Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications

Giovanni Chiola *
Dip. Informatica, Università di Torino

Claude Dutheillet
Lab. MASI, Université Paris 6

Giuliana Franceschinis
Dip. Informatica, Università di Torino

Serge Haddad
Lab. MASI, Université Paris 6

Abstract

The class of Stochastic Well Formed Colored Nets (SWN) was defined as a syntactic restriction of Stochastic High-Level Nets. The interest of the introduction of restrictions in the model definition is the possibility of exploiting the Symbolic Reachability Graph (SRG) to reduce the complexity of Markovian performance evaluation with respect to classical Petri net techniques. It turns out that SWNs allow the representation of any color function in a structured form, so that any unconstrained high-level net can be transformed into a well formed net. Moreover, most constructs useful for the modeling of distributed computer systems and architectures directly match the "well form" restriction, without any need of transformation. A non trivial example of the usefulness of the technique in the performance modeling and evaluation of multiprocessor architectures is included.

*This work has been done while G. Chiola was visiting researcher at the Lab. MASI of the University of Paris 6, with the financial support of a NATO-CNR annual research grant. The work has been partially supported by the CNR project "Progetto Finalizzato Sistemi Informatici e Calcolo Parallelo", grant 91.00879.PF69 and by an international cooperation grant from the University of Torino.

Index Terms

- High-Level Petri nets
- model symmetry
- symbolic reachability graph
- performance evaluation
- computational complexity
- Markov chains
- lumpability condition
- multiprocessor systems
- bus and memory contention

1 Introduction

Stochastic Petri nets have been proposed in the literature as a good modeling tool for the study and performance evaluation of multiprocessor computer architectures [1, 2, 3]. The technique is easier to use than more classical Markovian modeling techniques, but still leads to models whose size is too large when modeling realistic systems. Techniques for the development of "compact" or "folded" models [4, 5] have been proposed, but they had not been used by many researchers outside the restricted group that developed them. This is due to the deep knowledge required by the modeler of the behavioral characteristics and symmetries of the system under study. High level Petri net models have been proposed from time to time as a more adequate tool (see, e.g., [6, 7]) for the modeling of complex multiprocessor systems. Although easier to use even by inexperienced modelers because of their higher level of abstraction, classical high-level Petri nets do not alleviate the need for a thorough understanding of the symmetries of the system in order to allow the performance evaluation of reasonably large systems.

Regular Nets (RN) have been proposed in [8] as a restriction of Colored Petri Nets (CPN) [9]. The interest in introducing such a restriction on the color domains and on the arc functions was that complete algorithms have been proposed for the computation of flows, reductions, and the definition of a Symbolic Reachability Graph (SRG) in this case. Recently, stochastic models based on RNs called Regular Stochastic Petri Nets (RSPN) have been introduced for performance evaluation purposes [10, 11]. The steady-state performance of an RSPN model can be obtained by numerically solving a Markov chain (MC) corresponding to the SRG generated by the RSPN. The complexity in the computation of this solution is polynomial in the number of symbolic markings of the SRG, which can be much less than the number of ordinary markings generated by the Place/Transition net (P/T) resulting from the unfolding of the RN.

Some kinds of symbolic marking representations to exploit the symmetries of the reachability graph (RG) have already been proposed for general CPNs [12, 13, 7, 14] as well, but in all these cases heuristics were needed to decide the type of aggregations. These heuristics were based on an explicit knowledge by the modeler of the symmetries present in a particular model. Thus none of these methods could be implemented in a general algorithmic form. From this point of view, the superiority of RNs was due to the availability of a generic *symbolic firing rule* which allows the computation of the SRG without any actual instantiation of colors and without explicit knowledge of the symmetry of the model.

This use of the SRG for performance evaluation purposes is the main motivation that led us to the proposal of the extended SRG computation for more general classes of CPNs called "well-formed" colored nets (WN) [15]. The original RNs and their related SRG generation algorithm can be used to model many interesting systems, but unfortunately the strong restrictions imposed on the definition of the arc labeling functions, the color sets, and the initial marking, prevent their use in general cases.

In this paper we present the new class of stochastic well-formed colored nets (SWN) as extensions of RSPNs together with an extended SRG construction algorithm that allows an optimized Markovian analysis exploiting the system symmetries. We assume that our SWNs are always bounded in order to define a finite state space. The SRG defined for this class of models allows the same kind of performance evaluation presented in [11] in the case of general models. Moreover, this extended algorithm has the same advantages as the one originally proposed for RN, i.e.: 1) it uses a symbolic firing rule, so that both its time and space complexities depend only on the size of the SRG, and not on the size of the actual RG; 2) it does not require any particular heuristics to explicitly define the symmetries of the model; 3) it exploits the information that is implicit in the well structured function and color domain definitions.

From the modeling power point of view any general CPN model can be translated into an equivalent WN model with the same underlying structure; only the expression of the color functions and of the composition of color classes is re-written in a more explicit (and parametric) form, in terms of the basic constructs provided by the WN formalism. Moreover, in practical modeling this formalism translation is hardly needed: most (if not all) CPN models published in the literature can be directly represented as WNs, even without exploiting the power of predicate guards on the arc labeling functions.

We also present a complete modeling example taken from the literature on multiprocessor computer architecture. The example shows how the technique can be exploited even by non expert modelers to obtain good performance models of complex systems. The symmetries of the system are automatically taken into account by the proposed algorithm without any explicit intervention of the modeler. The time and space complexity of the analysis is however equal to the one of the best model devised by an expert modeler.

The balance of the paper is as follows. Sections 2 presents the WN formalism, and Section 3 presents the SRG, a sketch of the SRG construction algorithm with some examples, and some relevant mathematical properties of the graph. Section 4 contains the definition of SWNs, and the performance evaluation technique. Section 5 presents a non trivial example of the application of SWNs to the performance evaluation of a well known multiprocessor computer architecture. Finally, Section 6 contains some concluding remarks and perspectives of this work. The formal notation has been somewhat sacrificed, and the presentation has been based mainly on examples to provide easier comprehension for the non expert. A more rigorous notation and the formal proofs of many results can be found in [15] or in [16].

2 Well-Formed Nets: an introduction

We assume the reader is already familiar with the Petri net notation both uncolored [17] and colored [18] versions. We also assume the reader to be familiar with the GSPN formalism [19] and its multiprocessor applications [4].

2.1 Notation and basic definitions

We start by giving some basic definition and a short outline of the notation used in the paper.

Definition 2.1 *A* multiset *a over a non-empty set A is a mapping* $a \in [A \to \mathbb{N}]$, *we use the notation* $Bag(A)$ *to denote a multiset over A. Intuitively, a multi-set is a set that can contain several occurrences of the same element. It can be represented by a formal sum:* $a = \sum_{x \in A} a(x)\ x$.
The coefficient $a(x)$ *is called* multiplicity *of* x *in* a.

A multiset a_2 *is contained into the multiset* a_1, *denoted* $a_2 \leq a_1$ *iff* $\forall x \in A, a_2(x) \leq a_1(x)$.

Definition 2.2 *The* summation, subtraction *and* scalar multiplication *of multisets are defined as follows:*

- $a_1 + a_2 = \sum_{x \in A}(a_1(x) + a_2(x))x$
- $a_1 - a_2 = \sum_{x \in A}(a_1(x) - a_2(x))x$ *provided that* $a_2 \leq a_1$
- $n\ a = \sum_{x \in A}(n\ a(x))x$

Given a family of sets $\{A_i,\ i = 1, \ldots, n\}$ we denote $A_1, \ldots, A_n$ a new set obtained by Cartesian product of the sets in the family; an element in this new set is denoted $< e_1, \ldots, e_n >$ where $e_i \in A_i$.

By generalization, if a_i is a multiset over A_i, we denote $< a_1, \ldots, a_n >$ the multiset over $A_1, \ldots, A_n$ obtained by Cartesian product of the component multisets a_i. The multiplicity of the elements in the resulting multiset is defined as follows:

$$\forall < e_1, \ldots, e_n > \in A_1, \quad \ldots, A_n, < a_1, \ldots, a_n > (< e_1, \ldots, e_n >) \ = \ a_1(e_1) \cdot a_2(e_2) \cdot \ldots \cdot a_n(e_n)$$

2.2 WN informal definition

The introduction of Colored Petri Nets (CPN) as well as other High Level Petri Nets formalisms (e.g., Pr/T nets) was crucial from the point of view of the expressive power of this class of formalisms. The possibility of associating information with tokens and of parameterizing transition firing made it possible to represent very concisely systems that would have required huge uncolored nets to be described.

Well Formed Colored Nets (WNs) are substantially identical to CPNs from the expressive power point of view (see [20] for a proof of this statement). However the syntactic definition of WNs leads to new, more efficient analysis algorithms based on the original concept of *symbolic marking.* In the sequel of this section we give an informal description of the WN formalism. The syntax used in the explanation and in the examples is the one accepted by the WN design and analysis tool prototype that has been developed at the Computer Science Dept of the University of Torino [21] and that will be soon integrated in the package *GreatSPN* [22].

As in PNs, places of WNs together with their marking play the role of describing the system state while transitions represent events that cause the state changes. In WNs a token can incorporate some information, indeed a token can be regarded as an instance of a data structure with a certain number of fields whose semantics depend on the place the token belongs to. The definition of the "data type" associated with each place is called *place color domain* and is similar to a *C* structure declaration. The fields data types are selected from a set of basic types called *basic color classes*[1]. The specification of the basic color classes is part of the net definition. In our tool basic color classes have always finite cardinality and are defined by enumeration of the elements[2].

Often it may be useful to partition a basic color class into disjoint subclasses of objects with some common property. For example the class of *processes* could be partitioned into two subclasses: the *low priority processes* subclass and the *high priority processes* one. In the WN terminology these are called *static subclasses* (as opposite to *dynamic subclasses* that will be introduced to define symbolic markings). A basic class may be *ordered* so that a successor function is defined on its elements. The ordering is assumed to be circular, so that the successor function applied to the last element returns the first one. Here is the grammar for the definition of basic color classes and their static subclasses within our tool. The syntax is depicted in Figure 1, where "stat_sbc_id" is simply a string denoting a

Basic class grammar			**Examples**
class	→	classtype staticlist	$Msgs = UData\_msgs, Ack\_msg$
staticlist	→	stat_sbc_id | staticlist , stat_sbc_id	
classtype	→	O | U	

Static subclass grammar			**Examples**
static	→	STRING[NUM-NUM] |	$Data\_msgs \; = \; m[1-3]$
		[objectlist]	$Ack\_msg \; = \; [ack]$
objectlist	→	STRING | objectlist , STRING	

Figure 1: Class definition syntax

static subclass identifier;

[1] The fields are not named. Their identification is positional.

[2] Shortcuts are allowed to make the definition more concise: for example the set $Sites = \{s_1, \ldots, s_{10}\}$ can be specified using the syntax $s[1-10]$.

In the example in the same figure the basic class $Msgs$ of possible messages is an unordered (U) class split in two static subclasses $Data\_msgs$ and Ack_msg of cardinality 3 and 1 respectively.
The color domain of place p is denoted $C(p)$ and the syntax for its definition is shown in Figure 2 where "class_id" is simply a string denoting a previously defined basic color class.

Place color domain grammar			**Examples**
Placedomain	$\rightarrow$	classlist | ϵ	$C(T.Message\ buffer) = Sites, Sites, Msgs$
classlist	$\rightarrow$	class_id | classlist, class_id	$C(Idle\ sites) = Sites$

Figure 2: Place colour definition syntax

In WNs a place can be used to represent the value of a variable of given type provided that it never contains more than one token (the empty place could represent either an uninitialized variable or some fixed default value in the variable domain) and the place color domain is equal to the variable data type. Another use of places in WN models is for the representation of the state of a (multi)set of possibly distinguishable objects. The notation $M(p)$ denotes the marking of place p i.e., the multiset of $C(p)$ contained in p according to marking M.

The transitions in WNs can be considered as procedures with formal parameters. The formal parameters are called *transition color domain*; their declaration is part of the net description and the type associated with each parameter must be a basic color class. A transition color domain is defined in the same way as a place color domain. The list of classes in the color domain defines the type associated with the transition parameters. The color domain of a transition t (denoted $C(t)$) is constrained by the color domains of its input, inhibitor and output places. We shall see later on that the relation between transition and place color domains is defined through the *arc expressions*.

A transition whose formal parameters have been instanced to actual values is called *transition instance*. We use the notation $[t, c]$ for an instance of transition t, where c represents the *assignment* of actual values to the transition parameters. Observe that an assignment c is actually an element of the set $C(t)$ and for this reason it is often referred to as a "color instance" of t.
In order to fire a transition, it is necessary to specify actual values for its formal parameters (it is similar to the execution of a procedure call), i.e., we can only fire *transition instances*. The enabling check of a transition instance and the state change caused by its firing depend (again) on the *arc expressions* that label the arcs connected to the transition. Observe that many instances of the same transition can be concurrently enabled. They are considered as independent, concurrently occurring events (unless they are in conflict one with the other).

The arc expressions are (formal) sums of tuples; each element of a tuple in turn is a weighted sum of terms denoting multisets of the basic color classes. The arc expressions syntax is described in Figure 3.

The arc expressions are structured according to the corresponding place color domain. If the place color domain contains k basic color classes (i.e., k "fields"), then the corresponding arc expression is a weighted sum of k-tuples. The whole expression denotes a multiset in the Cartesian product of the basic color classes composing the corresponding place color domain.

The j^{th} element in each k-tuple is an expression denoting a multiset of C_j, where C_j is the "type" of the j^{th} "field" in the place color domain. This element is a weighted sum of three types of terms: (1) variable (e.g. $rightfork$); (2) successor function applied to a variable (e.g. $\bigoplus rightfork$); (3) basic class/static subclass identifiers (e.g. S_{Msgs} or S_{Data_msgs}).

The first two terms denote an object in C_j and the successor of an object in C_j respectively, S_{class_name} or simply S denotes the set *class_name* (C_j in this case), $S_{subclass\_name}$ denotes the static subclass *subclass_name*.

An arc expression that contains variables can be interpreted as a *pattern* standing for any multiset that can be obtained binding the variables to actual elements in the proper basic color class. We call *assignment* a collection of variable bindings.

For example let us consider the tuple $< src, dest, msg, \bigoplus cnt >$, and let us assume that the corresponding place color domain is $Sites, Sites, Msgs, MsgNumbers$. The variables *src*, *dest*, *msg* and *cnt* stand for any object in the corresponding basic color class, so that the arc expression is a pattern for any set of cardinality one whose only element is a four-tuple of elements from the place color domain. The expression $< rightfork > + < \bigoplus rightfork >$, where variable $rightfork$ is of type $Philosophers$, stands for a set of cardinality two containing any element of $Philosophers$ and its successor. The expression $< broadcast\_msg, S_{Sites} >$ denotes a set of cardinality $|Sites|$ of pairs. The set is obtained by applying the Cartesian product operator to any cardinality one subset of basic class

Arc expression grammar			**Examples**
arc_expr	→	ϵ| sum_tuples	$< processor, memory >$
sum_tuples	→	coeff optional_predicate <expr_list> | sum_tuples plusop coeff optional_predicate <expr_list>	$< broadcast\_msg, S_{Sites} >$ $\vert Sites \vert < S_{Ack\_Msg} >$ $< src, dest, msg, \bigoplus cnt >$
expr_list	→	expr_kernel | expr_kernel, expr_list	$< rightfork > + < \bigoplus rightfork >$
expr_kernel	→	term | expr_kernel plusop term	$< phonenum1 + phonenum2, channel >$
term	→	subclass_term | class_term | var_term | succ_term	$< S - source >$
subclass_term	→	coeff S staticname	
class_term	→	coeff S | coeff S classname	
var_term	→	coeff STRING	
succ_term	→	coeff ⊕ STRING	
coeff	→	NUM | VBAR staticname VBAR | VBAR classname VBAR | ϵ	
plusop	→	+ | −	

Figure 3: Arc expression definition syntax

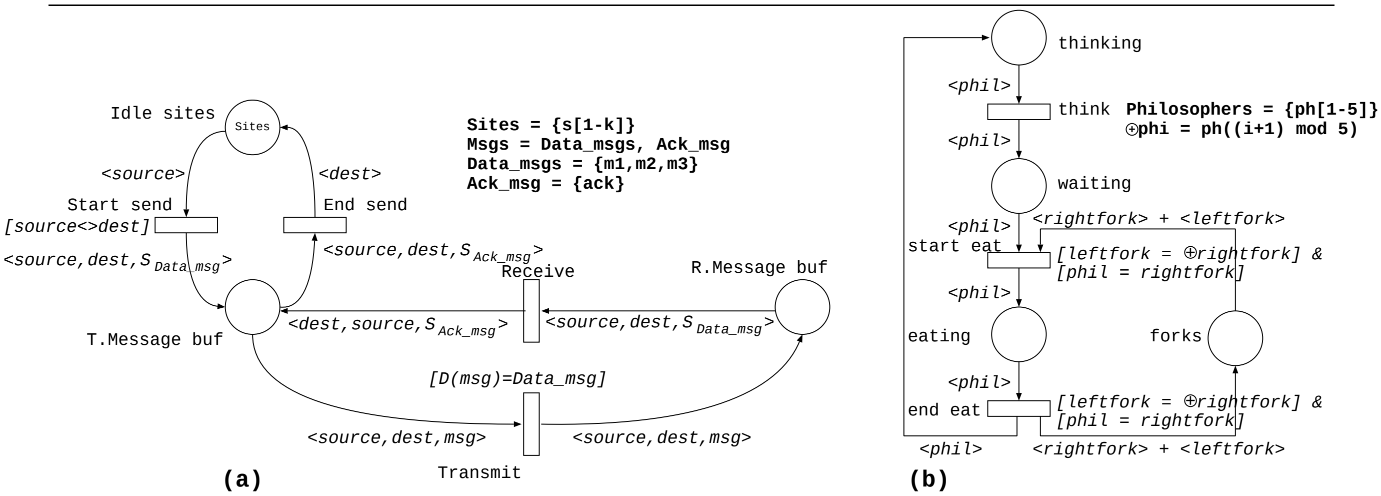

Figure 4: Examples of WNs

$Msgs$, the type of variable $broadcast\_msg$, and the set $Sites$. Finally expression $|Sites| < S_{Ack\_Msg} >$ is a multiset containing $|Sites|$ elements all with the same color $< ack >$.

The set of variables appearing in all arc expressions related to a single transition are its formal parameters. For example, in the net of Figure 4.(a) the parameters of transition $Start\ send$ are $source$ and $dest$, both ranging over class $Sites$. The parameters of transition $Transmit$ are $source$, $dest$, and msg. Variables $source$ and $dest$ represent objects from the basic class $Sites$ while variable msg represents an object from class $Msgs$.

Observe that when the same variable appears in many arc expressions related to the same transition, the different occurrences actually denote the same object, while when the same variable is used within several arc expressions each related to a different transition, there is no relation between the objects represented by the different variable occurrences.

Optionally a transition may have an associated *predicate*: a boolean expression of conditions on the transition formal parameters as defined by the grammar depicted in Figure 5, where $D(variable)$ denotes the static subclass of the object assigned to $variable$.

The enabling of a transition instance $[t, c]$ is determined by evaluating the transition predicate and the arc expressions of all input and inhibitor places with respect to the assignment c. Notice that in this case an arc expression can be seen as a *function*, whose arguments are the variables appearing in the expression itself. A transition instance $[t, c]$ is enabled iff the predicate evaluates to true, each input place contains the multiset resulting from the evaluation of the corresponding arc expression and for each inhibitor place, each tuple contained in it has a smaller multiplicity than the same tuple in the multiset resulting from the evaluation of the corresponding arc expression. An enabled transition instance $[t, c]$ can fire. The state change caused by the firing amounts to subtracting/adding from/to each input/output place p the multiset resulting by evaluating the corresponding arc expression through the assignment c.

In order to find all the enabled instances of a transition t it is possible to use either a "brute force" approach, i.e., generate all possible assignments and check for enabling, or a more sophisticated approach that takes into account the contents of input and inhibitor places to generate directly the subset of assignments that correspond to enabled instances.

In the second approach we consider each arc expression as a *pattern* for a multiset of colors. A *match* procedure can then be defined that is used to match the pattern specified by an arc expression with the marking of the corresponding input or inhibitor place. The match operation may fail or succeed; if it succeeds it returns a set of possible *local* assignments. The assignments obtained from all the input and inhibitor places must then be combined to form a set of proper *global* assignments, taking also into account the constraints due to the transition predicate. If a variable appears only on the output arc expressions, any assignment that satisfies the predicate constraints is valid.

Let us illustrate on some examples the enabling test and the firing. The net in Figure 4.(a) represents a communication system in which a site sends a message to a distant site. This message is split into three packets and each packet is transmitted separately over the network. Once the three packets have been received by the distant site, the latter sends an acknowledgement to the site that originated the communication. On reception of this acknowledgement, the communication ends and the sender becomes idle again. There are two basic classes in this model: the class of sites $Sites = \{s1, \ldots, sk\}$, and the class of messages, partitioned into two static subclasses: the subclass containing the (three) data messages and the subclass containing the (single) acknowledge message $Msgs = Data\_msgs \cup Ack\_msg = \{m1, m2, m3\} \cup \{ack\}$

Transition *Start send* firing is triggered by the presence of an element in place *Idle sites* to be matched with the variable *source*. The assignment to variable *dest* does not depend on any input place, however it must satisfy the predicate [$source \neq dest$]. Therefore, given a marking M all assignments of *source* and *dest* such that the site *source* is in *Idle sites* and *dest* is different from

Predicate grammar

predicate	$\rightarrow$	predicate OR pterm $\mid$ pterm
pterm	$\rightarrow$	term AND pfatt $\mid$ pfatt
pfatt	$\rightarrow$	(predicate) $\mid$
		D(STRING) eqop doperand $\mid$
		STRING eqop stroperand
doperand	$\rightarrow$	D (STRING) $\mid$ staticname
staticname	$\rightarrow$	STRING
staticindex	$\rightarrow$	NUM
stroperand	$\rightarrow$	STRING $\mid$ $\bigoplus$ STRING
eqop	$\rightarrow$	$= \mid <>$

Examples

$(leftfork = \bigoplus rightfork)$
$D(msg1) = D(msg2)$
$source <> destination$
$D(msg) = Ack\_Msg$

Figure 5: Predicate definition syntax

source correspond to enabled instances of transition *Start send*.
Thus if the instance[3] [*Start send*, (*source* = *s*1, *dest* = *s*4)] is fired the multiset {<**s1**>} is subtracted from place *Idle sites* while the multiset {<**s1,s4,m1**>, <**s1,s4,m2**>, <**s1,s4,m3**>} is added to place *T.Message buffer*.

Let us consider transition *Receive*. It is triggered by the presence in place *R.Message buf* of a set of three-tuples matching the pattern < *source*, *dest*,**m1**>, < *source*, *dest*,**m2**>, < *source*, *dest*,**m3**> (the boldface strings denote objects from basic classes), meaning that site *dest* has received all the three messages sent by *source*. In this case the assignment of proper values to the variables to find an enabled instance of *Receive* depends only on the input place marking. When firing the instance [*Receive*, (*source* =**si**, *dest* =**sj**)], the set {<**si**,**sj**,**m1**>, <**si**,**sj**,**m2**>, <**si**,**sj**,**m3**>} is subtracted from *R.Message buf*, and the set {<**sj**, **si**, **ack**>}, representing the acknowledge sent by *sj* to *si*, is added to *T.Message buffer*.

Observe that the matching procedure may be computationally expensive if the arc expressions are complex (in particular the arc expressions may contain predicates and in that case the match operation complexity may increase significantly). Our experience is that usually in big models only few arc expressions are complex, while many of them are very simple (e.g. identity functions) so that the more efficient procedure for generation of enabled instances can be applied to most transitions in the net.

2.3 WN formal definition

In this section we give the definition of WNs and formalize the enabling and firing of a transition instance.

Definition 2.3 (WN) *A well formed net* $WN = \langle P, T, C, J, W^-, W^+, W^h, \Phi, \pi, M_0 \rangle$ *is made of:*

P *the finite set of places;*

T *the finite set of transitions,* $P \cap T = \emptyset$, $P \cup T \neq \emptyset$;

C *the family of basic classes:* $C = \{C_1, \ldots, C_n\}$, *with* $C_i \cap C_j = \emptyset$ *(we denote* $I = \{1, \ldots, n\}$ *the ordered set of indexes);* C_i *is possibly partitioned in static subclasses:* $C_i = \bigcup_{q=1}^{n_i} D_{i,q}$;

$J : P \cup T \rightarrow Bag(I)$, *where* $Bag(I)$ *is the multiset on* I. $C(r) = C_{J(r)}$ *denotes the color domain of node* r;

$W^-, W^+, W^h : W^-(p,t), W^+(p,t), W^h(p,t) \in [C_{J(t)} \rightarrow Bag(C_{J(p)})]$ *the input, output, and inhibition functions are arc expressions;*

[3] We use the notations $(var_1 = value_1, \ldots, var_k = value_k)$ and $< value_1, \ldots, value_k >$ interchangeably to describe an assignment $c \in C(t)$.

$\Phi(t)$ *:* $C_{J(t)} \rightarrow \{True, False\}$ *is a standard predicate associated with the transition* t*. By default we will assume* $\forall t \in T$ *the standard predicate* $\Phi(t) = True$*;*

π *:* $T \rightarrow I\!N$ *the priority function. By default we will assume* $\forall t \in T$ *the value* $\pi(t) = 0$*;*

M_0 *:* $M_0(p) \in Bag(C(p))$ *is the initial marking of* p*.*

Definition 2.4 (Firing rule) *A transition instance* $[t, c]$ *(where* $c \in C(t)$*) is enabled in a marking* M *iff:*

i) $\forall p \in P,\ W^-(p,t)(c) \leq M(p) \wedge W^h(p,t)(c) > M(p),$ *and* $\Phi(t)(c)$
which is the expression of the firing rule in a net without priorities,

ii) $\forall t'$ *with* $\pi(t') > \pi(t),\ \forall c' \in C(t'),\ \exists p \in P$ *such that either* $W^-(p,t')(c') > M(p) \vee W^h(p,t)(c) \leq M(p)$ *or* $\neg\Phi(t')(c')$
which means that no higher priority transition is enabled.

The firing of transition instance $[t, c]$ *leads to a new marking* $M' = M[t, c\rangle$ *defined by:*
$\forall p \in P, \quad M'(p) = M(p) + W^+(p,t)(c) - W^-(p,t)(c)$

2.4 Basic Properties of WN

A major interest of WNs is that they provide a modeling framework in which symmetries appear naturally as a way of reducing the size and complexity of the representation. Another fortunate property of WNs is their modeling power equivalent to unconstrained colored nets. The two next propositions of WN illustrate this situation.

Proposition 2.1 *Any CPN can be transformed into a WN with the same basic structure, same color domains (possibly partitioned in static subclasses), equivalent arc labeling. See [20] for a proof.*

Definition 2.5 (Color permutation) *Let* $\xi = \{\ s = \langle s_1, \ldots, s_h,\ s_{h+1}, \ldots,\ s_n \rangle\ \}$ *be a subgroup of the permutations on* $C_1, \ldots, C_n$ *such that:*

- $\forall 0 < i \leq h$ s_i *is a permutation on* C_i *such that* $\forall D_{i,q},\ s_i(D_{i,q}) = D_{i,q}$*;*
- $\forall h < i \leq n$ s_i *is a rotation on* C_i *such that* $\forall D_{i,q},\ s_i(D_{i,q}) = D_{i,q}$*. Note that this condition implies that if the number of static subclasses of* C_i*,* $n_i > 1$ *then the only allowed rotation* s_i *is the identity.*

Let C_J *be a color domain and* $< c_1, \ldots, c_k > \in C_J,\ s \in \xi$*. Then* $s(< c_1, \ldots, c_k >)$ *is defined by :* $s(< c_1, \ldots, c_k >) = < s_{(1)}(c_1), \ldots, s_{(k)}(c_k) >$ *where* $s_{(i)}$ *is the permutation associated with the* i^{th} *basic class in color domain* C_J*.*

Definition 2.6 (Marking permutation) *Let M be a marking, and $s \in \xi$ a permutation. Then $M' = s.M$ is a marking defined by: $\forall p \in P,\ \forall c \in C(p), \quad s.M(p, s(c)) = M(p, c)$.*

For instance, if $C(p) = C_J$, then $M' = s.M$ is defined on p by:

$$\forall < c_1, \ldots, c_k > \in C_J, \quad M'(p, < s_{(1)}(c_1), \ldots, s_{(k)}(c_k) >) = M(p, < c_1, \ldots, c_k >)$$

Proposition 2.2 *The firing property is preserved by applying a permutation both on the markings and the transition instantiation. $\forall M$ ordinary marking, $\forall t \in T$, $\forall c \in C(t)$, $\forall s \in \xi$,*

$$M[t, c\rangle M' \iff s.M[t, s(c)\rangle s.M'$$

Definition 2.7 (Symbolic marking) *Let Eq be the equivalence relation defined by:*

$$M \ Eq \ M' \iff \exists s \in \xi, \ M' = s.M$$

An equivalence class of Eq is called a symbolic marking, denoted with $\mathcal{M}$.

Rather than considering a single initial marking M_0, we allow a WN to be initially marked by a symbolic marking $\mathcal{M}_0$. In this case, the WN no longer represents a single net, but a set of nets, each being initially marked by one of the markings contained in the equivalence class $\mathcal{M}_0$.

3 SRG

The symbolic reachability graph of a well-formed net is based on the idea of symmetry of objects of the basic color classes. It consists of a symbolic representation of all possible states of the model and the possibility of transition from one to the other. Of course its construction becomes algorithmically effective only in case of finite state space (bounded models).

In WNs it is possible to identify two basic kinds of symmetry: *rotation*, and *general permutation inside subsets*. Combinations of the two basic symmetry kinds can be found in actual models. The idea is to substitute the actual state representation with a symbolic representation that accounts for the symmetry properties holding in the model state space. A symbolic marking thus represents an equivalence class on the state space of the WN model, and the equivalence is in terms of the possible basic color permutations that yield the same behavior. In this section we first present an efficient algorithm for the computation of the SRG, whose application does not require any a-priori knowledge on the symmetries of the system modeled by the WN (i.e., the algorithm is based only on the syntax of the WN representation, and not on any sort of semantics of the specific model). Then we analyze the properties of the SRG that are relevant for the Markovian analysis of performance models based on SWNs.

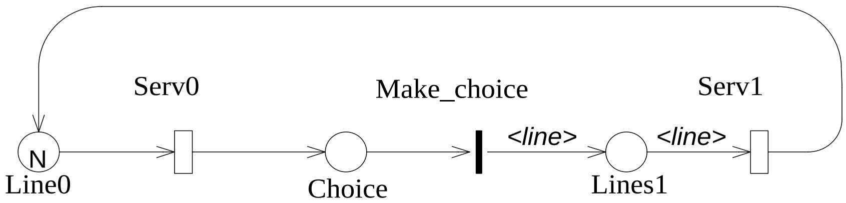

Figure 6: Model of two servers in tandem.

3.1 SRG Computation

We start by defining the *representation of symbolic markings* that, together with a *symbolic firing rule*, allows the construction of the symbolic reachability graph in an efficient algorithmic way.

3.1.1 Symbolic Marking

Equivalence classes of markings and firing instances The choice of a good representation of the data is always the first problem in the definition of an algorithm. Let us first present informally the idea of symbolic marking by considering the simple example of figure 6. It represents a closed system with two service stations in tandem. The first one is a single server station (place *Line0* transition *Serv0*), the second is a multiple server station (place *Lines1*, transition *Serv1(line)*) with four servers, each with a separate waiting line (basic color class $Lines = l[1-4]$). When a customer leaves the first station, it randomly chooses which line to join in the multiserver (place *Choice* and transition *Make choice(line)*). Consider the following marking:

$$Line0(1)Choice(1)Line1(2 < l_1 >, < l_2 >)$$

corresponding to the state with 1 customer in service on the first service station, 1 customer making the choice of a line to join in the multiserver 2 customers in the first line and 1 in the second line of the multiserver station. By looking at the possible behavior of the model the reader can check that a permutation symmetry exists in this case between the different objects of the basic color class. We can take an arbitrary permutation of the objects in basic color class *Lines* and obtain another legal state of the model with the same characteristics such as, e.g.

$$Line0(1)Choice(1)Line1(2 < l_4 >, < l_3 >)$$

The only relevant common characteristics of the two markings above is that one token is in place $Line0$, one token is in place $Choice$, while in $Line1$ there is an element with multiplicity 2 and a different element with multiplicity 1. From both these states only transition *Make choice(line)* can fire producing one out of three possible results: *a)* the customer represented by the token in place *Choice* chooses to join the line where two customers are already waiting, or *b)* the line where one customer is waiting, or *c)* one of the two remaining empty lines. Hence, also after the transition firing we can recognize a permutation symmetry, in which any permutation of objects in a given marking produces another valid marking with similar characteristics.

From the above informal reasoning it appears convenient to directly represent with an appropriate data structure the equivalence classes of markings based on the permutation and symmetry property of WN rather then the individual markings as usual in Petri nets. This can be achieved by abstracting from the actual identity of objects and retaining only enough information so that it is still possible to make an equality comparison between the new "symbolic objects."

Our first proposed abstraction consists of substituting object identifiers with variables. For example the "symbolic basic class" *Lines* would be defined[4] as $\{z_1, \ldots, z_4\}$. An example of symbolic marking would thus be the following:

$$\mathcal{M} = Line0(1)Choice(1)Line1(2 < z_1 >, < z_2 >)$$

To define the semantics of this symbolic marking we need the definition of *valid assignment*. An assignment of objects from a basic class to variables z_i is said to be valid iff the following three conditions are verified: *1)* every variable is assigned an object; *2)* the same object is not assigned to more than one variable; *3)* if the class is ordered, adjacent objects are assigned to subsequently numbered variables.

The symbolic marking $\mathcal{M}$ could thus represent the set of all ordinary markings that can be obtained from valid assignments of objects to the variables z_i. Observe that the symbolic marking

$$\mathcal{M}' = Line0(1)Choice(1)Line1(2 < z_2 >, < z_4 >)$$

represents the same set of ordinary markings represented by $\mathcal{M}$. This fact indicates the need for criteria that lead to a unique representation i.e., a *canonical representation*. We ignore this problem for the moment (we shall solve it later on) and we just consider $\mathcal{M}$ and $\mathcal{M}'$ as the same symbolic marking.

This symbolic marking actually implements the kind of abstraction that we were looking for, indeed the set of markings represented by the symbolic markings as defined above are exactly the equivalence classes identified by the equivalence relation Eq of Definition 2.7. Observe that it is rather natural to define a *symbolic firing rule* for this kind of symbolic marking since the variables play the same role that objects played in ordinary marking. Hence from marking $\mathcal{M}$ it is possible to fire the *symbolic transition instance* $[Makechoice, (line = z_2)]$ from which the new symbolic marking

$$\mathcal{M}'' = Line0(1)Line1(2 < z_1 >, 2 < z_2 >) \tag{1}$$

[4]A separate set of variables should be defined for each static subclass

is reached. The symbolic firing instance stands for all the ordinary instances that can be obtained by valid assignments of objects to variables. In Figure 7 a pictorial representation of the grouping induced by the symbolic marking and symbolic firing concept is shown.

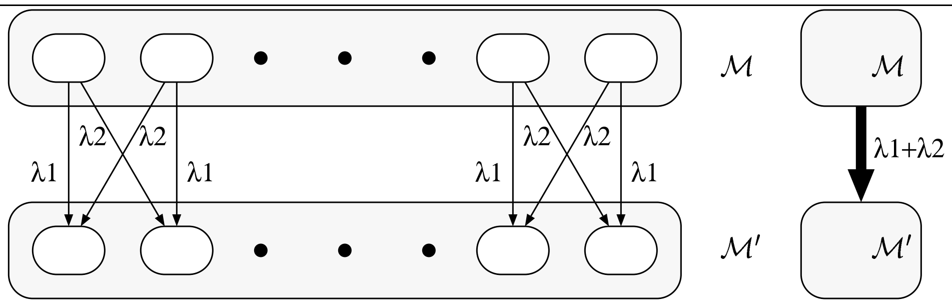

Figure 7: Grouping of ordinary markings and arcs

There is however a further step we can take to better exploit the grouping induced by the symbolic firing. It stems from the observation that firing either one of the two symbolic instances $[Makechoice, (line = z_3)]$ and $[Makechoice, (line = z_4)]$ form marking $\mathcal{M}$ we reach the same new symbolic marking (provided we use a canonical representation). In Figure 7 this situation is depicted by many arcs departing from the same ordinary marking within a symbolic marking and going to a group of ordinary markings all belonging to the same symbolic marking. It is possible to know in advance which are the symbolic instances that lead to the same new symbolic marking: indeed all those variables that have the same *distribution of tokens in the places* can be used interchangeably in a transition instance.

We then introduce the concept of *dynamic subclasses*, representing sets of objects that are not identified individually but that are known to be permutable one with the other in any firing instance to produce markings that belong to the same equivalence class. A dynamic subclass is characterized by its cardinality (i.e., the number of different objects represented by the dynamic subclass), and by the static subclass to which the represented objects belong (i.e., we can only group variables belonging to the same static subclass). In case of ordered basic classes, only contiguous objects can be represented by the same dynamic subclass and the ordering relation among objects is reflected by the ordering of the indexes of the dynamic subclasses.

Representation of symbolic markings: formal definition. The dynamic subclass concept affects both the symbolic marking representation and the symbolic firing. Using dynamic subclasses instead of variables in the marking representation allows a much more compact description of the marking itself. Moreover symbolic firing instances based on dynamic subclasses allow one to group many arcs connecting the same two symbolic markings.

Let us consider the marking $\mathcal{M}''$ defined in (1); the two variables z_1 and z_2 have the same distribution of tokens in the places (they both appear only in place $Line1$ with multiplicity 2), so that they can be grouped in the dynamic subclass Z^1_{Lines} of cardinality 2. Also the two remaining variables, z_3 and z_4 have the same distribution since they do not appear in any place, so that we can group them in the cardinality two dynamic subclass Z^2_{Lines}. The new symbolic marking representation can thus be written as

$$\mathcal{M}'' = Line0(1)Line1(2Z^1_{Lines}); \quad |Z^1_{Lines}| = |Z^2_{Lines}| = 2$$

More formally, the representation of the syntax and semantics of a symbolic marking can be defined as follows. Syntacticallly, a representation $\mathcal{R}$ of a symbolic marking $\mathcal{M}$ is a 4-tuple $\mathcal{R} = \langle m,\ card,\ d,\ mark \rangle$, where

- $m \ : \ I \rightarrow I\!N^+$, such that $m(i)$ (which will be also denoted m_i) is the number of dynamic subclasses of C_i in $\mathcal{M}$.
- The set of dynamic subclasses of C_i is denoted: $\hat{C}_i = \{Z^j_i \mid 0 < j \leq m_i\}$. We'll use the extended notation $\hat{C}(r), \quad r \in P \cup T$, to denote the set of all possible tuples of dynamic subclasses in a place/transition color domain.
- $card \ : \ \left(\bigcup_{i \in I} \hat{C}_i\right) \rightarrow I\!N$, such that $\forall i$, $\sum_{j=1}^{m_i} card(Z^j_i) = |C_i|$; i.e., the set of dynamic subclasses $\hat{C}_i$ forms a partition of C_i
- $d \ : \ \left(\bigcup_{i \in I} \hat{C}_i\right) \rightarrow I\!N$ such that $\forall Z^j_i$, $d(Z^j_i) = q \in [1..n_i]$ and $\forall 0 < i \leq n$, $\forall 0 < j < k \leq m_i$, $d(Z^j_i) \leq d(Z^k_i)$
- $\forall p \in P$, $mark(p) \ : \ \hat{C}(p) \rightarrow I\!N$

The associated semantics can be stated as:

- $\hat{C}_i$ is a non-instantiated partition of the basic class C_i
- Z^j_i represents any subset $C_{i,j}$ of C_i such that $|C_{i,j}| = card(Z^j_i)$
  in case C_i is ordered ($i > h$), the elements of $C_{i,j}$ are contiguous
- $\forall M \in \mathcal{R}, \quad \forall i \in I, \quad \exists \eta_i \ : \ C_i \rightarrow \hat{C}_i$
  (η_i is a *valid assignment* of subsets of C_i to dynamic subclasses Z^j_i);
- $\eta_i \ : \ C_i \rightarrow \hat{C}_i$ preserves
  - static subclass partitioning (function d)
  - cardinality (function $card$)
  - marking (function $mark$)
  - ordering relation in case of ordered classes (i.e. if $i > h$)

- $\eta_i^{-1}(Z_i^j) = C_{i,j}$ is the set of all possible instantiations (i.e., valid assignments) of a dynamic subclass that defines the set of markings $\{M : M \in \mathcal{R}\}$

When no ambiguity can arise, we denote the components of a representation simply by m, $card$, d, and $mark$. When referring to more than one representation, we remove the ambiguity by prefixing a proper representation identifier (e.g. $\mathcal{R}_1.m$).

Note that using the above representation without further constraints, one can find many representations for a given symbolic marking $\mathcal{M}$, as it happened with the former representation; we thus need to define a unique representation for a given symbolic marking.

Canonical representation of symbolic markings The first step in devising an efficient algorithm for the enumeration of the symbolic reachability graph of a WN is the definition of a *unique* representation for each symbolic marking $\mathcal{M}$. In order to obtain a unique representation we need a criteria to decide: *1)* how to partition the static subclasses (and hence the basic color classes) into dynamic subclasses of a given cardinality; *2)* how to name properly the dynamic subclasses.

The first problem is solved by defining a *minimality* property to be verified on the marking. The rationale behind the minimality requirement goes beyond the mere necessity of a canonical representation: indeed it also involves the maximization of both marking and firing instances grouping.
The second problem is solved by introducing the concept of *ordered* representation. The ordering criterion has to be defined in such a way that it is uniquely determined on some invariant characteristic of all possible minimal representations. We use lexicographic ordering of the minimal representation as such an invariant characteristics.

Both minimality and ordering definitions require the introduction of the *marking projection function* $mark_{sp}$.

Function $mark_{sp}$

This function is defined on the set of all possible tuples of dynamic subclasses from the Cartesian product of any subset of $\hat{C}$ (without repetitions of the same basic color class). For computing minimality the function is applied to single subclasses (i.e., tuples with arity one) while for computing ordering it is applied to tuples from $\hat{C}_I$. The formal definition of $mark_{sp}$ is rather cumbersome and not included here (it can be found in [15] or in [16]). Its meaning can be intuitively explained as follows. Given a symbolic marking $\mathcal{M}$, and a tuple of dynamic subclasses (possibly of arity one), function $mark_{sp}$ returns a vector of $|P|$ natural numbers encoding the *distribution of the tuple in the marking.*

Considering the symbolic marking $\mathcal{M}''$ used in the previous example,

$$mark_{sp}(Z_1^1)[Line0] = 1, mark_{sp}(Z_1^1)(Choice) = 0, mark_{sp}(Z_1^1)(Line1) = 2, (mark_{sp}(Z_1^1) =< 1, 0, 2 >)$$

$$mark_{sp}(Z_1^2)(Line0) = 1, mark_{sp}(Z_1^2)(Choice) = 0, mark_{sp}(Z_1^2)(Line1) = 0, (mark_{sp}(Z_1^2 =< 1, 0, 0 >)$$

Minimality

The minimality requirement refers to the number of dynamic subclasses in each basic class; intuitively we want to have the smallest possible number of dynamic subclasses since this maximizes the economy of both the marking representation and the number of possible firing instances.

Definition 3.1 (Minimality) *Let $\mathcal{M}$ be a symbolic marking. A representation $\mathcal{R}$ is minimal iff:*

$$\forall i \le h, \quad \forall j,k, \qquad j \neq k \implies \Big(mark_{sp}(Z_i^j) \neq mark_{sp}(Z_i^k)\Big) \vee \Big(d(Z_i^j) \neq d(Z_i^k)\Big)$$

$$\forall i > h, \quad \forall j,k, \qquad k = \oplus j \implies \Big(mark_{sp}(Z_i^j) \neq mark_{sp}(Z_i^k)\Big) \vee \Big(d(Z_i^j) \neq d(Z_i^k)\Big)$$

So the representation of $\mathcal{M}''$ is minimal since $mark_{sp}(Z_1^1)(Line1) \neq mark_{sp}(Z_1^2)(Line1)$.

Ordering

By properly ordering a minimal representation we can obtain a *canonical representation*. The re-ordering of a minimal symbolic marking representation consists of readjusting the dynamic subclasses indexes according to some univocal criteria.

Observe that if we restrict function $mark_{sp}$ to range over $\hat{C}_I$, we can represent $mark_{sp}$ as a n-dimensional matrix (one dimension for each basic class C_i with as many elements as $|\hat{C}_i|$). Hence, given a minimal symbolic marking, we can translate it into a matrix of this kind where the element $(i_1, \ldots, i_n)$ contains the place indexed vector returned by $mark_{sp}(Z_1^{i_1}, \ldots, Z_n^{i_n})$ (note that of course also the inverse translation is possible). Different minimal representations of the same symbolic marking translate to matrices that are identical up to a permutation of their elements. It is easy to define a canonical representation by chosing among all equivalent minimal representations, the one that corresponds to a matrix that is lexicographically ordered with respect to the following reading order of the matrix elements:

$$(1,1,\ldots,1),(2,1,\ldots,1),\ldots,(m(1),1,\ldots,1),(1,2,1,\ldots,1),\ldots \tag{2}$$

The complete algorithm for the computation of the canonical representation is described in [15] or in [16]. Intuitively the task accomplished by the algorithm is the following: it receives in input the matrix $mark_{sp}$ corresponding of a minimal representation $\mathcal{R}$ of a symbolic marking $\mathcal{M}$ and returns a set $\mathcal{S}$ of *symbolic permutations*[5]. The matrix obtained by application of any *symbolic permutation* in the set $\mathcal{S}$ is *lexicographically ordered.*

The algorithm can be informally described as follows: it executes a loop of at most $|\hat{C}_1| \ldots |\hat{C}_n|$ steps: let $(i_1, \ldots, i_n)$ be the index of a generic step (the indexes enumeration follows the order defined

[5]Symbolic permutations are defined as color permutations with dynamic subclasses replacing basic class objects

$\mathcal{S} := \hat{\xi}$; *(* The set* $\hat{\xi}$ *of all possible symbolic permutations is initially assigned to* $\mathcal{S}$ **)*
(restrict* $\mathcal{S}$ **)*
$< i_1, \ldots, i_n > = < 1, \ldots, 1 >$; *(* initialize the loop *)*
`while` $< i_1, \ldots, i_n > \neq < \mathcal{R}.m(1), \ldots, \mathcal{R}.m(n) >$ and $|\mathcal{S}| > 1$ `do`
    $min\_value = min_{s \in \mathcal{S}} mark_{sp}(s_1(i_1), \ldots, s_n(i_n))$
    $\mathcal{S} = \{s \in \mathcal{S} | mark_{sp}(s_1(i_1), \ldots, s_n(i_n)) = min\_value\}$
    `if` $|\mathcal{S}| > 1$
    `then` $< i_1, \ldots, i_n > = next(< i_1, \ldots, i_n >)$;
        (increment* $< i_1, \ldots, i_n >$ *according to the order* defined in eq. (2) **)*
    `fi`
`od`
$K(\mathcal{M}) := |\mathcal{S}|$

Figure 8: Sketch of the canonical representation computation algorithm

in 2); the result achieved in the generic step is the choice of the possible elements $\{ (j_1, \ldots, j_n) \}$ of the input matrix that may end up in position $(i_1, \ldots, i_n)$ in the ordered matrix. In other words this step constrains the possible values of $(s_1(i_1), \ldots s_n(i_n))$ to range over the set $\{ (j_1, \ldots, j_n) \}$. Obviously the choice at step $(i_1, \ldots, i_n)$ is in turn constrained by the choices performed in all the previous steps.

A sketch of the algorithm is given in Figure 8.

Note that if the representation of $\mathcal{M}$ input to the algorithm is minimal, then the cardinality $K(\mathcal{M})$ of the set $\mathcal{S}$ depends on the symbolic marking, but not on the chosen representation.

The computation of the canonical representation is one of the most critical parts of the SRG generation algorithm from a complexity point of view. Of course the actual complexity of a canonicalization operation depends on "how far" the input representation is from the canonical representation. The experiments we have done show that usually the representation obtained after firing a transition instance from a canonical representation is rather close to the ordered representation, so that the computational cost of canonicalization is often less than the worst case cost. The canonicalization of the initial marking $\mathcal{M}_0$ may instead have the highest cost since the initial representation $\mathcal{R}_0$ is given by the modeler.

3.1.2 Symbolic firing rule

In order to build the SRG directly starting from a symbolic initial marking $\mathcal{M}_0$ (i.e., without building the RG and then grouping markings into equivalence classes, which would be much easier but too costly), we first define a symbolic firing rule on the symbolic marking representations.

In a symbolic firing instance dynamic subclasses are assigned to the transition parameters instead of objects. The meaning is that any object in the subclass can be assigned to the parameter. When several type C_i parameters of t are assigned the same dynamic subclass Z_i^j we also need to specify whether the parameters are instanced to the same object or to different objects[6] of Z_i^j. So if we denote $param_i^x$ the x^{th} parameter of type C_i, in $C(t)$, then the parameter instance can be specified by a pair $\langle \lambda_i(x), \mu_i(x) \rangle = \langle j, k \rangle$ meaning that the parameter represents the k^{th} (arbitrarily chosen) element of Z_i^j. Of course k must be less than $|Z_i^j|$. Moreover if there are m parameters instanced to Z_i^j, k cannot be greater than m. More formally,

Definition 3.2 (Symbolic instance) *Let t be a transition with $C(t) = C_{(1)}, \ldots, C_{(k)}$; let e_i be the number of occurrences of C_i in $C(t)$ (i.e., the number of parameters of t whose type is C_i). Let $\mathcal{R}$ be a symbolic representation. A symbolic instance of t, denoted $[\lambda, \mu]$ is an instantiation of $\hat{C}(t)$ for $\mathcal{R}$ defined by:*

$$\lambda = \{\lambda_i : \{1, \ldots, e_i\} \to I\!N^+\}, \qquad \mu = \{\mu_i : \{1, \ldots, e_i\} \to I\!N^+\}, \quad \text{such that } \forall i \in I,\ \forall 0 < x \leq e_i,$$

- $\lambda_i(x) \leq \mathcal{R}.m(i)$,
- $\mu_i(x) \leq \mathcal{R}.card(\mathcal{R}.Z_i^{\lambda_i(x)})$
- *if* $i \leq h$ *then* $\forall 0 < k < \mu_i(x),\ \exists x' < x$ *such that* $\lambda_i(x') = \lambda_i(x) \wedge \mu_i(x') = k$.

$\lambda_i(x)$ is used to choose the subclasses of $\hat{C}_i$ to be instantiated. In case of non ordered classes C_i, $\mu_i(x)$ is used to distinguish already instantiated elements from the other elements within the subclass selected by $\lambda_i(x)$. The additional conditions on $\mu_i(x)$ guarantee that the functions λ and μ define a partition of the ordinary arcs of the reachability graph in symbolic arcs (an ordinary arc cannot satisfy two different pairs $[\lambda, \mu]$ simultaneously).
We denote $\mu_i^j = \sup(\mu_i(x) \mid \lambda_i(x) = j)$ the number of different instantiations in the dynamic subclass Z_i^j. For any Z_i^j that is not instantiated (i.e. such that $\not\exists x : \lambda_i(x) = j$), then by definition $\mu_i^j = 0$.

We now introduce the notion of *split symbolic marking*. As mentioned before, when we assign a dynamic subclass to a transition parameter we mean that any object of the subclass may be selected for assignment to the parameter. It doesn't actually matter *which* object is chosen since they all behave the same way. Now in order to define the symbolic enabling and firing rules, a splitting is made in each subclass between the (arbitrarily chosen) objects that will be selected for the firing and the other objects. For ordered classes we always split the instanced dynamic subclasses into a set of new cardinality 1 dynamic subclasses.

Definition 3.3 (Splitting) *Let $\mathcal{R}$ be a representation of a symbolic marking $\mathcal{M}$. Then $\mathcal{R}_s = \mathcal{R}[\lambda, \mu]$ is defined by the following transformations on $\mathcal{R}$:* $\quad \mathcal{R}_s.\hat{C}_i = \{Z_i^{j,k}\}$ *with $Z_i^{j,k}$ defined as follows:*

Case 1: Non-ordered Class $C_i,\ i \leq h$

For each Z_i^j $s.t.$ $\exists x : \lambda_i(x) = j$ do

[6] Observe that we can instance at most as many different objects as the cardinality of the dynamic subclass

$split\_subclasses = \{Z_i^{j,k} | \exists x : \langle \lambda_i(x), \mu_i(x) \rangle = k\}$ *(selected objects subclasses)*

if $\mu_i^j < \mathcal{R}.card(Z_i^j)$

then $split\_subclasses = split\_subclasses \cup \{Z_i^{j,0}\}$ *(subclass of remaining objects)*

$\mathcal{R}_s.\hat{C}_i = \mathcal{R}.\hat{C}_i \ - \ \{Z_i^j\} \cup split\_subclasses$

$\mathcal{R}_s.card(Z_i^{j,k}) =$ if $k > 0$ then 1 *else* $(\mathcal{R}.card(Z_i^j) - \mu_i^j)$

od

Case 2: Ordered Class $C_i, \ i > h$

For each Z_i^j $s.t.$ $\exists x : \lambda_i(x) = j$ do

$split\_subclasses = \{Z_i^{j,k}, 0 < k < \mathcal{R}.card(Z_i^j) \ \}$

$\mathcal{R}_s.card(Z_i^{j,k}) = 1$

od

Concerning the fourth component of $\mathcal{R}_s$ i.e., $\mathcal{R}_s.mark$, it can naturally derived from $\mathcal{R}.mark$: observe that using the sum and Cartesian product operations on multisets defined in 2.1 we can write:

$$Z_i^j = Z_i^{j,0} + Z_i^{j,1} + \ldots + Z_i^{j,k_i}$$

so that the tuple $< Z_1^2, Z_2^3 >$ after the splitting of Z_1^2 into $Z_1^{2,0} + Z_1^{2,1} + Z_1^{2,2}$ and the splitting of Z_2^3 into $Z_2^{3,1} + Z_2^{3,2}$ can be rewritten as

$$< Z_1^{2,0} + Z_1^{2,1} + Z_1^{2,2}, Z_2^{3,1} + Z_2^{3,2} >$$

Since the notation $< a, b >$ (where a and b are multisets) denotes the Cartesian product of a and b, the above tuple can be transformed into

$$< Z_1^{2,0}, Z_2^{3,1} > + < Z_1^{2,1}, Z_2^{3,1} > + < Z_1^{2,2}, Z_2^{3,1} > + < Z_1^{2,0}, Z_2^{3,2} > + < Z_1^{2,1}, Z_2^{3,2} > + < Z_1^{2,2}, Z_2^{3,2} >$$

Proposition 3.1 *Let $\mathcal{R}$ be a representation of the symbolic marking $\mathcal{M}$. Then $\mathcal{R}_s = \mathcal{R}[\lambda, \mu]$ is another representation of $\mathcal{M}$.*

With such a definition of the split marking, subclasses can substitute objects in the transition firing.

The evaluation of arc expressions and predicates does not change when dynamic subclasses of cardinality one replace objects in variable assignments. We use the notation $\hat{W}$ and $\hat{\Phi}(t)$ to indicate the symbolic arc expressions and transition predicates respectively.

The four step symbolic firing. The canonical representation of the symbolic marking obtained by firing (t, λ, μ) in $\mathcal{M}$ (i.e., $\mathcal{M}' = \mathcal{M}[t, \lambda, \mu\rangle$) is computed in four steps, that use different intermediate (non canonical) representations.

1. Splitting $\mathcal{M}$ with respect to $[\lambda, \mu]$

   let $\mathcal{R}_s = \mathcal{R}[\lambda, \mu]$ be the split representation of $\mathcal{M}$ in which (t, λ, μ) is enabled;

2. Actual Firing $\mathcal{M}' = \mathcal{M}[t, \lambda, \mu\rangle$

   Define $\mathcal{R}_f$ by copying the components m, $card$, and d from $\mathcal{R}_s$, and computing $\mathcal{R}_f.mark$ by applying the incidence functions on $\mathcal{R}_s.mark$.

$$\forall p \in P, \qquad \mathcal{R}_f.mark(p) = \mathcal{R}_s.mark(p) - \hat{W}^-(p,t)(\lambda,\mu) + \hat{W}^+(p,t)(\lambda,\mu)$$

   $\mathcal{R}_f$ is a (possibly non canonical) representation of $\mathcal{M}'$;

3. Grouping $\mathcal{M}'$

   Compute a minimal representation $\mathcal{R}_m$ of $\mathcal{M}'$ by grouping dynamic subclasses of $\mathcal{R}_f$;

4. Ordering $\mathcal{M}'$

   Compute the canonical representation of $\mathcal{M}'$ by transforming $\mathcal{R}_m$ into an ordered representation. This is obtained by applying one of the permutations in S to the dynamic subclasses of $\mathcal{R}_m$.

Let us make a final remark on the splitting of ordered classes. The complete splitting into dynamic subclasses of cardinality 1 is not strictly necessary, but it is convenient since it allows a homogeneous treatment of the different cases that arise considering all the different positions occupied by the selected element in the instantiated dynamic subclass. The reason for considering each case separately is due to the need to maintain the ordering among the dynamic subclasses of an ordered class throughout the algorithm.

Example of Symbolic Firing. Let us consider a very simple artificial example in order to illustrate the four steps of the symbolic firing rule. The different steps are illustrated in graphical form in Figure 9. The WN considered in this example has no particular meaning. The top-left portion of the figure depicts the canonical representation of an arbitrary marking $\mathcal{M}$. The other portions represent the four steps of the symbolic firing of transition t for $\langle Z_1^2, Z_2^1\rangle$ (which is enabled in $\mathcal{M}$, as the reader can check).

The Symbolic firing rule can be easily cast into the usual algorithm structure for the computation of the Reachability Graph of a Petri net provided that the Symbolic canonical representation is used to store the markings of the reachability set.

3.2 SRG relevant properties

We present now the most interesting properties of the symbolic reachability graph that can be exploited for a performance evaluation of WN models. Other interesting properties can be shown concerning the qualitative behavior of WN models based on the analysis of the SRG, but they are not reported here for the sake of conciseness.

The first two properties that we consider establish the equivalence between the RG and the SRG from the point of view of the reachability of markings.

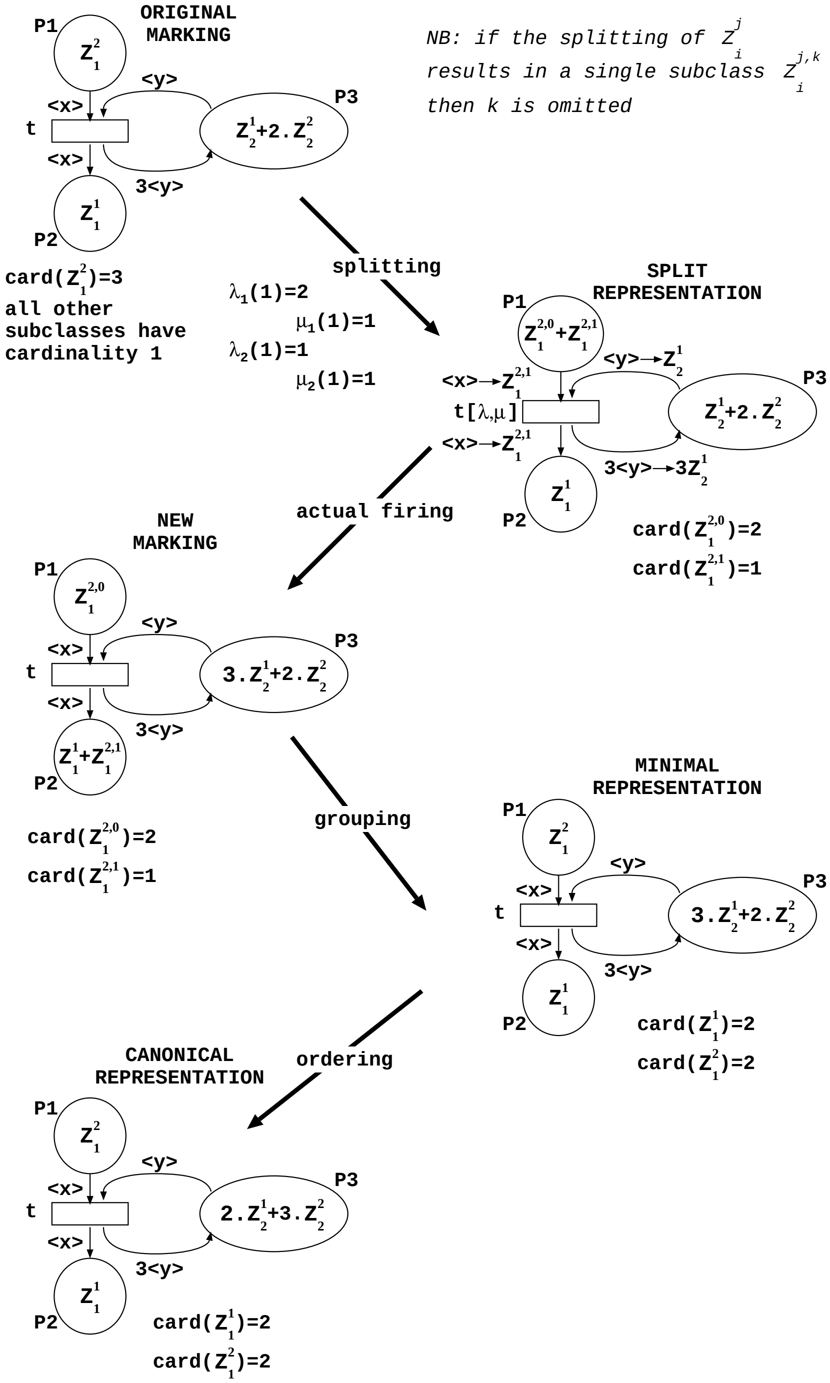

Figure 9: An example of symbolic firing

Property 3.1 *Equivalence between symbolic and ordinary reachability.*

Let $\mathcal{M}_0$ be a symbolic marking, and $[\mathcal{M}_0\rangle$ be the set of symbolic markings reachable from $\mathcal{M}_0$. Then

$$\bigcup_{M_0 \in \mathcal{M}_0} [M_0\rangle \quad = \quad [\mathcal{M}_0\rangle$$

Property 3.2 *Cardinality of a symbolic marking.*

Let $\mathcal{M}$ be a symbolic marking and $|\mathcal{M}|$ be the number of ordinary markings belonging to the equivalence class of $\mathcal{M}$.

Let $K(\mathcal{M})$ be the number of permutations computed during the ordering phase of the symbolic firing algorithm. Then

$$|\mathcal{M}| \quad = \quad \frac{1}{K(\mathcal{M})} \left(\prod_{i=1}^{h} \prod_{q=1}^{n_i} \frac{|D_{i,q}|!}{\prod_{d(\mathcal{M}.Z_i^j)=q} card(\mathcal{M}.Z_i^j)!} \right) \prod_{i=h+1}^{n} \nu(i)$$

where $\nu(i) =$ if $(\mathcal{M}.m(i) > 1 \wedge n_i = 1)$ then $|C_i|$ else 1.

The above properties ensure that no information on reachability is lost by analyzing the SRG instead of the RG. However, in order to derive an improved technique for performance evaluation based on the SRG instead of the RG, we still need to know how to test the ergodicity of the Markov chain and how to compute its transition rates. These additional requirements lead to the formulation of the following additional propositions[7].

Property 3.3 *Necessary condition for ergodicity.*

Strong connection of RG $\Longrightarrow$ Strong connection of SRG, but not vice-versa.

In [15] an example is given in which the SRG is strongly connected while the underlying RG is not.

Property 3.4 *A sufficient condition for ergodicity.*

Strong connection of SRG

$\wedge \quad \forall 0 < i \leq h, \ \ \exists \mathcal{M}' \in SRG$ *such that* $(\mathcal{M}'.m(i) = n_i)$

$\wedge \quad \forall h < i \leq n, \ \ ((n_i > 1) \ \vee \ (\exists \mathcal{M}' \in SRG$ *such that* $(\mathcal{M}'.m(i) = 1) \ )$

$\Longrightarrow$ *Strong connection of RG*

The above property is an extension of a property already proven in [8] for the restricted case of Regular nets[8]. On the other hand, the condition is not necessary in order for the RG to be strongly connected, even in the simpler case of RNs. Indeed, in [15] an example of WN is given in which the SRG is strongly connected, dynamic subclasses are never completely grouped, and yet the RG is strongly connected.

[7]Remember that a finite, continuous time Markov chain is ergodic iff its graph representation is strongly connected.

[8]Indeed, the additional condition on the dynamic subclass partition is always satisfied by the (symmetric) initial marking in an RN.

Property 3.5 *Equivalence between symbolic firing and ordinary firing.*
Let $\mathcal{M}$ and $\mathcal{M}'$ be two symbolic markings of the SRG, and $M \in \mathcal{M}$ be an ordinary marking of the RG. Let $A_{M,\mathcal{M}'}$ be the set of arcs of the RG connecting M to any marking $M' \in \mathcal{M}'$, and $A_{\mathcal{M},\mathcal{M}'}$ be the set of symbolic arcs of the SRG connecting $\mathcal{M}$ to $\mathcal{M}'$.
Then there exists a mapping ω from $A_{M,\mathcal{M}'}$ onto $A_{\mathcal{M},\mathcal{M}'}$ such that:

- *if the label of an arc $a \in A_{M,\mathcal{M}'}$ is $[t, < c_{i_1}^{j_1}, \ldots, c_{i_k}^{j_k} >]$ then the label of $\omega(a)$ is $[t, \lambda, \mu]$ with $c_i^j \in D_{i,q} \Longleftrightarrow \mathcal{M}.d(Z_i^{\lambda_i(j)}) = q$*
- *if the label of a symbolic arc $a \in A_{\mathcal{M},\mathcal{M}'}$ is $[t, \lambda, \mu]$ then the cardinality of the reciprocal image of a denoted $|\omega^{-1}(a)|$ is*
$$\prod_{i=1}^{h} \prod_{j=1}^{m_i} \frac{card(Z_i^j)!}{(card(Z_i^j) - \mu_i^j)!}$$
where $\mu_i^k = \sup_{x \,:\, \lambda_i(x)=k} \mu_i(x)$

4 Stochastic WNs and lumped Markov chains

Stochastic Regular nets where already defined in [10]. The basic principle for timing a colored Petri net is to associate a function from the markings to positive real numbers to each arc of the reachability graph. In this way a discrete-state semi-Markov process is defined whose state space is corresponding to the reachability set of the colored net. In this section, after a formal definition of stochastic well-formed colored nets (SWN), we prove that an aggregate Markovian process can be defined based on the SRG in order to compute the same performance estimates with a lower computational cost compared to the usual technique based on the RG.

4.1 Stochastic well-formed nets

In WNs a priority structure is defined on transitions. This priority is reflected in the timing semantics of SWNs in the same way as was originally defined for GSPNs [19]. Transitions with priority level 0 are called *timed* transitions, and they fire at the instant of the elapsing of a delay from the instant of the transition enabling. The delay is determined for each instantiation of the enabling of a transition according to a random process with negative exponential probability distribution. Transitions with priority level greater than 0 are called *immediate* transitions, and they fire in zero time at the instant of their enabling. In case of conflicting immediate transitions a firing probability is assigned to each conflicting transition, proportionally to a weight. The probability is computed by normalizing the weights of all conflicting transitions enabled in the same marking.

In order to guarantee the presence of symmetry not only from a logical but also from a stochastic point of view, we restrict the possibility of marking dependency for the mean values of transition firing delays. The average firing rate of a transition instance can be a function of the static subclass that the objects assigned to the transition parameters belong to, and not a function of the assigned objects themselves. In this way all objects of a given static subclass determine the same transition firing delay. This can be formalized by introducing the notation

$$\tilde{C}_i = \{ \; D_{i,1}, \ldots, D_{i,q} \; \}$$

In analogy with the notation introduced for the representation of symbolic markings and dynamic subclasses, given a transition t with color domain $C(t) = C_{J(t)}$ we define

$$\tilde{C}(t) \;\; = \;\; \{< D_{i_1,j_1}, \ldots, D_{i_k,j_k} > \; | \;\; 0 < j_l \leq n_{i_l} \; \}$$

For any $c =< c_{i_1}^{j_1}, \ldots, c_{i_k}^{j_k} > \in C(t)$ we also define $\tilde{c} =< \tilde{c}_{i_1}^{j_1}, \ldots, \tilde{c}_{i_k}^{j_k} > \;\; \tilde{C}(t)$ such that $\tilde{c}_i^j = D_{i,q}$ iff $c_i^j \in D_{i,q}$. Finally we define the *static partition of a marking* M denoted $\tilde{M}(p) \in Bag(\tilde{C}(p))$ as follows:

$$\tilde{M}(p)(\tilde{c}) \;\; = \sum_{c' \; : \; \tilde{c}' = \tilde{c}} M(p)(c')$$

The static partition of a marking represents, for each place and for each Cartesian product of static subclasses, the number of tuples in the place that belong to the same Cartesian product of static subclasses.

Property 4.1 *Static partition of symbolic markings.*

$$\forall M, \;\; M' \in \mathcal{M}, \quad \forall p \in P, \quad \tilde{M}(p) = \tilde{M}'(p)$$

Hence we can define the static partition of a symbolic marking as well and denote it $\tilde{\mathcal{M}}$.

Definition 4.1 (SWN) *A stochastic well-formed colored net is a pair* $SWN = \langle WN, \theta \rangle$ *such that*

WN *is a well-formed colored Petri net*

θ *is a function defined on the set of transitions* T *such that*

$$\theta(t) \quad : \quad \tilde{C}(t) \times \;\; Bag(\tilde{C}(p_1)) \times Bag(\tilde{C}(p_2)) \times \ldots Bag(\tilde{C}(p_{|P|})) \;\; \longrightarrow \;\; I\!R^+$$

For any timed transition t, the function $\theta(t)(\tilde{c}, \tilde{M})$ represents the average firing rate for any instance of transition $[t, c]$ enabled in marking M. In case of immediate transitions, the same function is interpreted as the weight to be normalized within a conflict set in order to obtain the firing probability:

$$\frac{\theta(t)(\tilde{c}, \tilde{M})}{\sum_{M[t',c'\rangle} \theta(t')(\tilde{c}', \tilde{M})}$$

4.2 SRG based lumped process

We present now an algorithm that exploits the partition in equivalence classes of markings implicitly determined by the computation of the SRG of an SWN in order to reduce the cost of the numerical Markovian analysis.

We denote by SRS the set of all reachable symbolic markings $\mathcal{M}_i$ of the WN. An actual marking of the equivalence class defined by $\mathcal{M}_i$ is denoted here with a double index $M_{i,k}$, where k represents an internal ordering within the equivalence class. The weight function θ is extended to symbolic subclasses as follows: $\forall \mathcal{M}$, $\forall [\lambda, \mu]$, $\forall M \in \mathcal{M}$, $\forall t \in T$, $\forall c \in C(t)$ such that $\eta_i(c_i^j) = Z_i^{\lambda_i(j)}$, $\Theta(t)(\lambda, \mu, \tilde{\mathcal{M}}) = \theta(t)(\tilde{c}, \tilde{M})$.

The performance evaluation algorithm can thus be outlined as follows:

1. Construction of the aggregate state space description
   - construction of the SRG of the WN
   - elimination of self-loop arcs $(\hat{W}^+(p,t)(\lambda,\mu) - \hat{W}^-(p,t)(\lambda,\mu) = 0)$
   - for each arc of the SRG, computation of $\Theta(t)(\lambda, \mu, \tilde{\mathcal{M}})$
2. Construction of the square matrix $\mathcal{Q} = [q_{i,j}]$ of dimension $N' = |SRS|$ with $q_{i,i} = -\sum_{j \neq i} q_{i,j}$

$$\text{and } \forall j \ : \ i \neq j, \quad q_{i,j} = \frac{\sum_{[t,\lambda,\mu] \ : \ \mathcal{M}_i[t,\lambda,\mu\rangle \mathcal{M}_j} \Theta(t)(\lambda, \mu, \tilde{\mathcal{M}}_i).|\eta^{-1}(t,\lambda,\mu)|}{\sum_{[t',\lambda',\mu'] \ : \ \mathcal{M}_i[t',\lambda',\mu'\rangle} \Theta(t')(\lambda', \mu', \tilde{\mathcal{M}}_i).|\eta^{-1}(t',\lambda',\mu')|}$$

3. Numerical solution of the Semi-Markov process described by $\mathcal{Q}$, (the solution is unique if the SRG contains a single strongly connected component) and computation of the steady-state probability distribution $\Psi$for tangible symbolic markings.

From the probability distribution Ψ of the tangible symbolic markings it is always possible, if needed, to compute the probability distribution ψ of the tangible actual markings of the SWN as:

$$\forall M_j \in \mathcal{M}_i, \qquad \psi_j = \frac{\Psi_i}{|\mathcal{M}_i|}$$

Note that usually the actual marking distribution is not needed for the computation of performance indexes defined at the net level and that in any case the complexity of the above outlined numerical analysis is polynomial in the cardinality of the SRG instead of the cardinality of the RG. Thus, the proposed techniques exploits at its greatest extent the color structure of the model in order to reduce the size of the state space of the stochastic process to be analyzed.

The prove that the analysis of the stochastic process defined on the SRG yields the same steady-state solution that can be computed from the general technique based on the RG is divided in three steps.

1. The embedded Markov chain is lumpable. Hence it is possible to derive a reduced linear system whose variables are the symbolic marking probabilities.

2. The coefficients of the reduced linear system can be computed directly from the SRG.

3. All actual markings within a symbolic marking have the same probability.

See [20] for an outline of the formal proof.

5 A Complex Application to Multiprocessor Architectures

In this section we apply the WN formalism to the modeling and performance evaluation of a multiprocessor system already studied in [23, 2, 5]. The reasons for reconsidering this almost classic architecture are many. First, it is intrinsically more complex and more realistic than other systems already studied by colored or high-level Petri nets, such as, e.g., [7, 11]. Second, the solutions based on Markov or Stochastic Petri net techniques presented in [23, 2] are not satisfactory due to their inherent complexity in terms of increasing of the number of states as a function of the number of processors considered in the system, that limited the availability of results to very few processors (≤ 3). Third, the solution based on folded GSPN model presented in [5], although much better from the computational complexity point of view was still not satisfactory from a modeling point of view due to the complexity of the GSPN model itself, whose construction required a great deal of ingenuity and a deep understanding of the behavioral symmetries of the system.

The aim of this example is twofold. On one hand we show the considerable difference in size of the (aggregated) Markov chain directly obtained from the SRG with respect to the size of the Markov chain obtained from the detailed RG, which is equivalent to those presented in [23, 2]. On the other hand we compare the aggregations automatically performed in the SRG generation phase with those devised in [5] at the price of a thorough (human) analysis of the system behavior and find that the same level of lumping is now achieved without any effort on the part of the modeler. Indeed the gain in space enlarges the class of numerically solvable models.

The multiprocessor architecture analyzed in this example is composed by a set of processors connected through a bus. Each processor p_i is associated with a local memory composed of two sections, a private one (PM_i) and a common one (CM_i). Private memory areas (PM) can be accessed only by the corresponding processor through its private bus (PB); private memory accesses never generate contention. Common memory areas (CM) are accessible to all the processors in the system. Accesses to the local module of the CM are performed through the private bus plus the local bus (LB), while accesses to non local CM modules are performed using the global bus (GB) plus the local bus of the destination CM module.

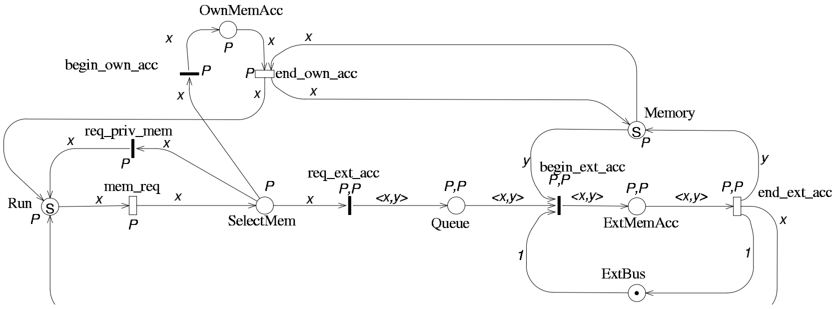

Figure 10: An intuitive WN model of the multiprocessor architecture

Contention arises in the use of GB as well as of the local busses (and the CM modules). A processor is delayed when some of the resources it is trying to access are busy. We assume however that external access requests to CM modules have *priority* over the local CM accesses and cause their *preemption*.

The overall behavior of the system can be described as follows: processors alternate periods of processing requiring only access to the private memory (we call these periods CPU bursts), with periods of CM modules accesses. For simplicity we assume that the system is made up of n identical processor-memory modules. In order to build the WN model of the system it can be useful to classify the possible states of a processor as follows:

Active: Processor executing in its private memory;

Accessing local CM: Processor performing a local CM module access;

Accessing remote CM: Processor performing an external CM module access;

Queued: Waiting for the GB to become available;

Blocked: Waiting to continue a local CM access preempted by an external access.

The behavior of the system is described in a straightforward manner by the WN model in Figure 10. Let us note that there is an additional guard $[x \neq y]$ on the transition 'req_ext_acc'. Places represent the possible states of each processor as follows. 'Run' contains tokens whose color represent the identity of processors in the "Active" state. Similarly, places 'ExtMemAcc' and 'Queue' represent processors in the "remote access" and in the "Queued" states, respectively. Place 'OwnMemAcc' can represent either state "local access" or "Blocked", depending on the fact that a token of the same color is or is not present in place 'Memory'. A probabilistic choice among private, local, or external memory access is modeled by the three conflicting immediate transitions 'req_priv_mem', 'begin_own_acc', and 'req_ext_acc'. In the latter case, the choice of the external memory is represented by the choice of the second component (y) of the color domain of transition 'req_ext_mem' (which is enabled for any $y \neq x$).

This WN representation of the system, although correct and quite easy to understand, is not the simplest one can draw. For example, the three conflicting immediate transitions can be "agglomerated" into their preceding timed transition 'mem_req' by applying a well known net reduction rule that

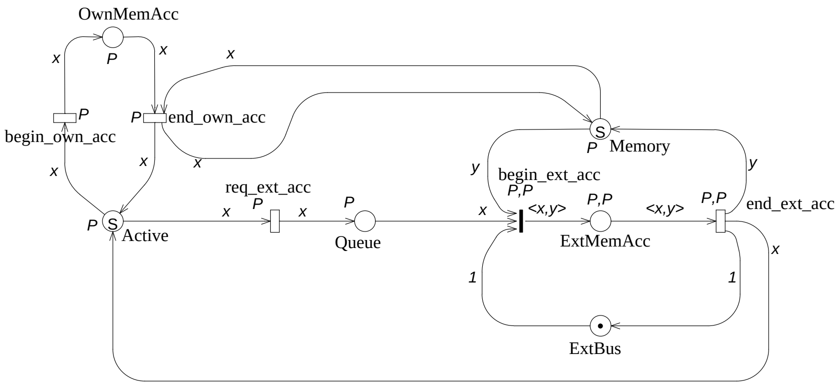

Figure 11: A more compact WN model of the multiprocessor architecture

preserve all behavioral properties of the net [24] as well as the timing semantics of the model [25]. Moreover, the timed transition resulting from the fusion of transitions 'mem_req' and 'req_priv_mem' in the WN in Figure 10 can be deleted since its firing determines no change in the marking of the net. Finally, the WN representation can be further simplified by "delaying" the choice of the external memory to be accessed until the global bus is available: indeed, the information about the identity of the memory module to be accessed is useless before the GB access is granted to the processor.

By applying all the above discussed simplifications, one can finally draw the more compact WN model depicted in Figure 11. Let us note that there is an additional guard $[x \neq y]$ on the transition 'begin_ext_acc'. The Basic colour class P is introduced to represent module identities (both processor and associated local CM identities). For example place *Active* with colour domain P contains tokens representing the active processors; place *ExtMemAcc* with colour domain P, P contains pairs $\langle processor, CMmodule \rangle$ representing the external accesses currently going on[9]; place *Memory* with domain P contains tokens corresponding to the CM modules not used by any external processor; place *OwnMemAcc* with domain P contains both processors in *Blocked* and in *Access local CM* states, the former (latter) being characterized by the absence (presence) of the corresponding coloured token in place *Memory*; place *ExtBus* has no associated colour domain (has neutral domain) and describes the GB state: empty for GB busy, filled with one token for GB idle.

Transitions *begin_own_acc*, *end_own_acc*, and *req_ext_acc*, have colour domain P. Transition *begin_ext_acc* has domain P, P and has an associated predicate.[10]

Transition *begin_ext_acc* is immediate because we assume that both arbitration and release time of busses are negligible (and thus set to zero). Furthermore we have assumed that both CPU bursts and CM access periods are independent, exponentially distributed, random variables. Finally, external access requests from each processor are directed to any non local CM module with probability $\frac{1}{n-1}$; this is represented in the model by stating that all possible (conflicting) instantiations of transition

[9] Of course, due to the availability of a single GB, there cannot exist more than one token at a time in this place.

[10] This transition represents the beginning of an *external* access of processor x to the CM module y. Because of the predicate only firing instances with $x \neq y$ are enabled. Indeed $x = y$ would represent a *local* access. The selection of a request from the external access queue has been modeled assuming a random order queueing discipline.

n	# tang. SRG	# tang. RG	$E(AP)/n$	$U(GB)$
2	6	10	0.6752411	0.27009645
3	13	62	0.6600941	0.39605642
4	23	340	0.642929	0.51434340
5	36	1652	0.6227463	0.62274684
6	52	7354	0.5991253	0.71895120
7	71	30746	0.5719982	0.80079870
8	93	122728	0.5417373	0.86678098
9	118	472904	0.5092124	0.91658070
10	146	1772494	0.475696106	0.95139024

Table 1: Results

begin_ext_acc have equal probability. The parameters of our model are the number n of processor-memory modules, the average CPU burst length $(1/\lambda)$, and the average external accesses duration $(1/\mu)$. As in [23, 2, 5], we define the system load factor ρ as the ratio λ/μ. Some performance figures that can be obtained from our model are: the average number of active processors ($E[AP]$); the GB utilization ($U[GB]$). These performance figures can be computed from the steady state probability of the symbolic markings using the following formulas:

- $E[AP] \;=\; \sum_{\mathcal{M}\in tang(SRG)} \Psi(\mathcal{M})\ \#Active$
  where $\Psi(\mathcal{M})$ is the steady state probability of the symbolic marking $\mathcal{M}$, while $\#Active$ is the number of elements in place 'Active' defined as $\#Active = \sum_{<Z_1^j>\in\mathcal{M}(Active)} card(Z_1^j)$

- $U[GB] \;=\; \sum_{\mathcal{M}\in tang(SRG):\mathcal{M}(GB)=0} \Psi(\mathcal{M})$

In Table 1 the performance figures computed for different values of the number of processors assuming a load factor $\rho = 0.2$ are shown. The mean number of active processors is here divided by n in order to obtain a normalized version of this figure.

In Table 1 the size of the aggregate and complete Markov chains are also reported for different values of n, to give a flavor of the considerable gain that can be achieved with this method.

Now let us compare the automatic aggregations performed by the SRG generation algorithm with those devised in [5]. In that paper a non colored GSPN model was built having already in mind the aggregated state information required. That is, the redundant information of the original (much simpler and easy to build) model was discovered by carefully studying the system behavior and eliminated by changing the kind of state encoding at the net level. It is interesting to observe that the same redundancies are automatically detected on the WN model by the SRG generation algorithm as can be observed comparing the tangible SRG of the WN model and the tangible RG of the uncolored

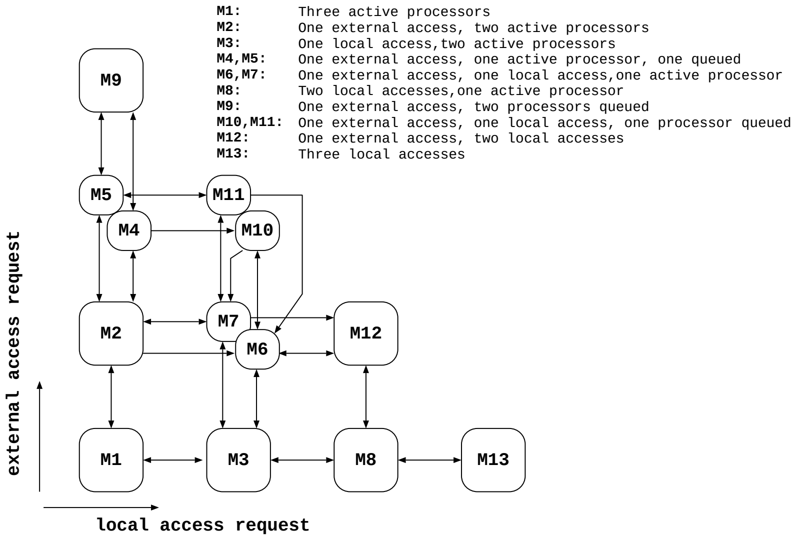

Figure 12: SRG of the multiprocessor WN

model shown in Figure 12. Notice that the construction of the WN description, as compared to the folded GSPN model originally presented in [5], although equivalent from the point of view of efficiency of the analysis requires a substantially smaller intellectual effort to be understood and to be devised starting from the description of the system behaviour.

6 Conclusions

Stochastic Petri nets were introduced some time ago as a good modeling tool for the performance evaluation of multiprocessor computer systems. They offered a much higher level tool than previously used Markov chains, but they suffered from the same large dimension problem that often hampers the possibility of obtaining results for realistic size systems. High-level Petri net formalisms have already been employed to further reduce the cost of the model definition and validation, and sometimes also to perform lumpings of the underlying Markovian models in order to reduce the cost of the performance evaluation for larger system configurations. So far, however, it was the total responsibility of the modeler to identify such system characteristics as symmetry in behavior, in order to exploit the characteristics to obtain models that are less complex to analyze.

For the first time a completely algorithmic approach is proposed that allows the exploitation of these model characteristics for the construction of lumped Markov chains. The user need not be aware of the lumping technique, as long as he specifies the behavior of the system in terms of well-formed colored nets. The intrinsic symmetries are automatically detected and used at their greatest extent by means of the construction of the *symbolic reachability graph*. The technique presented in this paper is a direct extension of that already proposed for a restricted class of "regular nets". The novelty of the result presented here is that, with the extension of the SRG technique to well-formed colored nets, the technique is now applicable to any colored model.

Of course the SRG algorithm presented here fails in producing a smaller description than the usual RG if the model does not exhibit useful symmetries as defined in section 3. This situation may happen in two cases: if the system to be modelled does not contain any symmetry, or if the model does not take the intrinsic symmetries of the system into account properly. It is interesting to note that the syntax for the definition of SWN models encourages the modeller to group objects into basic colour classes and to avoid splitting them in static subclasses if this is not necessary, since this way the textual part of the model description is smaller and faster to produce. Thus models that take all symmetries into account are more likely to be produced than equivalent models that do not. In this sense the modeller implicitly gives to the SRG construction algorithm the best of his knowledge about system symmetries by simply trying to reduce his model description effort.

We have presented an example of use of stochastic well-formed nets for the study of a non trivial multiprocessor architecture. Another complex example of modelling a concurrent algorithm using this formalism can be found in [26]. These examples show how the technique overcomes the difficulties found in the use of other performance modeling techniques. On the one hand the model is easier to devise in terms of the WN formalism, and fairly easy to understand even by non Petri net experts. On the other hand, the exploitation of the SRG construction algorithm allows an evaluation whose cost is comparable to the analysis cost of thoroughly designed GSPN models where all crucial behavioral symmetries of the system have been fully identified by the modeler himself.

In conclusion we notice that the performance analysis by construction of a Markov chain from the RG of a Petri net model is in practice most often hampered by the large size of the graph caused by two different reasons. The first reason is the representation of the parallelism in terms of all possible interleavings that can be observed on the global state of the model. The second reason is the possible presence of intrinsic symmetries of the model that produce many different portions of a graph representing essentially the same behavior. In this paper we have addressed and solved the second problem. The first problem is inherent to the state-space representation, and in our opinion can be avoided only by using non state-space based analysis techniques.

The initial simplification of the WN model in Figure 10 in order to obtain the one in Figure 11 could be an example of use of formal net structural reduction techniques for the reduction of the cardinality of the state space. Work is currently in progress in order to formalize these reduction techniques in the performance evaluation framework provided by SWNs.

References

[1] M.K. Molloy. Performance analysis using stochastic Petri nets. *IEEE Transaction on Computers*, 31(9):913–917, September 1982.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, and F. Gregoretti. Modeling bus contention and memory interference in a multiprocessor system. *IEEE Transactions on Computers*, 32(1):60–72, January 1983.

[3] M. Ajmone Marsan, G. Balbo, and G. Conte. A class of generalized stochastic Petri nets for the performance analysis of multiprocessor systems. *ACM Transactions on Computer Systems*, 2(1), May 1984.

[4] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems.* MIT Press, Cambridge, USA, 1986.

[5] M. Ajmone Marsan and G. Chiola. Construction of generalized stochastic Petri net models of bus oriented multiprocessor systems by stepwise refinements. In *Proc.* 2^{nd} *Intern. Conf. on Modeling Techniques and Tools for Performance Analysis*, Sophia Antipolis, France, June 1985. ACM.

[6] M. Ajmone Marsan, G. Balbo, G. Chiola, and G. Conte. Modeling the software architecture of a prototype parallel machine. In *Proc. 1987 SIGMETRICS Conference*, Banf, Alberta, Canada, May 1987. ACM.

[7] C. Lin and D.C. Marinescu. On stochastic high level Petri nets. In *Proc. Intern. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.

[8] S. Haddad. *Une Categorie Regulire de Reseau de Petri de Haut Niveau: Definition, Proprietes et Reductions.* PhD thesis, Lab. MASI, Universite P. et M. Curie (Paris 6), Paris, France, Jun 1987. These de Doctorat, RR87/197 (in French).

[9] K. Jensen. Coloured Petri nets and the invariant method. *Theoretical Computer Science*, 14:317–336, 1981.

[10] C. Dutheillet and S. Haddad. Regular stochastic Petri nets. In *Proc. 10th Intern. Conf. Application and Theory of Petri Nets*, Bonn, Germany, June 1989.

[11] C. Dutheillet and S. Haddad. Aggregation and disaggregation of states in colored stochastic Petri nets: Application to a multiprocessor architecture. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

[12] P. Huber, A.M. Jensen, L.O. Jepsen, and K. Jensen. Towards reachability trees for high-level Petri nets. In G. Rozenberg, editor, *Advances on Petri Nets '84*, volume 188 of *LNCS*, pages 215–233. Springer Verlag, 1984.

[13] A. Zenie. Colored stochastic Petri nets. In *Proc. Intern. Workshop on Timed Petri Nets*, pages 262–271, Torino, Italy, July 1985. IEEE-CS Press.

[14] J.A. Carrasco. Automated construction of compound Markov chains from generalized stochastic high-level Petri nets. In *Proc.* 3^{rd} *Intern. Workshop on Petri Nets and Performance Models*, pages 93–102, Kyoto, Japan, December 1989. IEEE-CS Press.

[15] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On Well-Formed coloured nets and their symbolic reachability graph. In *Proc.* 11^{th} *Intern. Conference on Application and Theory of Petri Nets*, Paris, France, June 1990. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.

[16] Claude Dutheillet. *Symétries dans les réseaux colorés: Définition, analyse et application à l'évaluation de performances.* PhD thesis, Laboratoire MASI, Université Paris 6, France, January 1991. thèse de l'Université P. et M. Curie (in French).

[17] T. Murata. Petri nets: properties, analysis, and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[18] K. Jensen and G. Rozenberg, editors. *High-Level Petri Nets. Theory and Application.* Springer Verlag, 1991.

[19] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized stochastic Petri nets: A definition at the net level and its implications. *IEEE Transactions on Software Engineering*, 19(2):89–107, February 1993.

[20] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic Well-Formed coloured nets and multiprocessor modelling applications. In K. Jensen and G. Rozenberg, editors, *High-Level Petri Nets. Theory and Application*. Springer Verlag, 1991.

[21] G. Chiola, R. Gaeta, and M. Ribaudo. Designing an efficient tool for Stochastic Well-Formed Coloured Petri Nets. In R. Pooley and J. Hillston, editors, *Computer Performance Evaluation: Techniques and Tools*, pages 308–311, Edinburgh, UK, September 1992. Edinburgh University Press.

[22] G. Chiola. Great spn1.5 software architecture. In *Proc. 5^{th} Intern. Conf. on Modeling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, Feb 1991. IEEE-CS Press.

[23] M. Ajmone Marsan, G. Balbo, and G. Conte. Comparative performance analysis of single bus multiprocessor architectures. *IEEE Transactions on Computers*, 31(12), December 1982.

[24] S. Haddad. Generalization of reduction theory to coloured nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

[25] G. Chiola, S. Donatelli, and G. Franceschinis. GSPN versus SPN: what is the actual role of immediate transitions? In *Proc. 4^{th} Intern. Workshop on Petri Nets and Performance Models*, pages 20–31, Melbourne, Australia, December 1991. IEEE-CS Press.

[26] G. Balbo, G. Chiola, S.C. Bruell, and P. Chen. An example of modelling and evaluation of a concurrent program using coloured stochastic Petri nets: Lamport's fast mutual exclusion algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):221–240, March 1992. Reprinted in *High-Level Petri Nets. Theory and Application*, K. Jensen and G. Rozenberg (editors), Springer Verlag, 1991.

List of Figures

List of Tables