functions, $\{e^{(i)} : \hat{\hat{C}}i \rightarrow Bag(\{1, \ldots, ns_i\})\}$, where $Ci \in \mathcal{C}_D$ ($\mathcal{C}_D$ is the subset of $\mathcal{C}$ corresponding to the classes with several static subclasses, called *distinguished classes*), $\hat{\hat{C}}i$ is the set of type Ci dynamic subclasses in SR, and ns_i is the number of static subclasses of Ci, such that:

$\forall Z_i^j \in \hat{\hat{C}}i \;: |e^{(i)}(Z_i^j)| = |Z_i^j|$,
$\forall 1 \leq k \leq ns_i \;: \sum_{Z_i^j \in \hat{\hat{C}}i} e^{(i)}(Z_i^j)(k) \leq |Ci_k|$.

$e^{(i)}(Z_i^j)(k)$ (the multiplicity of element k in Bag $e^{(i)}(Z_i^j)$) represents the number of (arbitrarily chosen) objects of Z_i^j that are assigned (instantiated) to static subclass Ci_k.

Turning to our running example, even if C is split into three static subclasses, the three symbolic markings $\widehat{m}'_1, \widehat{m}''_1$ and $\widehat{m}'''_1$ are now grouped into a single ESM $\widehat{\widehat{m}}_1$, whose symbolic representation coincides with that of symbolic marking $\widehat{m}_1$ defined in the previous section (before splitting of C). The symbolic markings $\widehat{m}'_1, \widehat{m}''_1$ and $\widehat{m}'''_1$ are now the *eventualities* of $\widehat{\widehat{m}}_1$. Actually these eventualities are not explicitly represented in the ESRG of Fig. 2 because no asymmetric transition is enabled in this ESM.

As an example of the representation of eventualities, let us consider the ESM $\widehat{\widehat{m}}_{12} = ID(Z^1) + GS(Z^2), |Z^1| = 1, |Z^2| = 2$ in Fig. 2. The first eventuality of this ESM is characterized as follows: the unique element of dynamic subclass Z^1 is assigned to static subclass C_1, one of the two elements of Z^2 is assigned to static subclass C_2 and the other element of Z^2 is assigned to static subclass C_3. The notation used is the following: the dynamic subclasses of $\langle SR_{12}, e_1\rangle$ are $Z^1_{C_1}, Z^1_{C_2}$ and $Z^1_{C_3}$, all of cardinality 1, and they are a refinement of the SR dynamic subclasses: $Z^1 = Z^1_{C_1}$, $Z^2 = Z^1_{C_2} \cup Z^1_{C_3}$. The representation of the symbolic marking corresponding to $\langle SR_{12}, e_1\rangle$ is obtained from SR_{12} by substituting the dynamic subclasses according to the above refinement definition.

The transitions instances in the ESRG can be either generic or instantiated. They are generic when they refer to the SR dynamic subclasses, while they are instantiated when they refer to the instantiated dynamic subclasses of the eventualities. For example transition instance $(t_5, \langle Z^2\rangle)$ enabled in $\widehat{\widehat{m}}_{11}$ is generic symmetric. Instead the instance $(t_4, \langle Z^1_{C_2}, Z^1_{C_1}\rangle)$, enabled in eventuality $\widehat{\widehat{m}}_{10}.e_3$, is instantiated since it refers to the dynamic subclasses of the eventuality.