Chapter 1

SWN Definition

1.1 Introduction

Coloured Petri nets (CPNs) [1] have been proposed in the literature as a good modelling tool for the study of complex distributed systems. Initially, coloured nets were proposed as a kind of shortened notation to describe large and symmetric Petri net models. The common behaviour of several entities constituting a large system is described by the Petri net structure, while different entities are identified by different "coloured" tokens that can flow through the net.

Subsequently, the idea of *parametrization* has been suggested by Genrich [2]. A single net model can represent many different systems that differ only in the total number of components that behave similarly (i.e., that differ only in the cardinality of the colour sets). The whole family of such similar systems is modelled and studied using a single coloured Petri net, provided that the analysis results are parametric with respect to the colour sets. In [3] an example of parametric proof of a distributed algorithm is provided using coloured Petri net techniques.

One of the most powerful analysis techniques for bounded (coloured) Petri net systems is the analysis of the *reachability graph* (RG), i.e., the complete enumeration of the state space. In this case parametric analysis independent of the cardinality of the colour sets is of course impossible: a CPN model must be instantiated with particular choices of colour sets and initial markings in order to compute the RG. However, one can still take advantage of the coloured net formalism for the analysis: the symmetry properties of coloured models can be exploited in order to reduce the total number of states that should be analyzed to check for the properties of the model [4].

Well-formed coloured Petri nets (WNs) have been introduced in [5] together with a totally algorithmic procedure for the construction of a "symbolic reachability graph" (SRG) that exploits symmetries to reduce the size of the state space representation. The same reduction in state space can also be exploited for performance evaluation based on Markovian analysis. In [6] Stochastic WNs (SWNs) have been introduced to allow this kind of improved analysis. The aggregation of the state space has been proven to satisfy the strong lumpability condition for Markov chains [7].

Stochastic Petri nets have also been proposed for the study and performance evaluation of multiprocessor computer architectures [8, 9, 10]. 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 [11, 12] 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 behavioural characteristics and symmetries of the system under study. High level Petri net models are a more adequate tool (see, e.g., [13, 14]) 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) were proposed in [15] as a restriction of Colored Petri Nets (CPN) [1]. The interest in introducing such a restriction on the color domains and on the arc functions was that complete algorithms were also proposed for the computation of flows, reductions, and the definition of a Symbolic Reachability Graph (SRG) in this case. Stochastic models based on RNs called Regular Stochastic Petri Nets (RSPN) were subsequently introduced for performance evaluation purposes [16, 17]. 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 [4, 18, 14, 19] 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 was 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) [5]. 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 [20], some semi-formal transformation rules have been introduced and exploited to transform "quasi-regular" CPN models into RN models. However, the possibility of using quasi-regular net models in which some of the restrictions of RNs were relaxed with respect to the original definition and yet that were able to exploit the same SRG construction algorithm proposed for RNs, indicated the possibility of extending the original definition.

Here we present the 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 [17] 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 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 several modeling examples of multiprocessor computer architecture. The examples show how the technique can be exploited 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.

1.2 Well-Formed Nets: an informal introduction

We assume the reader is already familiar with the Petri net notation, both uncolored [21] and colored [22] versions. We also assume the reader to be familiar with the GSPN formalism [10, 23] and its multiprocessor applications [11].

1.2.1 Notation and basic definitions

We start by giving some basic definition and a short outline of the notation we shall use.

Definition 1.2.1 *A* multiset *a over a non-empty set A is a mapping $a \in [A \rightarrow I\!N]$, we use the notation $Bag(A)$ to denote the set of multisets 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 $a(x) \in I\!N$ 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 1.2.2 *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 $\langle e_1, \ldots, e_n\rangle$ where $e_i \in A_i$.

By generalization, if a_i is a multiset over A_i, we denote $\langle a_1, \ldots, a_n\rangle$, 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:

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

1.2.2 WN basics

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 [24] 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 [25] and that is now integrated in the package *GreatSPN* [26].

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. Within the PN formalism, a place can be used to represent a condition (or equivalently a boolean variable) by using the convention that the condition is true when one token is present in the place and false when no token is in the place. Alternatively a place may represent the "state" for a certain object, and the number of tokens in the place represents the number of (indistinguishable) objects that are in that state.

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. The syntax for the definition of basic color classes and their static subclasses within our tool is depicted in Figure 1.1, where "stat_sbc_id" is simply a string denoting a static subclass identifier. 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 1.2 where "class_id" is simply a string denoting a previously defined basic color class.

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

[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 $s[1-10]$.

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 1.3.

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

Basic class grammar			**Examples**
class	→	classtype staticlist	$Msgs = U Data\_msgs, Ack\_msg$
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 1.1: Class definition syntax.

assignment a collection of variable bindings.

For example let us consider the tuple $\langle src, dest, msg, \bigoplus cnt\rangle$, 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 $\langle rightfork\rangle + \langle bigoplusrightfork\rangle$ where variable $rightfork$ is of type $Philosophers$, stands for a set of cardinality two containing any element of $Philosophers$ and its successor. The expression $\langle broadcast\_msg, S_{Sites}\rangle$ 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 $Msgs$, the type of variable $broadcast\_msg$, and the set $Sites$. Finally expression $|Sites|\langle S_{Ack\_Msg}\rangle$ is a multiset containing $|Sites|$ elements all with the same color $\langle ack\rangle$.

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 1.4.(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 1.5, 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

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

Figure 1.2: Place colour definition syntax.

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

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 $< source, dest, \mathbf{m1}>$, $< source, dest, \mathbf{m2}>$, $< source, dest, \mathbf{m3}>$

[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 $c \in C(t)$.

Arc expression grammar

arc_expr	→	ϵ| sum_tuples
sum_tuples	→	coeff optional_predicate <expr_list> | sum_tuples plusop coeff optional_predicate <expr_list>
expr_list	→	expr_kernel | expr_kernel, expr_list
expr_kernel	→	term | expr_kernel plusop term
term	→	subclass_term | class_term | var_term | succ_term
subclass_term	→	coeff S staticname
class_term	→	coeff S | coeff S classname
var_term	→	coeff STRING
succ_term	→	coeff $\bigoplus$ STRING
coeff	→	NUM | VBAR staticname VBAR | VBAR classname VBAR | ϵ
plusop	→	$+ \mid -$

Examples

$< processor, memory >$
$< broadcast\_msg, S_{Sites} >$
$|Sites| < S_{Ack\_Msg} >$
$< src, dest, msg, \bigoplus cnt >$
$< rightfork > + < \bigoplus rightfork >$
$< phonenum1 + phonenum2, channel >$
$< S - source >$

Figure 1.3: Arc expression definition syntax.

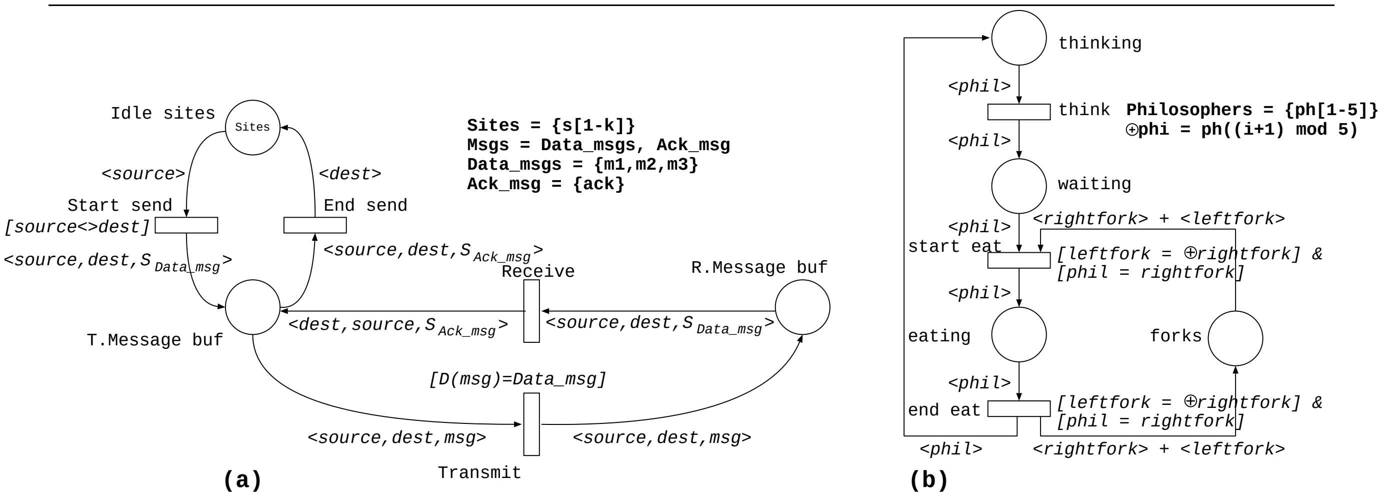


Figure 1.4: Examples of WNs.

(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 buf*.

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.

Predicate grammar

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

Examples

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

Figure 1.5: Predicate definition syntax.

1.2.3 WN Definition

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

In a CPN, subnets representing similar processes are folded one upon another. This folding consists in representing all the processes by a single subnet and identifying different resources or processes by different colours. A colour domain is then attached to each place and transition. The colour domain of a place determines the different colours of tokens that are allowed in that place. The colour instances of a transition represent the different possibilities for firing that transition. An arc between a place and a transition is labelled with a function mapping the colour domain of the transition on the set of bags associated with the colour domain of the place.

When firing a transition one colour is chosen in the transition domain and the functions labelling the input and inhibitor arcs of the transition are evaluated for this colour. This evaluation gives the number of tokens required for each colour in the input and inhibition places of the transition. If the current marking satisfies the firing condition, coloured tokens are removed from the input places according to the values of the functions, and they are put in the output places according to the values of the functions labelling the output arcs.

Well-formed Nets are a formalism with the same modelling power of CPNs from the point of view of representation conciseness. The main difference between WNs and CPNs is that in WNs the functions are defined following a precise syntax in a structured way. This structured definition allows one to directly infer some algebraic properties of functions or some behavioural symmetries in the net.

In the following subsections we first introduce the basic ingredients for building WNs and then we give their formal description and the associated firing rule. Finally we discuss the problems that predicates on arcs could introduce in the task of discovering and eliminating colour redundancies at a purely structural level. We also suggest a possible net transformation that allows one to move predicates from arcs to transitions, at the price of increasing the number of transitions in the net.

Object classes

The colours representing resources of the same type, such as a set of files, a set of identical processors, or a set of memory modules, are grouped in a class. Colour classes of a net are denoted C_i, with i ranging from 1 to n. If we call "objects" the basic components of the system (such as files or memories), we can realize that both place and transition domains are made of Cartesian products of object classes. These classes can be ordered if such a relation exists between the objects they contain (e.g. sites on a ring). Colours denote the elements of colour domains that are actually tuples of objects.

When objects belonging to the same class have different behaviours it is convenient to partition the class in *static subclasses*, so that all the objects belonging to the same static subclass behave the same way. Thus the model obtained respects the intrinsic symmetries of the system. The static subclasses of C_i are denoted $D_{i,q},\ \ q = 1, \ldots n_i$.

Formally, the family of colour classes of a WN is defined by: $C = \{C_1, \ldots, C_h,\ C_{h+1}, \ldots, C_n\}$ where $0 \leq h \leq n$ and

$C_i, \ \ 0 < i \leq h$ are non ordered classes,

$C_i, \ \ h < i \leq n$ are ordered classes. With each ordered class C_i we associate a function called "successor" denoted by $\oplus$ that fulfills the condition: $\forall c \in C_i,\ \ \bigcup_{k=1}^{|C_i|} \{\oplus^k c\} = C_i$

The partition of a colour class in static subclasses is defined by: $C_i = \bigcup_{q=1}^{n_i} D_{i,q}$
with $D_{i,q}$ predefined and disjoint sets of elements, i.e., $\forall 0 < i \leq n$, $\forall 0 < q \leq n_i$, $|D_{i,q}| > 0$ and $\forall 0 < r \leq n_i$, $q \neq r \Rightarrow D_{i,q} \cap D_{i,r} = \emptyset$.
In case of ordered classes (i.e. if $i > h$), each static subclass $D_{i,q}$ contains *contiguous* elements, i.e., $\forall q, \forall j > 1,\ \ c \in D_{i,q} \wedge \oplus^j c \in D_{i,q} \Longrightarrow \forall 0 < k < j,\ \oplus^k c \in D_{i,q}$. Moreover, the ordering of subclasses respects the successor function, i.e., $c \in D_{i,q} \wedge \oplus c \notin D_{i,q} \Longrightarrow \oplus c \in D_{i,r} \wedge$ if $q < n_i$ then $r = q+1$ else $r = 1$.

Colour domains

Let $I = \{1, \ldots, n\}$ be the set of indexes of the object classes.
Let $J = \bigotimes_{i=1}^{n} e_i$ be an n-tuple of integers belonging to $Bag(I)$.
We denote $C_J = \bigotimes_{i=1}^{n} (C_i)^{e_i}$ (or equivalently $C_J = \bigotimes_{i=1}^{n} \left(\bigotimes_{j=1}^{e_i} C_i \right)$), a Cartesian product of object domains.[4] If $J = \vec{0}$ then by definition $C_{\vec{0}} = \{\varepsilon\}$, where ε is called the *neutral colour*. An object tuple $c_J \in C_J$ is denoted $\bigotimes_{i=1}^{n} \bigotimes_{j=1}^{e_i} c_i^j$.

Definition 1.2.3 (Colour domain) *Let $r \in P \cup T$. The colour domain of r is defined by $C(r) = C_{J(r)}$, where $J(r)$ is a tuple of integers.*

Example: let $I = \{1,2,3\}$, $p_1 \in P$, and $J(p_1) = \langle 2,0,1 \rangle$; then $C(p_1) = C_1 \times C_1 \times C_3$ (or $C(p_1) = (C_1)^2 \times C_3$).

Standard predicates

Here we define the predicates that can be used in a standard colour function and that can also be used to restrict the firing possibilities of a transition. When firing a transition, the same colour class can be instantiated several times. It is then interesting to make some selection on the possible firings by introducing predicates on the transitions. The predicates that we allow in our model are: $X_i^k = X_i^{k'}$ (the same object is chosen for both instantiations); $X_i^k = \oplus X_i^{k'}$ (the object chosen for the k^{th} instantiation is the successor of the object chosen for the k'^{th} instantiation); $d(X_i^k) = q$ (the chosen object belongs to a particular static subclass[5]; $d(X_i^k) = d(X_i^{k'})$ (two chosen objects belong to the same static subclass); and combinations of those predicates using the $\wedge$, $\vee$, and $\neg$ logical operators.
The values of the predicates are given by: $\phi_J : C_J \rightarrow \{True, False\}$

- $(True)(c_J) \quad = \quad (True)$
- $(X_i^k = X_i^{k'})(c_J) \quad = \quad (c_i^k = c_i^{k'})$
- $(X_i^k = \oplus X_i^{k'})(c_J) \quad = \quad (c_i^k = \oplus c_i^{k'})$
- $(d(X_i^k) = q)(c_J) \quad = \quad (c_i^k \in D_{i,q})$
- $(d(X_i^k) = d(X_i^j))(c_J) \quad = \quad \exists q :\ (c_i^k \in D_{i,q}) \wedge (c_i^j \in D_{i,q})$

and all the possible combinations with the operators $\wedge, \vee$, and $\neg$:

- $(\phi_J \vee \phi'_J)(c_J) = (\phi_J(c_J) \vee \phi'_J(c_J))$

[4] Note that if $e_i = 0$, then C_i does not appear in the Cartesian product.
[5] Function d gives for each colour element $c_i \in C_i$, the index q of the static colour domain $D_{i,q}$ to which it belongs: $d : \bigcup_{i \in I} C_i \rightarrow I\!N$ such that $\forall c_i \in D_{i,q},\ \ d(c_i) = q$

- $(\phi_J \wedge \phi'_J)(c_J) = (\phi_J(c_J) \wedge \phi'_J(c_J))$
- $(\neg\phi_J)(c_J) = \neg(\phi_J(c_J))$

Notice the predicates $(X_i^k = X_i^{k'})$ (and often also the predicates $(X_i^k = \oplus X_i^{k'})$) on transitions could be avoided by using the same function X_i^k instead of $X_i^{k'}$ plus the predicate (or the function $\oplus X_i^{k'}$ instead of X_i^k plus the predicate); a side effect of these simplifications is that the number of transition colour components decreases by one. Observe however that this simplification is not possible in predicates like $(X_i^k = X_i^{k'}) \vee (X_i^j = X_i^{j'})$.

Colour functions

The number of colour functions needed for actual CPN models is usually limited. This is due to the fact that CPNs are used to represent only a limited set of synchronization mechanisms of parallel and distributed systems. This is the main practical justification for the introduction of a constrained syntax in WNs.

We can actually distinguish three main families of colour functions. All these functions can be composed by Cartesian product and/or linear combinations.

Projection functions denoted by X_i^j allow one to select an object from class C_i. The behaviour of the selected object is independent of the behaviour of the other objects of the class when firing the transition. If several functions X_i^j appear next to the same transition with the same superscript j, then the same object must be selected to instantiate the different occurrences of the function, whereas for different values of j, the objects can be different.

$$X_i^j \left(\bigotimes_{k=1}^{n} \bigotimes_{l=1}^{e_i} c_k^l \right) = c_i^j$$

Successor functions are used, among other applications, in case of message numbering, counters, or token ring protocols. These functions denoted $\oplus X_i^j$ imply an ordering relation among objects of the basic classes. Of course, $\oplus X_i^j$ makes sense only if X_i^j appears near the same transition, and it represents the successor of the object selected by X_i^j.

Diffusion functions allow one to synchronize all the objects belonging to the same class C_i if they label an input or inhibitor arc, or to diffuse to all the objects of C_i if they label an output arc. In many cases a diffusion function applies only to objects belonging to the same static subclass. This function is denoted $S_{i,q}$, where $D_{i,q}$ is the static subclass to which it applies.

$$\forall c \in C(t), \quad S_{i,q}(c) = \cup c_i \in D_{i,q} c_i$$

We use also S_i as a shortened notation for $\sum_{q=1}^{n_i} S_{i,q}$. Note that the value of the diffusion function $S_{i,q}$ does not depend on the objects chosen to instantiate the transition so that C_i might not appear in the transition domain. In fact, $S_{i,q}$ is a constant function. In particular, it can be used to denote a constant object if $|D_{i,q}| = 1$.

If we consider a place whose domain is reduced to a single occurrence of an object class C_i the colour functions that may apply to this place are linear combinations of projection functions X_i^j, diffusion functions $S_{i,q}$ applying to static subclasses, and (in case C_i is ordered) successor functions $\oplus X_i^j$. The coefficients of the combination may be negative, provided that it does not result in a negative number of objects selected in any static subclass. Moreover, the function may be guarded by a standard predicate.

In the most general case, the colour domain of a place is a Cartesian product of object classes that possibly involves several occurrences of the same class. Thus, as the kind of functions described above may apply to each occurrence of each class, the resulting colour function is a Cartesian product both on the indexes of classes and the number of occurrences of each class of the functions that apply to a single object class. These functions can be linearly combined, furthermore they can be guarded by a predicate.

The formal definition of standard colour functions in WNs is the following:

$$\sum \delta.[pred] \bigotimes_{i \in I} \bigotimes_{j=1}^{e_i} \left\{ \sum_{q=1}^{n_i} \alpha_{i,q}.S_{i,q} + \sum_{k=1}^{e_i} \left(\beta_{i,k}.X_i^k + \gamma_{i,k}. \oplus X_i^k \right) \right\} \tag{1.1}$$

where δ is a positive integer, and $[pred]$ stands for a standard predicate. The value of the colour function "$[pred]f$" is given by: $[pred]f(c) \;=\;$ If $pred(c)$ then $f(c)$ else 0.

WN formal definition

Definition 1.2.4 (WN) *A well formed net* $\mathcal{N} = \langle P, T, C, J, W^-, W^+, W^H, \Phi, \pi \rangle$ *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 object classes:* $C = \{C_1, \ldots, C_n\}$ *(we denote* $I = \{1, \ldots, n\}$ *the ordered set of indexes);* C_i *is partitioned in static subclasses:* $C_i = \bigcup_{q=1}^{n_i} D_{i,q}$*; if* $n_i = 1$ *then no distinction is made between class* C_i *and the unique subclass* $D_{i,1}$*; the partition in static subclasses of a colour class* C_i *is correct if at most one static subclass is never used in the net* [6]*;*

$J : P \cup T \rightarrow Bag(I)$*, where* $Bag(I)$ *is the set of multisets on* I*.* $C(r) = C_{J(r)}$ *denotes the 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 standard functions of the form shown in Equation (1.1);*

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

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

A WN can represent a whole family of models that differ only for the actual cardinality of the basic colour sets and for the initial marking. Each individual model obtained by instantiating a WN with a given set of parameters is called a WN *system*.

Definition 1.2.5 (WN system) *A well formed net system* $\Sigma = (\mathcal{N}, CS, M_0)$ *is made of:*

$\mathcal{N}$ *a well-formed net;*

[6] A static subclass is *used* in a net if the function $S_{i,q}$ occurs in some function or a reference to the subclass occurs in a predicate or in the initial marking. Notice that a predicate of the kind $[d(X_i^k) = d(X_i^j)]$ makes every static subclass to be considered *used*.

CS *an instantiation function for the basic colour sets of the WN such that* $\forall i : 0 < i \leq n$, $\forall q : 0 < q \leq n_i$, $CS(D_{i,q})$ *is a particular set of basic elements;*

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

Definition 1.2.6 (Enabling rule) *A transition* t *is enabled for a colour* c *in a marking* M, *and denoted* $(t,c) \in \mathcal{E}(M)$ *iff:*

i) $\Phi(t)(c) \wedge \quad \forall p \in P,\ W^-(p,t)(c) \leq M(p) \ \wedge\ (W^H(p,t)(c) = 0 \vee W^H(p,t)(c) > M(p))$
(which is the expression of the firing rule in a net without priorities)

ii) $\forall t' : \pi(t') > \pi(t),\ \forall c' \in C(t'),\quad (t',c')$ *does not fulfill (i)*

Definition 1.2.7 (Firing rule) *The firing of the enabled transition* t *for colour* c *(i.e. such that* $(t,c) \in \mathcal{E}(M)$*) leads to a new marking* M', *denoted* $M[t,c\rangle M'$, *such that:*
$\forall p \in P, \quad M'(p) = M(p) + W^+(p,t)(c) - W^-(p,t)(c)$

The colour c is called the colour *instantiated* by the firing. As c is a product of basic objects, each component is also said to be instantiated by the firing.

1.2.4 The 4×400 Relay Race Example

Here we present a simple example of the use of the formalism that exploits most of its features. The WN model is depicted in Figure 1.6.

1.2.5 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.

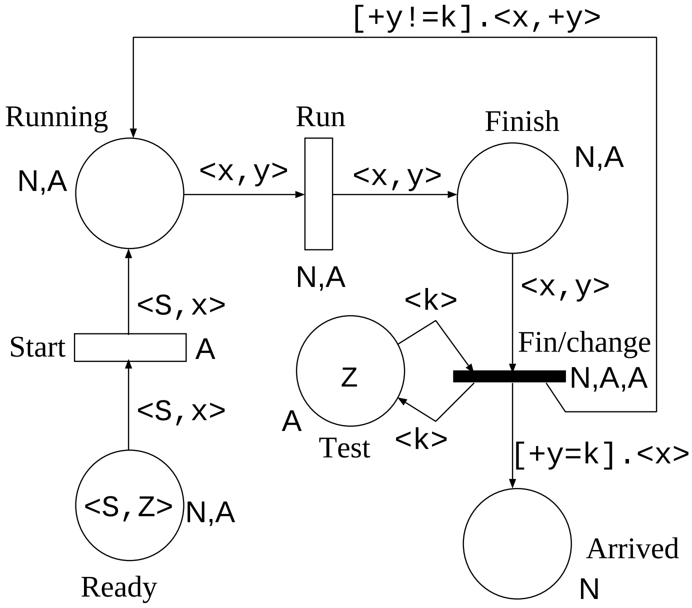
Proposition 1.2.1 *Any CPN can be transformed into a WN with the same basic structure, same color domains (possibly partitioned in static subclasses), equivalent arc labeling. See [24] for a proof.*

Definition 1.2.8 (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:*

- $\forall i : 0 < i \leq h\ s_i$ *is a permutation on* C_i *such that* $\forall D_{i,q},\ s_i(D_{i,q}) = D_{i,q}$;
- $\forall h : h < i \leq n\ s_i$ *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 1.2.9 (Marking permutation) *Let* M *be a marking, and* $s \in \xi$ *a permutation. Then* $M'' = s.M$ *is a marking defined by:* $\forall p \in P,\ \forall c \in C(p),\ M''(p, s(c)) = s.M(p, s(c)) = M(p,c)$.



N **:** non ordered set of Nation identities

$$N \quad = \quad \{\ n_1, n_2, \ldots, n_8\ \}$$

A **:** ordered set of 4 Athlets identities for each team

$$A \quad = \quad \{\ a_1, a_2, a_3, a_4\ \}$$

Z **:** Dynamic subclass of A with cardinality 1
(representing the first athlet to run)

Figure 1.6: The 4×400 Olympic Relay Race.

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 1.2.2 *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 1.2.10 (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 $\mathcal{M}_0$.

1.3 Symbolic Reachability Graph

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

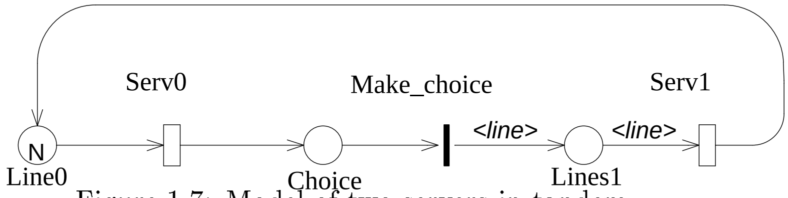


Figure 1.7: Model of two servers in tandem.

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 behaviour. 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.

1.3.1 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.

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 1.7. 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:

$$Line0(1)Choice(1)Line1(2\langle l_1\rangle, \langle l_2\rangle)$$

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 behaviour 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.

$$Line0(1)Choice(1)Line1(2\langle l_4\rangle, \langle l_3\rangle)$$

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[7] as $\{z_1, \ldots, z_4\}$. An example of symbolic marking would thus be the following:

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

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

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

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 1.2.10. 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

$$\mathcal{M}'' = Line0(1)Line1(2\langle z_1\rangle, 2\langle z_2\rangle) \tag{1.2}$$

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 1.8 a pictorial representation of the grouping induced by the symbolic marking and symbolic firing concept is shown.

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

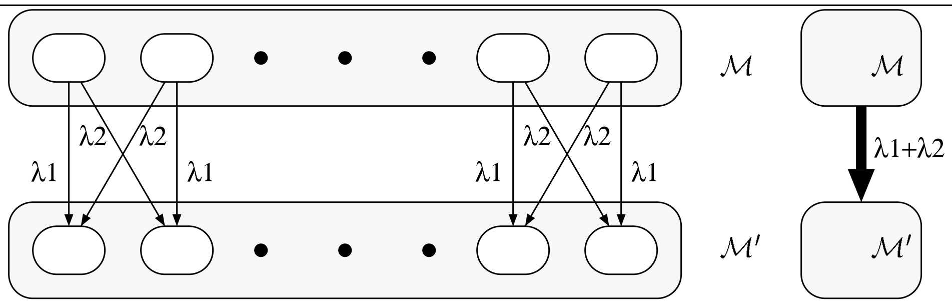


Figure 1.8: Grouping of ordinary markings and arcs.

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 1.8 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 (1.2); 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: $\hat{C}_i = \{Z_i^j \mid 0 < j \leq m_i\}$. We'll use the extended notation $\hat{C}(r)$, $r \in P \cup T$, 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 $\forall i$, $\sum_{j=1}^{m_i} card(Z_i^j) = |C_i|$; 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 $\forall Z_i^j$, $d(Z_i^j) = q \in [1..n_i]$ and $\forall 0 < i \leq n$, $\forall 0 < j < k \leq m_i$, $d(Z_i^j) \leq d(Z_i^k)$

- $\forall p \in P$, $mark(p) : \hat{C}(p) \rightarrow I\!N$

The associated semantics can be stated as:

- $\hat{C}_i$ is a non-instantiated partition of the basic class C_i

- Z_i^j represents any subset $C_{i,j}$ of C_i such that $|C_{i,j}| = card(Z_i^j)$
  in case C_i is ordered ($i > h$), the elements of $C_{i,j}$ are contiguous

- $\forall M \in \mathcal{R}$, $\forall i \in I$, $\exists \eta_i : C_i \rightarrow \hat{C}_i$
  (η_i is a *valid assignment* of subsets of C_i to dynamic subclasses Z_i^j);

- $\eta_i : C_i \rightarrow \hat{C}_i$ 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 [5] or in [27]). 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,

$$mark_{sp}(Z_1^1)(Line0) = 1, mark_{sp}(Z_1^1)(Choice) = 0, mark_{sp}(Z_1^1)(Line1) = 2, (mark_{sp}(Z_1^1) = \langle 1, 0, 2\rangle)$$

$$mark_{sp}(Z_1^2)(Line0) = 1, mark_{sp}(Z_1^2)(Choice) = 0, mark_{sp}(Z_1^2)(Line1) = 0, (mark_{sp}(Z_1^2 = \langle 1, 0, 0\rangle)$$

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

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

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

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

$\mathcal{S} := \hat{\xi}$; *(* 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. (1.3) **)*
    `fi`
`od`
$K(\mathcal{M}) := |\mathcal{S}|$

Figure 1.9: Sketch of the computation algorithm for the canonical representation.

corresponds to a matrix that is lexicographically ordered with respect to the following reading order of the matrix elements:

$$(1, 1, \ldots, 1), (2, 1, \ldots, 1), \ldots, (m(1), 1, \ldots, 1), (1, 2, 1, \ldots, 1), \ldots \tag{1.3}$$

The complete algorithm for the computation of the canonical representation is described in [5] or in [27]. 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*[8]. 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 enumeration of the indexes follows the order defined in 1.3); 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 1.9.

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.

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

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[9] 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 1.3.2 (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:*

$\lambda = \{\lambda_i : \{1, \ldots, e_i\} \to I\!N^+\}, \qquad \mu = \{\mu_i : \{1, \ldots, e_i\} \to I\!N^+\},$ *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 $\nexists x : \lambda_i(x) = j$), 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 1.3.3 (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

[9] 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 be derived from $\mathcal{R}.mark$: observe that using the sum and Cartesian product operations on multisets defined in 1.2.1 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 1.3.1 *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 $\mathcal{M}'$;

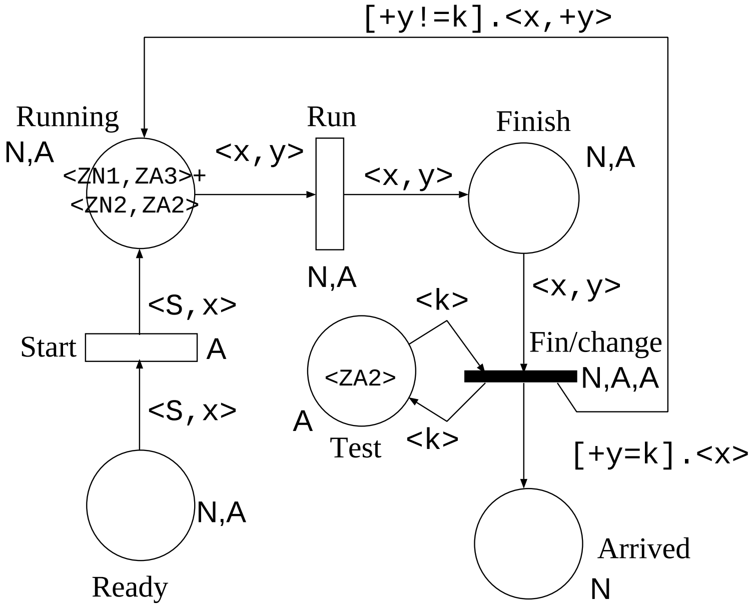


Figure 1.10: An example of symbolic firing: initial marking.

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 the simple example shown in Figure 1.6 in order to illustrate the four steps of the symbolic firing rule. The different steps are illustrated in graphical form in Figure 1.10, Figure 1.11 and Figure 1.12.

The initial marking $\mathcal{M}$ of the model shown in Figure 1.10 is

$$\mathcal{M} = Running(\langle ZN1, ZA3\rangle + \langle ZN2, ZA2\rangle)Test(\langle ZA2\rangle)$$
$$|ZN1| = 1,\ |ZN2| = 7,\ |ZA1| = 2,\ |ZA2| = 1,\ |ZA3| = 1$$

The splitting of marking $\mathcal{M}$ is shown in Figure 1.11. Notice that the input and output arcs of transition *Run* are labelled with the tuple $\langle ZN3, ZA2\rangle$ which is actually involved in the firing operation.

Figure 1.12 shows the new tanglible state $\mathcal{M}'$ reached after the firing of the timed transition *Run* and the immediate transition $Fin/change$. The cardinalities of the dynamic subclasses are:

$$|ZN1| = 1,\ |ZN2| = 6,\ |ZN3| = 1,\ |ZA1| = 2,\ |ZA2| = 1,\ |ZA3| = 1$$

In this marking the dynamic subclasses $ZN1$ and $ZN3$ can be merged into a single one of cardinality 2 because they have the same distribution in the places of the net. After the merging phase, the

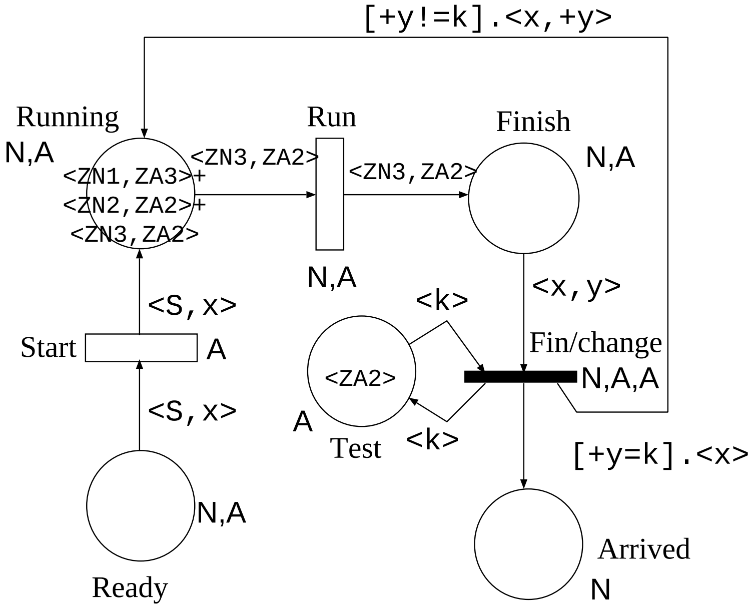


Figure 1.11: An example of symbolic firing: splitting.

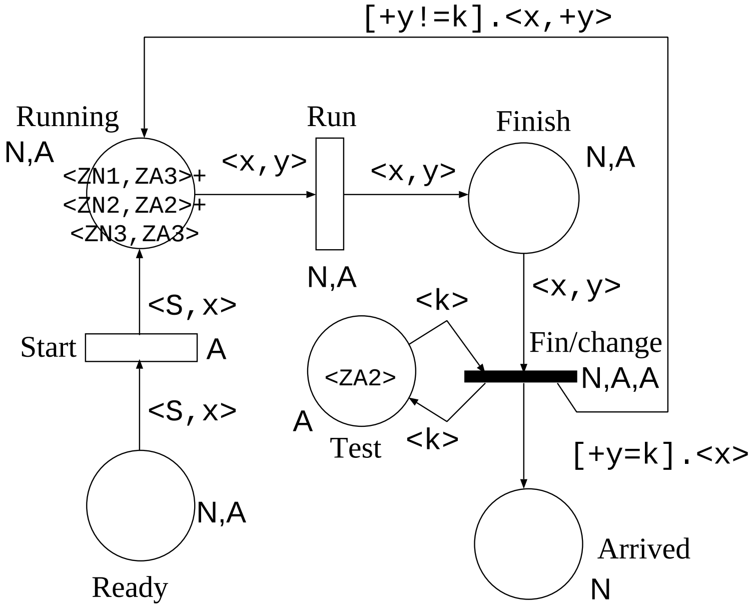


Figure 1.12: An example of symbolic firing: merging.

canonical representation of $\mathcal{M}'$ can be computed. The model reaches exactly the marking shown in Figure 1.10 but in this case the cardinalities of the dynamic subclasses are equal to:

$$|ZN1| = 2,\ |ZN2| = 6,\ |ZA1| = 2,\ |ZA2| = 1,\ |ZA3| = 1$$

1.3.2 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 behaviour 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.

Property 1.3.1 *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 1.3.2 *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[10].

Property 1.3.3 *Necessary condition for ergodicity.*
Strong connection of RG $\Longrightarrow$ Strong connection of SRG, but not vice-versa.

In [5] an example is given in which the SRG is strongly connected while the underlying RG is not.

Property 1.3.4 *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 [15] for the restricted case of Regular nets[11]. 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 [5] 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.

Property 1.3.5 *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:

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

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

- *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 $\mu_i^k = \sup_{x \,:\, \lambda_i(x)=k} \mu_i(x)$

Chapter 2

Symmetries and Markovian Analysis

2.1 Stochastic WNs and lumped Markov chains

Stochastic Regular nets where already defined in [16]. 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.

2.1.1 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 [23]. 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 2.1.1 *Static partition of symbolic markings.*

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

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

Definition 2.1.1 (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*

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

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:

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

2.1.2 RG based Markovian technique

Stochastic Petri nets (without distinction between ordinary or colored ones) can be used as a convenient modeling tool for the definition of a semi-Markov stochastic process based on the reachability graph of the net. The steady-state probability distribution of this process can be numerically computed by deriving the embedded Markov chain and solving the associated eigenvector equations. We report here a brief algorithmic outline of the technique in the case of colored Petri nets. Note that in this case the color structure is not taken into account for the Markovian analysis.

The embedded chain is defined on the instants of transition firings that determine a change of marking in the net. We assume that all transitions (if any) from a state to itself (corresponding to the case $\forall p \in P$, $W^+(p,t)(c) - W^-(p,t)(c) = 0$) have been previously eliminated from the RG representation. The probability distribution vector $\psi = [\psi_1, \ldots, \psi_k]$, where ψ_l denotes the probability of the l-th state (or marking), is obtained as follows:

1. Construction of the state space description
    - construction of the RG of the net
    - elimination of self-loop arcs
    - for each arc of the RG, computation of $\theta(t)(\tilde{c}, \tilde{M})$

2. Construction of the square matrix $Q = [q_{i,j}]$ of dimension $N = |RS|$

$$\text{if } i \neq j, \quad q_{i,j} = \frac{\sum_{[t,c]\ :\ M_i[t,c\rangle M_j} \theta(t)(\tilde{c}, \tilde{M}_i)}{\sum_{[t,c]\ :\ M_i[t,c\rangle} \theta(t)(\tilde{c}, \tilde{M}_i)}$$

$q_{i,i} = -\sum_{j \neq i} q_{i,j}$

3. Numerical solution of the system of linear equations

$$\begin{cases} Y.Q = 0 \\ ||Y|| = \sum_{i=1}^{N} Y_i = 1 \end{cases}$$

The solution is unique (*ergodic* model) if the RG contains a unique strongly connected component

4. Computation of the steady-state probability distribution for tangible markings:

$$\psi_i = Y_i . \frac{\frac{1}{\sum_{[t,c]\ :\ M_i[t,c\rangle} \theta(t)(\tilde{c}, \tilde{M}_i)}}{\sum_j \frac{Y_j}{\sum_{[t',c']\ :\ M_j[t',c'\rangle} \theta(t')(\tilde{c}', \tilde{M}_j)}}$$

2.1.3 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:

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

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.

In the following section we give an outline of the formal lumpability proof.

2.1.4 Proof of lumpability

Note that properties 3.3 and 3.4 give a characterization of the ergodicity of the model in terms of the strong connection of the SRG. In particular, they provide respectively a necessary and a sufficient ergodicity condition that can be checked directly on the SRG without an explicit computation of the RG.

Lumpability

Because of the symmetry of the weighting function θ and the fact that the firing relation is preserved by applying a permutation both to the markings and to the transition instantiation, we can show the following property.

Proposition 2.1.1 *The probability of going from $M_{i,k}$ to $M_{j,q}$ is the same as the probability of going from $M_{i,s.k}$ to $M_{j,s.q}$, i.e.:*

$$q_{(i,k)(j,q)} = q_{(i,s.k)(j,s.q)}$$

As a consequence, it is easy to show that the lumping condition is satisfied.

Proposition 2.1.2 *The embedded Markov chain verifies the lumping condition [7], i.e.:*

$$\forall \mathcal{M}_j, \quad \forall M_{i,k}, \quad \forall s \in \xi, \qquad \sum_{M_{j,q} \in \mathcal{M}_j} q_{(i,k)(j,q)} = \sum_{M_{j,q} \in \mathcal{M}_j} q_{(i,s.k)(j,q)}$$

From the lumping theory we know that a reduced system can be computed whose variables are the symbolic marking probabilities and whose coefficients are: $\sum_{M_{j,q} \in \mathcal{M}_j} q_{(i,k)(j,q)}$. We show now that these coefficients can be computed using the information contained in the SRG alone.

Computation of the coefficients

Using the values given by the RG based algorithm one obtains:

$$\sum_{M_{j,q} \in \mathcal{M}_j} q_{(i,k)(j,q)} = \frac{\sum_{M_{j,q} \in \mathcal{M}_j} \sum_{[t,c] \,:\, M_{i,k}[t,c\rangle M_{j,q}} \theta(t)(\tilde{c}, \tilde{M}_{i,k})}{\sum_{[t,c] \,:\, M_{i,k}[t,c\rangle} \theta(t)(\tilde{c}, \tilde{M}_{i,k})}$$

We now state two propositions showing that this expression can be computed from the SRG information.

Proposition 2.1.3

$$\sum_{M_{j,q}\in\mathcal{M}_j}\ \sum_{[t,c]\ :\ M_{i,k}[t,c\rangle M_{j,q}} \theta(t)(\tilde{c},\tilde{M}_{i,k}) = \sum_{[t,\lambda,\mu]\ :\ \mathcal{M}_i[t,\lambda,\mu\rangle\mathcal{M}_j} \Theta(t)(\lambda,\mu,\tilde{\mathcal{M}}_i).|\eta^{-1}(t,\lambda,\mu)|$$

Proposition 2.1.4

$$\sum_{[t,c]\ :\ M_{i,k}[t,c\rangle} \theta(t)(\tilde{c},\tilde{M}_{i,k}) = \sum_{[t,\lambda,\mu]\ :\ \mathcal{M}_i[t,\lambda,\mu\rangle} \Theta(t)(\lambda,\mu,\tilde{\mathcal{M}}_i).|\eta^{-1}(t,\lambda,\mu)|$$

The proof of the two propositions is similar. As stated in Property 1.3.5 in Section 1.3.2, the ordinary arcs between $M_{i,k}$ and any marking of the equivalence class $\mathcal{M}_j$ can be partitioned according to their image by η. Hence we have

$$\sum_{M_{j,q}\in\mathcal{M}_j}\ \sum_{[t,c]\ :\ M_{i,k}[t,c\rangle M_{j,q}} \theta(t)(\tilde{c},\tilde{M}_{i,k}) = \sum_{[t,\lambda,\mu]\ :\ \mathcal{M}_i[t,\lambda,\mu\rangle}\ \sum_{\eta^{-1}(\lambda,\mu)=[t,c]} \Theta(t)(\lambda,\mu,\tilde{\mathcal{M}}_i)$$

$$= \sum_{[t,\lambda,\mu]\ :\ \mathcal{M}_i[t,\lambda,\mu\rangle\mathcal{M}_j} \Theta(t)(\lambda,\mu,\tilde{\mathcal{M}}_i).|\eta^{-1}(t,\lambda,\mu)|$$

The coefficients of the system computed using the SRG based algorithm are the same as the coefficients of the system obtained when lumping the Markov chain built on the RG. The SRG based algorithm allows one to derive the symbolic marking probabilities. With the additional property that all the markings within a symbolic marking have the same probability it is immediate to derive the ordinary marking probabilities.

Equiprobability

Proposition 2.1.5

$$\forall\mathcal{M}_i,\quad \forall M_{j,q},\quad \forall s\in\xi,\qquad \sum_{M_{i,k}\in\mathcal{M}_i} q_{(i,k)(j,q)} = \sum_{M_{i,k}\in\mathcal{M}_i} q_{(i,k)(j,s.q)}$$

This proposition, together with Proposition 2.1.1, proves that all the actual markings within a symbolic marking have the same input rate from any symbolic marking and the same output rate to any symbolic marking. It is then easy to prove that they have the same probability.

Chapter 3

Bounds for Quasi-Lumpable Nets

3.1 Motivations

The work presented in this section is related to the lumpability approach to state space reduction. It focuses on Markov chains that incorporate some structural regularity of the kind required by lumpability methods. It can be regarded as a generalization of the lumpability method since it applies to a larger set of models, and gives the exact values for the desired performance figures when the model is actually lumpable, while it gives upper and lower bounds on these figures when the model is not exactly lumpable. We first define the concept of a *quasi-lumpable* Markov chain and show that given a *quasi-lumpable* Markov chain, it can be altered in such a way that the new resulting MC is lumpable and is such that Courtois and Semal's steady state probability bounding methods [28, 29] can be applied to the new lumped MC to obtain bounds on the performance measures of the original model.

In Section 3.2 the definition of quasi-lumpable MC is given, and some issues about the extent of applicability of the method are discussed. The most important issue addressed in this section is how to check if a given system has a quasi-lumpable underlying MC. We claim that it is not reasonable to try to detect the quasi-lumpability property by inspecting the MC itself, but rather by exploiting some high level property of the abstract model from which the MC is derived, such as in the case of SWNs.

The computational advantages of the proposed method are clear in this case. The possibility of deriving the lumped MC directly, without building the complete one first, makes the method more effective.

3.2 Quasi-Lumpable Markov Chains

Definition 3.2.1 *A Deterministic Time Markov Chain is said to be* ϵ-quasi-lumpable *with respect to a given state space partition* $\mathcal{A}$ *if its transition probability matrix* P *can be rewritten as* $P = P^- + P^\epsilon$, *where* P^- *is the largest lower bound (componentwise) for* P *that satisfies the following lumpability conditions [7]:*

$$\forall a_i, a_j \in \mathcal{A}, \forall s \in a_i : \sum_{s' \in a_j} P^-[s, s'] = K_{i,j}$$

($K_{i,j}$ *is a constant value that depends only on* i *and* j*), and no element in* P^ϵ *is greater than* ϵ *in value. A similar definition applies to CTMC, where the probability matrix* P *is substituted with the infinitesimal generator* Q*.*

The intention is that P^ϵ is a matrix with many more zero elements than P^- and with relatively small non-zero elements. Henceforth we use the term "quasi-lumpable DTMC" for a DTMC that is ϵ-quasi-lumpable for some ϵ.

An example[1] of P, P^- and P^ϵ is given in Figure 3.1 (we assume that $\lambda_2 = \lambda_1 + \epsilon$). The state partition is the following:

$$\mathcal{A} = \{\{1\}, \{2,3\}, \{4,5\}, \{6\}\}$$

$$P = \begin{array}{c|c|cc|cc|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & \\ \hline
2 & & \bullet & & \lambda_1 & \lambda_2 & \\
3 & & & \bullet & & 2\lambda_1 & \\ \hline
4 & & & & \bullet & & \lambda_2 \\
5 & & & & & \bullet & \lambda_1 \\ \hline
6 & \mu & & & & & \bullet \\ \hline
\end{array}
=
\begin{array}{c|c|cc|cc|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & \\ \hline
2 & & \bullet & & \lambda_1 & \lambda_1 & \\
3 & & & \bullet & & 2\lambda_1 & \\ \hline
4 & & & & \bullet & & \lambda_1 \\
5 & & & & & \bullet & \lambda_1 \\ \hline
6 & \mu & & & & & \bullet \\ \hline
\end{array}
+
\begin{array}{c|c|cc|cc|c|}
 & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline
1 & & & & & & \\ \hline
2 & & & & & \epsilon & \\
3 & & & & & & \\ \hline
4 & & & & & & \epsilon \\
5 & & & & & & \\ \hline
6 & & & & & & \\ \hline
\end{array}$$

Figure 3.1: An example of P, P^-, P^ϵ.

Quasi Lumpable MC and High Level Modeling Formalisms The motivation for this work on quasi-lumpable MC came from the observation that there are systems that happen to have a very regular state space due to their highly symmetric structure. Often the regularity allows the exact lumping of the state space, while other times the lumping is prevented by some "small" perturbation in the transition rates.

Some work exists that takes advantage of a high level description of the system, in which the structural symmetries are evident, to automatically generate a lumped state space [30, 6]. In this section we will explain through an example that this approach can be extended to the cases in which the state space has only a quasi-lumpable structure; we will specifically refer to the high level formalism of Stochastic Well-formed Nets (SWN).

Usually, a state space generated from a high level model of a system is lumpable when the chosen state representation is so detailed that the behaviour description incorporated in the state space is redundant. In this case lumping is equivalent to choosing a more abstract state representation in which many states that were distinct with respect to the more detailed representation become undistinguishable.

A classical example for different possible state description detail levels is the following (already used in Figure 1.7). Consider the closed system depicted in Figure 3.2.(a) and assume that there are 5 customers in the system. Customers cycle between two service centers. The first one is a single server machine (the associated delay is an exponentially distributed random variable with parameter μ). The second is a multiple server machine with four servers. A customer chooses randomly one of the four servers on each visit to the multiserver node. Let's assume that the choice is equiprobable for all queues. We also assume that the service time at each server is exponential and does not depend on the customer identity. Of course the state description chosen depends on the distribution of service times and on the scheduling policy of each server, moreover it might also depend on what performance measure we need. In Figure 3.2.(b) a SWN representation of the system is depicted.

[1] The diagonal elements of P are such that the matrix is stochastic.

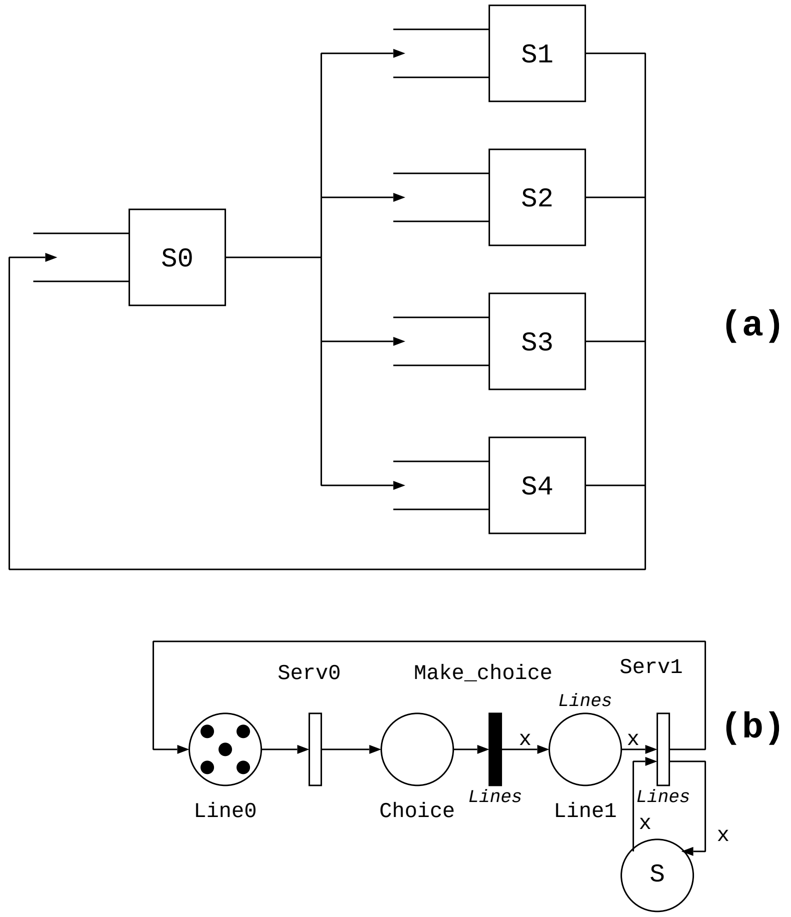


Figure 3.2: A Multiple Server System Model.

A quite natural representation for the state of this system (suggested by the queueing network model) is a five element vector whose i^{th} element represents the number of clients in the i^{th} queue. Let us call this state representation, $R1$. However, as long as the service rates of the four servers that work in parallel ($S1 - S4$) are the same (e.g., λ), a more abstract representation $R2$ for the state can be used, namely $\langle n_0, nq_0, nq_1, \ldots, nq_5 \rangle$ where n_0 is the number of customers in the first queue, and $nq_j,\ j = 0, \ldots, 5,$ is the number of queues of length j in the second stage of four servers. Hence, the state $\langle 1, 1, 2, 1, 0, 0, 0 \rangle$ of type $R2$ comprises several (more detailed) states of type $R1$, for example $\langle 1, 2, 1, 1, 0 \rangle$ and $\langle 1, 1, 0, 1, 2 \rangle$.

In case the four servers in parallel did not have the same speed, for example two of them had a given speed λ_1 while the other two had speed $\lambda_2 > \lambda_1$, then the first description would still be too detailed while the second would be too abstract. The appropriate state representation (called $R3$) in the latter case is $\langle n_0, nq_0^1, \ldots, nq_5^1, nq_0^2, \ldots, nq_5^2 \rangle$ where n_0 is still the number of customers in the first queue while nq_j^i is the number of queues of length j among the servers with speed λ_i.

Observe that if the detailed state description $R1$ is used, then the structure[2] of the state transition graph for the system where $S1 - S4$ have the same speed is equal to that of the system where two servers have speed λ_1 and the other two have speed λ_2. However if the four parallel servers do not have exactly the same service time then the aggregation can only be partial because some of the state

[2]Two state transition graphs have the same structure iff there is a bijection γ between the two set of states $\mathcal{S}_1$ and $\mathcal{S}_2$, and the set of non-negative state transitions $\mathcal{T}_1 = \{(s, s')\}$ of $\mathcal{S}_1$ and $\mathcal{T}_2 = \{(q, q')\}$ of $\mathcal{S}_2$ are such that $\mathcal{T}_2 = \{(q, q') \ : \ q = \gamma(s) \wedge q' = \gamma(s') \wedge (s, s') \in \mathcal{T}_1\}$.

transitions cannot be merged due to their different associated rates. If the difference in the service time of the two groups of servers is relatively small, then we will call the system quasi-lumpable. So quasi-lumpability is a property that can be observed in systems where a regularity in the state space structure is only partially reflected in the transition rates. Very often in this situation the modeler might decide to introduce an approximation and force the transition rates to become equal, thus forcing the model to become lumpable, using some kind of weighted average for the modified transition rates. This approach however may produce misleading results. The method proposed in the next section is safer from this point of view since it gives bounds rather than an approximation of uncertain error.

In the literature a number of works exist dealing with the automatic construction of a lumped MC from a high level description of the model. Some of these methods could be extended using the bounding method proposed in this paper; for example such extension is possible for the formalism of Stochastic Well-Formed Nets with the associated Symbolic Reachability Graph lumping method.

The Symbolic Reachability Graph (SRG) approach for SWNs relies on the possibility to automatically discover and exploit possible symmetries in the state transition graph (reachability graph): indeed from the SWN representation of the above system the SRG generation algorithm automatically chooses the correct level of state description detail. This is done by defining the color class of servers ($Lines$), with one color for each of the four parallel servers in the second stage. The clients flowing in the system are represented by tokens in the places. The indistinguishable tokens in place $Line0$ represent customers in the first server. The tokens in place $Line1$ represent customers in the four parallel servers and take the color that corresponds to the queue that they join. The natural detailed state description for a state in this net is given by the distribution of tokens (with associated color) in the places. So for example the state $\langle 1,2,1,1,0\rangle$ would be represented as $Line0(1)$ $Line1(2\langle S1\rangle, 1\langle S2\rangle, 1\langle S3\rangle)$. Clearly this state representation is equivalent to the detailed one, previously defined for the queueing description[3]. The event associated with $choice(Si)$ is the choice of joining the queue of server $Si,\ i = 1, \ldots, 4$ after exiting $S0$ (this event is supposed to take no time). The event associated with $Serv1(S_i)$ is the completion of a service by S_i.

The peculiar aspect of SWNs is the symmetry property that defines symbolic markings. The equivalence classes induced by this relation are the state aggregates to be used in the lumping process. The justification for this automatic state grouping is due to the fact that objects in the same class (the servers class $Lines$ in our examples) will surely behave homogeneously due to the definition of SWN dynamics. Color classes can be partitioned into subclasses when it is necessary to specify different behaviour of similar objects (for example $Lines$ is partitioned in $Lines_1$ and $Lines_2$ to distinguish between fast and slow servers). When this happens, the permutations used to define the equivalence relation are restricted to permute objects within the same subclass.

In our example with different speed servers, the servers class must be partitioned into two subclasses to distinguish between the faster and slower servers. This is enough to properly restrict the extent of automatic state aggregation. However, it might be useful to distinguish the case in which a partition into subclasses is used only to specify different transition rates and identical token flow from the case in which the partition into subclasses is used to specify differences in the token flow. In fact in the former case the underlying MC has a quasi-lumpable structure.

[3]Place $Choice$ can be ignored in the state description because tokens spend no time in it since transition $Make\_choice$ takes no time to fire; the place simply represents the choice point where a neutral token decides which server color to take before entering place $Line1$.

The bounding method also applies to systems were the departure/arrival rates are state dependent and are a smooth function of the system population. In this case an aggregate is naturally defined as a set of states whose associated population is within a given range.

When a reward process is associated with the CTMC, then the reward function definition should also be taken into account when checking lumpability (see [31]). A sufficient (though strong) condition is that the reward function gives the same value for every state in an aggregate. We'll see how this condition can be relaxed in our framework of bounding.

3.3 Bounds for Quasi-lumpable Markov Chains

In this section we first summarize the Courtois and Semal's results that we are going to use; then we show how a quasi-lumpable MC can be transformed into a lumpable one and how Courtois and Semal's methods can be applied to it.

3.3.1 The Courtois and Semal Bounding Methods

In this section we summarize Courtois and Semal's results [29, 28] that are relevant to our discussion. The results reported here are specific to stochastic matrices, and are thus less general than those presented in [29, 28]; the reader can refer to the original papers for the more general theorems and their proofs (which are for non-negative matrices).

Let P be a stochastic matrix, and let P^- and P^+ be a lower and upper bound (componentwise) for P respectively. We assume that P, P^- and P^+ are all irreducible.

Lemma 3.3.1 *[29]*

- $(I - P^-)$ *is non-singular*
- $(I - P^-)^{-1} \geq 0$
- $(I - P^-)^{-1}\mathbf{e}^{\,T} > 0$

where $\mathbf{e}^{\,T}$ *denotes a properly dimensioned column of ones.*

Theorem 3.3.1 *[29] Let P be a stochastic matrix, and let P^- be a lower bound for P. We assume that both P and P^- are irreducible. The steady state probability π of P belongs to the polyhedron whose vertices are the rows $\{z_j\}$ of the matrix Z with indices in the set $J = \{j \in 1,\ldots,n,\ s.t.\ \exists i : P_{i,j} > P^-{}_{i,j}\}$*
where

$$Z = \Omega^{-1}(I - P^-)^{-1}$$

and Ω is a diagonal normalization matrix defined as follows:

$$\Omega = diag((I - P^-)^{-1}\mathbf{e}^{\,T})$$

Hence there exists a vector β s.t.

- $\beta\ \mathbf{e}^{\,T} = 1$
- $\beta_j = 0,\ j \notin J$
- $\pi = \beta\ Z$

and

$$\forall i,\ \pi_i^- \leq \pi_i \leq \pi_i^+$$

where

$$\pi_i^- = max\{min_{k \in J}(Z)_{k,i}; 1 - \sum_{j \neq i} max_{k \in J}(Z)_{k,j}\}$$

and

$$\pi_i^+ = min\{max_{k \in J}(Z)_{k,i}; 1 - \sum_{j \neq i} min_{k \in J}(Z)_{k,j}\}$$

The following theorem gives an alternative way for obtaining bounds without computing inverses.

Theorem 3.3.2 *[29] Given a lower bound P^- for P, the k^{th} row of matrix Z is the steady state probability vector of the matrix obtained by incrementing the k^{th} column of P^- in order to make it stochastic.*

Thus if we have only a lower bound P^- of the transition probability matrix of a given DTMC (let's assume that the chain has n states), we can still compute bounds for its steady state probability vector π. This is done by computing the solution of n DTMCs; the transition probability matrix P_i of the i^{th} DTMC is obtained from the substochastic matrix P^- by increasing the elements of column i enough to make P_i stochastic. Let π^i be the steady state probability vector solution of the i^{th} DTMC. The lower (upper) bound on the steady state probability of state k is computed as $min_i\ \pi_k^i$ $(max_i\ \pi_k^i)$.

The next theorem is analogous to Theorem 3.3.1 when an upper bound P^+ of P is available instead of a lower bound. However this result is slightly weaker because the inverse is not guaranteed to exist; furthermore, even when it does exist some additional constraints have to be tested in order to derive bounds from it (i.e., there is no result similar to Lemma 3.3.1 in this case).

Theorem 3.3.3 *[29] Let P be a stochastic matrix, and let P^+ be an upper bound for P. We assume that both P and P^+ are irreducible. If the following inverse matrix*

$$(I - P^+)^{-1}$$

exists and satisfies

$$(I - P^+)^{-1}\mathbf{e}^{\ T} < \mathbf{0}^{\ T} \qquad \mathbf{e}\ (I - P^+)^{-1} < \mathbf{0}$$

then there exists a vector β s.t.

- $\beta\mathbf{e}^{\ T} = 1$
- $\pi = \beta\Omega^{-1}(I - P^+)^{-1}$

where Ω^{-1} is a diagonal normalization matrix.

In [29, 28] two applications of the above theorems are presented.

1. Computation of bounds on conditional steady state probability of a subset of states $\mathcal{S}$ in a DTMC when the transition probability among the states in the subset is known while only partial or no information is available about the transition probability between states in $\bar{\mathcal{S}}$ and states in $\mathcal{S}$. Of course the amount of information available affects the accuracy of bounds.

   The less accurate and at the same time less computationally expensive way of obtaining these bounds is to applying Theorem 3.3.1 using the submatrix $P_{\mathcal{S},\mathcal{S}}$ of transition probabilities among states in $\mathcal{S}$ as P^-.

2. Computation of bounds on the steady state probability vector of a large system by decomposition into smaller subsystems: this is called the *bounded aggregation method*. This method consists of two steps:

   - computation of bounds on the conditional state probabilities within each aggregate using the method just explained;
   - computation of bounds on the probability of being in each aggregate. This part is divided into two steps: (1) derivation of a lower bound $\mathcal{P}^-$ for the inter-aggregate probability matrix $\mathcal{P}$ using the results of previous step; (2) application of Theorem 3.3.1 to $\mathcal{P}^-$ to compute bounds on the aggregates probability vector.

   Observe that if we knew the vector of conditional probabilities $\pi^{\mathcal{S}}$ of the states in aggregate $\mathcal{S}$, then we could easily compute the transition probability $\mathcal{P}_{\mathcal{S},\mathcal{S}'}$ between aggregates $\mathcal{S}$ and $\mathcal{S}'$ as $\sum_{s,s':s\in\mathcal{S},s'\in\mathcal{S}'} \pi_s^{\mathcal{S}} P_{s,s'}$. Since we know only bounds on $\pi^{\mathcal{S}}, \forall\mathcal{S}$, we are able to compute only a bound $\mathcal{P}^-$ for $\mathcal{P}$.

These techniques can be used to compute bounds not only on the steady state probability vector, but also on any kind of a performance metric defined in terms of a reward function r from the set of states to $I\!R$. Let r_j denote the reward associated with state j, and Let π^i denote the steady state probability vector obtained by solving the DTMC characterized by the matrix P_i obtained from P^- by increasing its i^{th} column such as to make it stochastic. The mean accumulated reward $\bar{R}$ is defined as

$$\bar{R} = \pi r^T = \sum_i \pi_i r_i$$

and can be rewritten as

$$\bar{R} = \sum_i (\sum_j \beta_j \pi_i^j) r_i = \sum_j \beta_j \sum_i \pi_i^j r_i.$$

The term $\sum_i \pi_i^j r_i$ is equal to the mean accumulated reward $\bar{R}_j = \pi^j r^T$ computed from $P^-{}_j$ and

$$\bar{R} = \sum_i \beta_i \bar{R}_i$$

so that $min_i \bar{R}_i$ and $max_i \bar{R}_i$ are a lower and an upper bound for $\bar{R}$ respectively.

3.3.2 The proposed method

Transforming a Quasi-Lumpable MC into a lumpable MC The definition of a quasi-lumpable MC contains the first hint on how to proceed in transforming the quasi-lumpable chain into a lumpable one: with reference to matrix P in Figure 3.1, if the values of ϵ were small enough we could be tempted to just use $P^- + diag(P^\epsilon \mathbf{e}^{\ T})$ instead of P and obtain an approximate result from the lumped version of this new matrix.

Even if this method leads to a good approximation in some cases, in general it is difficult to forecast the actual influence of the approximation on the result both from a quantitative point of view (how far is the approximation from the actual result?) and a qualitative point of view (is the approximation an upper or a lower bound with respect to the correct result?).

Instead of just "forgetting" about the elements in P^ϵ, we could imagine redirecting these "*disturbing*" transition probabilities (or rates) to a new extra state s. By doing this we end up with a new stochastic matrix P_s that has one more state than P^-. The new matrix is depicted in Figure 3.3; vectors $\mathbf{x}$ and $\mathbf{y}^T$ are defined as follows:

- $\mathbf{y}^T = P^\epsilon \mathbf{e}^{\,T} = (P - P^-)\mathbf{e}^{\,T}$
  where $\mathbf{e}^{\,T}$ is a column vector of 1s, of appropriate dimension; $\mathbf{y}^T$ is a column vector whose elements are the row sums of P^ϵ (hence the interpretation of element i of vector $\mathbf{y}^T$ as the global probability of going from state i to state s due to a redirection of probability);

- $\mathbf{x}$ is a row vector of the same dimension as a row of P^-; right now we do not define its contents but we do impose the constraint $\mathbf{x}\mathbf{e}^{\,T} = 1$.

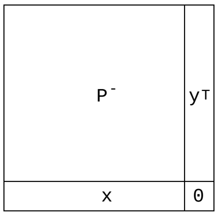


Figure 3.3: Modified matrix P_s.

We claim that there exists an assignment of values to the elements of $\mathbf{x}$ such that the steady state probability of MC P is equal to the conditional probability of the first n states[4] of chain P_s. In other words, if we redistribute "properly" the transition probabilities from the extra state to the states that have had a decrement in the incoming transition probabilities, we end up with the same conditional probability distribution as in the original model.

Theorem 3.3.4 *Let P be the transition probability matrix of an ergodic DTMC with n states. Let P^- be a lower bound (componentwise) for P, i.e., $P^- \leq P$. Let $\mathbf{y}^T = (P - P^-)\mathbf{e}^{\,T}$ and finally let P_s be the stochastic matrix of Figure 3.3.*

There exists a vector $\mathbf{x}$ such that the conditional steady state probabilities for the first n states of P_s is equal to the steady state probabilities of DTMC P.

Proof: The equations for the computation of the steady state probabilities in the derived chain characterized by the state transition matrix P_s are

$$\begin{cases} (\pi', a)P_s = (\pi', a) \\ (\pi', a)\mathbf{e}^{\,T} = 1 \end{cases} \tag{3.1}$$

Let's assume that π is the vector satisfying the equations

$$\begin{cases} \pi P = \pi \\ \pi \mathbf{e}^{\,T} = 1 \end{cases}$$

i.e., π is the steady state probability vector of the original problem.

The first equation in system (3.1) can be rewritten as the system

$$\begin{cases} \pi' P^- + a\mathbf{x} = \pi' \\ \pi' \mathbf{y}^T = a \end{cases}$$

Substituting $\pi'\mathbf{y}^T$ for a in the first equation we get

$$\pi' P^- + \pi' \mathbf{y}^T \mathbf{x} = \pi' \tag{3.2}$$

[4] n is the number of states in the original chain P.

We prove the existence of vector $\mathbf{x}$ by showing that

$$\mathbf{x} = \frac{\pi(P - P^{-})}{\pi(P - P^{-})\mathbf{e}^{\,T}} \tag{3.3}$$

satisfies the requirements stated in the theorem i.e., when $\mathbf{x}$ is defined as in (3.3), $\pi' = c\pi$ (where c is the normalization factor $c = \pi'\mathbf{e}^{\,T}$).
Substituting the proposed expression for $\mathbf{x}$ in equation (3.2) we get:

$$\pi' P^{-} + \pi' \mathbf{y}^T \frac{\pi(P - P^{-})}{\pi(P - P^{-})\mathbf{e}^{\,T}} = \pi'$$

Now let's verify that $c\pi$ is a solution for this equation:

$$c\pi P^{-} + c\pi \mathbf{y}^T \frac{\pi(P - P^{-})}{\pi(P - P^{-})\mathbf{e}^{\,T}} = c\pi$$

by substituting $\mathbf{y}^T$ with its definition we get:

$$c\pi P^{-} + c\pi (P - P^{-})\mathbf{e}^{\,T} \frac{\pi(P - P^{-})}{\pi(P - P^{-})\mathbf{e}^{\,T}} = c\pi$$

after some simplifications the equation is transformed into

$$\pi P = \pi$$

which holds true by hypothesis[5]. Hence when $\mathbf{x}$ is defined as in (3.3), $(c\pi, 1 - c\pi\mathbf{e}^{\,T})$ is indeed a solution for the derived system (3.1). □

At this point we can use Courtois and Semal's technique to compute bounds on the conditional probabilities of a subset of states in a system when only the transition probabilities among the states in the subset are known [28]. In this case the subset of states in the system represented by P_s is the set of original states, and the rest of the system is the extra state. Since the bounds can be computed without knowledge of the transition probabilities from the rest of the system to the selected subset of states there is no problem in not knowing the values of the elements of $\mathbf{x}$. However if something is known about $\mathbf{x}$ then better bounds can be computed. Some information about $\mathbf{x}$ can be easily derived from P^{ϵ}, namely any column j of P^{ϵ} with all zero elements, corresponds to a zero element in $\mathbf{x}$. Theorem 3.3.1 takes into account this factor by defining the set J of Z rows that contribute to the computation of bounds.

So far, the lumpability property of P^{-} has not been exploited. Observe that the MC P_s in general does not satisfy the lumpability conditions with respect to a given $\mathcal{A}$.

For example matrix P_s depicted below and derived from matrix P of Figure 3.1 is not lumpable.

$$P_s = \begin{array}{|c|c|cc|cc|c|c|}
\hline
 & 1 & 2 & 3 & 4 & 5 & 6 & s \\
\hline
1 & \bullet & 2\lambda_1 & \lambda_2 & & & & \\
\hline
2 & & \bullet & & \lambda_1 & \lambda_1 & & \epsilon \\
3 & & & \bullet & & 2\lambda_1 & & \\
\hline
4 & & & & \bullet & & \lambda_1 & \epsilon \\
5 & & & & & \bullet & \lambda_1 & \\
\hline
6 & \mu & & & & & \bullet & \\
\hline
s & \multicolumn{6}{c|}{x} & 0 \\
\hline
\end{array}$$

[5]Constant c can be found using the normalization equation $c + a = 1$; furthermore the solution $\pi' = c\pi$ is unique since if P is irreducible, so is P_s.

But now let's modify the column $\mathbf{y}^T$ so that the lumpability conditions are satisfied. This can be done by increasing some of the elements of $\mathbf{y}^T$ and decreasing the matrix diagonal elements by the same quantity[6].

$\tilde{P}_s =$

	1	2	3	4	5	6	s
1	$\bullet$	$2\lambda_1$	λ_2				
2		$\bullet$		λ_1	λ_1		ϵ
3			$\bullet - \epsilon$		$2\lambda_1$		ϵ
4				$\bullet$		λ_1	ϵ
5					$\bullet - \epsilon$	λ_1	ϵ
6	μ					$\bullet$	
s				x			0

$\mathcal{P}_s =$

	a1	a2	a3	a4	s
a1	$\bullet$	$2\lambda_1 + \lambda_2$			
a2		$\bullet$	$2\lambda_1$		ϵ
a3			$\bullet$	λ_1	ϵ
a4	μ			$\bullet$	
s			x		0

The modified matrix $\tilde{P}_s$ is lumpable ($\mathcal{P}_s$ is the lumped chain) and the conditional probabilities of the aggregate states are still bounds for the sum of the probabilities in the aggregates computed on the original model. This statement is true because in the above proof we didn't make any assumption about P^- and P^ϵ other than $P^- \leq P$. In the sequel, n' is used to denote the number of aggregate states (i.e., $n' = |\mathcal{A}|$) and $\mathcal{S}$ is used to denote the first n' states of $\mathcal{P}_s$. Courtois and Semal's method can now be applied to the new lumped model as well.

Bounds on accumulated reward Often the performance indices of interest are not the steady state probabilities, but rather the accumulated reward given a reward function r from the set of states to $I\!R$. When the bounding method proposed has to be applied, it may be the case that aggregated states do not have the same associated reward. Of course if every state in aggregate ai had the same reward r_{ai}, then the aggregate reward would be r_{ai}, otherwise the computation of the exact reward associated with the aggregate would require the knowledge of the conditional probability distribution π^{ai} of the states in the aggregate:

$$r_{ai} = \sum_{s \in ai} \pi_s^{ai} r_s.$$

In this case the method described in Section 3.3.1 for the computation of bounds on the mean accumulated reward needs some minor adjustment. For each aggregate ai we have to compute the minimum and maximum reward rate:

$$r_{ai}^- = min_{s \in ai} r_s$$

$$r_{ai}^+ = max_{s \in ai} r_s$$

then instead of computing $\bar{R}_i$ we compute $\bar{R}_i^+$ using for each aggregate the maximum reward and $\bar{R}_i^-$ using for each aggregate the minimum reward. The lower and upper bounds for $\bar{R}$ are then derived as $min_i \bar{R}_i^-$ and $max_i \bar{R}_i^+$ respectively. Notice that this is the best we can do if we don't have any information about the conditional probability of the states in each aggregate, but this approach could result in loose bounds.

Interpretation in terms of Bounded Aggregation The previous method can be interpreted in a different way, as a simplified version of the bounded aggregation technique [29]. However, the former interpretation is useful for devising bound improvement methods as we'll see in the next section.

The bounded aggregation method consists of two steps:

[6]Observe that the diagonal element could become negative by subtracting ϵ. To overcome this problem we can scale the diagonal of the matrix by performing the following transformation preserving the steady state probability result π

$$P \rightarrow \alpha P + (1 - \alpha) I$$

where $0 < \alpha \leq 1$.

1. computation of bounds on the conditional state probabilities within each aggregate.

2. computation of bounds on the probability of being in each aggregate.

The bounds obtained in the two steps are then combined to obtain bounds on the steady state probability vector over the original set of states.

We are actually interested in computing only the probability vector of the aggregates (second step) since our aim is to deal directly with the lumped process. The $n' \times n'$ upper-leftmost submatrix $\mathcal{P}^-$ of the lumped matrix $\mathcal{P}_s$ derived in the previous paragraph is a lower bound for the correct inter-aggregates transition probability matrix $\mathcal{P}$ associated with P given state partition $\mathcal{A}$; indeed the correct transition probability between aggregate I and aggregate J is a convex combination of the rowsums in the submatrix $P_{I,J}$ (where the weight used for each row in the submatrix is the conditional probability of the corresponding state in aggregate I). In our approach we are giving a weight of one to the minimum rowsum, thus obtaining a lower bound with respect to the correct convex combination of the rowsums. In our example, matrix $P_{a2,a3}$ is

	4	5
2	λ_1	λ_2
3		$2\lambda_1$

and the transition probability between $a2$ and a_3 in $\mathcal{P}^-$ is $min(2\lambda_1, \lambda_1 + \lambda_2)$ while the corresponding value in $\mathcal{P}$ would be $2\alpha\lambda_1 + (1-\alpha)(\lambda_1 + \lambda_2)$ for some $0 \leq \alpha \leq 1$.

Observe that an upper bound, $\mathcal{P}^+$, for the probability between two aggregates could be computed as well by choosing the maximum among the rowsums in the submatrix (in the example above $max(2\lambda_1, \lambda_1 + \lambda_2)$ is an upper bound for the transition probability between $a2$ and $a3$).

3.3.3 Application of the method to the multiserver example

The model of Figure 3.2 is a good candidate for the application of the method. In fact this is a typical case in which the partition of the servers in two subclasses with equal speed within each subclass leads to a quasi-lumpable MC structure. We assume that the rates λ_1 and λ_2 associated with the two classes of servers are $\lambda_1 = 1.00$ and $\lambda_2 = 1.01$.

In Figure 3.6 the aggregated matrix (split into two parts due to space constraints) and the vector $\mathbf{y}^T$ are shown . All we know about $\mathbf{x}$ is that its 13^{th} element must be zero (because the corresponding column is not modified to get $\mathcal{P}^-$ from P). In Figure 3.4 the description of the states for the aggregated model is given. The state description for the complete model is not given here: it is the intermediate detail level representation $R2$ presented in Section 3.2. The reward rates for all aggregates used to compute the mean queue length at the entrance of the multiple server ($S1 - S4$) and the throughput[7] out of $S0$ are shown in Figure 3.5. Observe that in this example we were able to compute an exact reward for each aggregate because all the aggregated states have the same associated reward.

Solving the 17 (aggregated) models (each with only one element of $\mathbf{x}$ equal to 1 and the others equal to 0) we obtain the following steady state probabilities, mean queue length, and throughput bounds. The table also contains the correct values computed on the complete model (that has 50 states).

[7] μ is the service rate of $S0$.

State #	Description (Clients in each server)	State #	Description (Clients in each server)
1	Serv1: 5, Serv2: 0	10	Serv1: 1. Serv2: 2 q of 2
2	Serv1: 4, Serv2: 1 q of 1	11	Serv1: 1, Serv2: 1 q of 1 & 1 q of 3
3	Serv1: 3, Serv2: 2 q of 1	12	Serv1: 1, Serv2: 1 q of 4
4	Serv1: 3, Serv2: 1 q of 2	13	Serv1: 0, Serv2: 3 q of 1 & 1 q of 2
5	Serv1: 2, Serv2: 3 q of 1	14	Serv1: 0, Serv2: 1 q of 1 & 2 q of 2
6	Serv1: 2, Serv2: 1 q of 1 & 1 q of 2	15	Serv1: 0, Serv2: 2 q of 1 & 1 q of 3
7	Serv1: 2, Serv2: 1 q of 3	16	Serv1: 0, Serv2: 1 q of 2 & 1 q of 3
8	Serv1: 1, Serv2: 4 q of 1	17	Serv1: 0, Serv2: 1 q of 1 & 1 q of 4
9	Serv1: 1, Serv2: 2 q of 1 & 1 q of 2	18	Serv1: 0, Serv2: 1 q of 5

Figure 3.4: States description.

State #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	1 6	17	18
MQL	0	1	2	2	3	3	3	4	4	4	4	4	5	5	5	5	5	5
Throughput	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ	μ	0	0	0	0	0	0

Figure 3.5: Reward rates.

	Lower bound	Exact value	Upper bound
Prob(State 1)	0.307430	0.321631	0.329239
Prob(State 2)	0.307430	0.320039	0.323022
Prob(State 3)	0.115106	0.119420	0.122238
Prob(State 4)	0.078126	0.079616	0.084354
Prob(State 5)	0.019179	0.019804	0.022648
Prob(State 6)	0.057889	0.059415	0.063442
Prob(State 7)	0.019131	0.019806	0.026669
Prob(State 8)	0.001198	0.001232	0.003351
Prob(State 9)	0.014373	0.014780	0.018140
Prob(State 10)	0.007139	0.007389	0.011374
Prob(State 11)	0.014219	0.014782	0.019938
Prob(State 12)	0.004703	0.004928	0.012848
Prob(State 13)	0.001194	0.001226	0.003879
Prob(State 14)	0.003562	0.003677	0.007472
Prob(State 15)	0.003544	0.003677	0.007506
Prob(State 16)	0.003527	0.003677	0.009229
Prob(State 17)	0.003506	0.003677	0.009698
Prob(State 18)	0.001164	0.001226	0.012440
Mean Queue Length	1.250705	1.273430	1.385868
Throughput	0.967612	0.98284	0.983502

Figure 3.6: The aggregated matrix $\mathcal{P}$ and the corresponding vector $\mathbf{y}^T$.

The spread of the bounds for the mean queue length and throughput is 11% and 1.6% respectively.

The example model satisfies the constraints for the applicability of Theorem 3.3.3 that exploits the upper bound $\mathcal{P}^+$, so that we could use that theorem to obtain improved bounds. In Section 3.4 we show a technique for bounds improvement that doesn't require the computation of the inverse and the verification of constraints to be applied.

The reader may object that good bounds for the throughput and mean queue length of this specific example could be found by just observing that the (actually lumpable) model obtained by setting the service time of all the servers in the multiple server to the minimum among the service times in the original model, gives an upper bound for the throughput and a lower bound for the queue length. If the maximum among the service times in the original model is assigned to the servers, then we get a lower bound for the throughput and an upper bound for the queue length. This can be proved by induction on the number of customers in the system. Note however that it is not possible to compute bounds on the state probability distribution in the same simple way. Another possible objection is that the model is a product form queueing network, and efficient algorithms exist for the computation of exact results, however if we changed the policy for choosing the queue to join in the multiserver to a load dependent one (e.g., shortest-queue), product form solutions would not be applicable any more, while our method still applies. This example was chosen on purpose for its simplicity, in general it is not easy to prove that by lowering/increasing a rate in the system we get a bound on a given performance index. In Section 3.5 a more complex example is presented in which the presence of synchronization among the system components makes it difficult to find any intuitive argument for simple bounds computation.

3.3.4 Discussion on the computational advantages of the method

In this section we'll discuss the proposed method from the point of view of its convenience in terms of computational complexity. The main advantage of the method is that it allows us to deal with a smaller system than the one we start with, however there is a price we have to pay to be able to work with the reduced model. In fact there are at least two kinds of overhead we have to take into account in evaluating the method:

- derivation of the reduced matrix;
- computation of bounds.

Concerning the derivation of the reduced matrix, the ideal situation is to derive the lumped matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ directly from the high level model without having to compute (or at least to store) the large complete matrix. We'll show how this can be done when the high level formalism used is that of SWNs.

The computational complexity issue insteads, is related to the fact that the computation of bounds requires the solution of n' matrices of dimension $n' \times n'$ in the worst case. Later in this section we'll discuss how much smaller the lumped matrix must be for the method to be more efficient from the time complexity point of view, considering also the possibility of solution parallelization. The space complexity issue is discussed as well.

The issue of automatic construction of the lumped matrix is closely related to that of the detection of the symmetries on the state space that induce a partition into state aggregates. The same problem has to be faced when exact lumping methods are used, indeed our bounding methods can be considered as an extension of the existing exact lumping methods.

Automatic construction of a lumped matrix from a SWN model The color classes are used to define sets of "similar" objects and the partition of a class into subclasses is used to identify subsets of objects in a class that share the same behaviour. As already pointed out, it is possible to distinguish between two possible situations: (1) the objects in different subclasses have different *qualitative* behaviour, i.e., they cannot play the same "role" in the system because they have different possible evolutions, (2) the objects in different subclasses have the same *qualitative* behaviour, but the event sequences happen with different rates depending on which subclass the object belongs to.
For example, in the model of Figure 3.2 the colored tokens representing customers in the "multiserver" station follow the same route through transition end_serv2, independently of their color, however, the firing rate of the transition depends on the token color.

Hence, given a SWN model of the system under study, a first analysis is needed to detect all the subclasses in each class with similar "qualitative" behaviour. This permits automatic determination of the candidate quasi-lumpable state aggregates. Then two approaches are possible, (1) apply the Symbolic Reachability Graph generation algorithm to the system as it is specified and use the information about aggregate states to compute the lumped matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ in a second step; (2) apply the SRG generation algorithm to a modified system with some of the homogeneously behaving subclasses merged in such a way that $\mathcal{P}^-$ and $\mathcal{P}^+$ are directly derived.

The two algorithms that follow are respectively the subclasses merge algorithm and the modified model construction algorithm, where the modified model is the one from which the aggregate matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ can be directly derived.

Subclasses merge algorithm For each basic color class, the following static classes merge procedure has to be performed:

$old_static_i \ = \ \langle$set of static subclasses in the original $C_i\rangle$
$new\_static_i \ = \emptyset$
while $old\_static_i \ \neq \ \emptyset$ do
    $\langle$ remove a subclass sc form $old\_static_i\rangle$
    $lsc = sc$
    $sc\_list = emptylist$
    append(sc, sc_list)
    for each $sc' \in old\_static_i$ do
        if $semilumpable(sc, sc')$ then
            $lsc = lsc \cup sc'$
            $\langle$ remove sc' form $old\_static_i\rangle$
            append(sc', sc_list)
        end if
    end for
    $old\_sc\_list[lsc] = sc\_list$
    $\langle$ add static subclass lsc to $new\_static_i\rangle$
end while

A set of static subclasses old_static_i is the input for this procedure. The output is a new set new_static_i, of static subclasses. The cardinality of the new set is less than or equal to that of the old set. For each new static subclass lsc, a list $old\_sc\_list[lsc]$ of the old static subclasses that have been merged into lsc is maintained. Function $semilumpable(sc, sc')$ returns $true$ iff for all transitions that have C_i in their color domain definition, no predicates of type $d(X_i^j) = sc$, $d(X_i^j) = sc'$, $d(X_i^j) = d(X_i^k)$ or their negation are used either in the transition predicate or in the function predicates labeling the arcs connected to the transition (i.e., neither the transition enabling conditions nor the transition state change can depend on the fact that some element in the transition color instance belong or not to sc or sc'). Observe that more complex but less restrictive conditions may be defined: we might for example allow the presence of predicate $d(X_i^j) = sc$ provided that it is in *or* with $d(X_i^j) = sc'$.

Aggregate model computation algorithm The aggregate matrices $\mathcal{P}^-$ and $\mathcal{P}^+$ are computed by applying the usual SRG generation algorithm to a modified model. The modified model has the same structure as the original one, while it differs from it in the static subclasses definition and, as a consequence, in the transition firing rates.

In a SWN, the transition firing rate of a generic transition t is defined as a function θ_t from tuples of static color classes to reals.

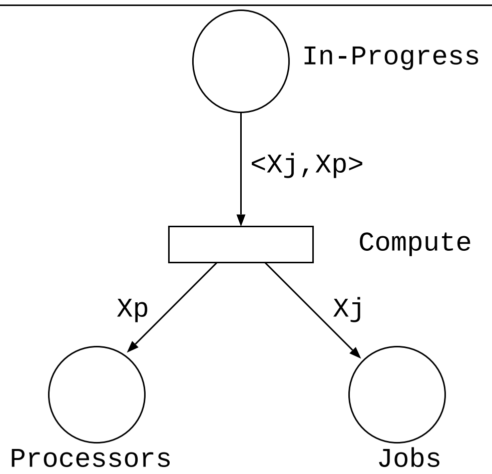


Figure 3.7: A simple SWN transition example.

For example transition *compute* in Figure 3.7 takes a token with its two component color from its input place: the first component represents a processor and the second represents a job. If the jobs class J is divided into two subclasses of short jobs J_s and long jobs J_l and the processors are divided into two subclasses of fast processors P_f and slow processors P_s, then a possible definition for the transition rate could be: $\theta_t(\langle < P_s, J_s > \rangle) = 0.3$, $\theta_t(\langle < P_f, J_s > \rangle) = 1$, $\theta_t(\langle < P_s, J_l > \rangle) = 0.15$, $\theta_t(\langle < P_f, J_l > \rangle) = 0.5$.

When a transition instance $t(z)$ is fired, its rate $\lambda(t(z))$ is computed as follows: (1) derive from z the tuple d of corresponding static subclasses (each dynamic subclass is associated with exactly one static subclass, see [6]; (2) $\lambda(t(z)) = m\ \theta_t(d)$ where m is a factor that depends on the cardinality of both the subclasses in z and in d. The reason for the multiplicative factor is that a symbolic firing instance is an aggregation of m regular firing instances all with the same rate $\theta_t(d)$. In our example a possible symbolic instance for transition $Compute$ could be $Compute(Z_P^1, Z_J^2)$ with $|Z_P^1| = 2, Z_P^1 \in P_s$, and $|Z_J^2| = 1, Z_J^2 \in J_l$ meaning that there are 2 $(= |Z_P^1|)$ slow processors each of which is processing 1 $(= |Z_J^2|)$ long job. This symbolic instance stands for 2 "ordinary" instances that are grouped in the lumping process.

In the following we show how to derive the modified model to compute the aggregate matrices given a specific aggregation of the static subclasses. We denote $D_{i,j}$ the new static subclasses and $\{D_{i,j}^{l}\}$ the set of the original subclasses that are aggregated into $D_{i,j}$.

The rates of the resulting new transitions, in general, will depend on the cardinality of the dynamic subclasses in the instantiation.

Let $t(z)$ be the transition instance for which a rate has to be computed. Let d be the associated static subclasses tuple. Let d' be the subtuple of d composed of aggregate static subclasses and z' the corresponding subtuple of z.

For each element $d_k = D_{i,j}$ in d', compute the set $\mathcal{Z}_k$ of possible partitions of dynamic subclass z_k into one or more new dynamic subclasses, each associated with a different static subclass $D_{i,j}^{l}$ of $D_{i,j}$ (see Figure 3.8).

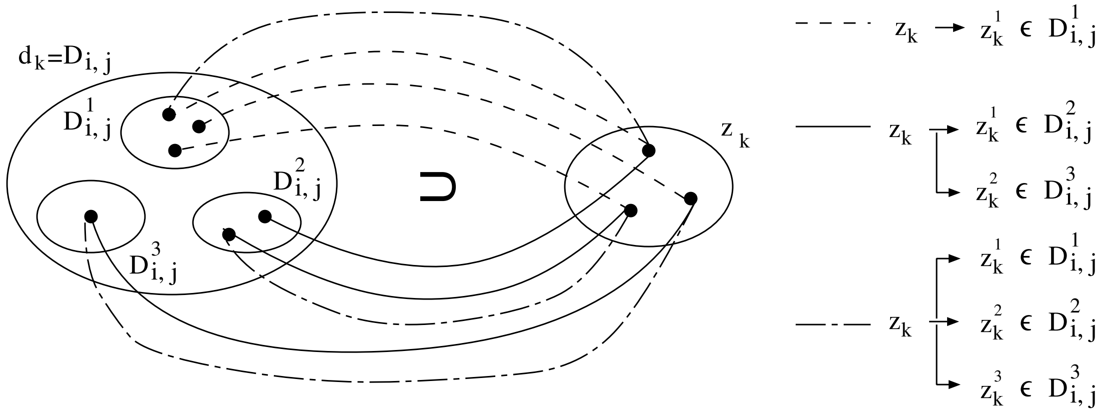


Figure 3.8: Possible partitions of z_k's with respect to the "merged" static subclass $D_{i,j}$.

The Cartesian product of the $\mathcal{Z}_k$s leads to a set of sums of original transition instances. From each sum a rate can be computed that correspond to a value for a rowsum in the aggregate transition represented by $t(z)$. The minimum and maximum in this set of rates gives the value for the corresponding transition rate in the aggregate matrices.

In the example of Figure 3.7 assume that $|P_s| = |P_f| = 2$ and $|J_s| = 2, |J_l| = 1$, and suppose we want to merge slow and fast processors into a unique class P and short and long jobs into a unique class J. Hence the symbolic transition instance $Compute(Z_P^1, Z_J^2)$ with $|Z_P^1| = 2, Z_P^1 \in P$ and $|Z_J^2| = 2, Z_J^2 \in J$ incorporates the following 6 possible instances with respect to the old classes partition.

- $Compute(Z_P^{1a}, Z_J^{1a}) + Compute(Z_P^{1a}, Z_J^{1b})$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_s$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$, $|Z_J^{1b}| = 1$ and $Z_J^{1b} \in J_l$; corresponding rate 0.9
- $Compute(Z_P^{1a}, Z_J^{1a}) + Compute(Z_P^{1a}, Z_J^{1b})$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$, $|Z_J^{1b}| = 1$ and $Z_J^{1b} \in J_l$; corresponding rate 3
- $Compute(Z_P^{1a}, Z_J^{1a})$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_s$, $|Z_J^{1a}| = 2$ and $Z_J^{1a} \in J_s$; corresponding rate 1.2
- $Compute(Z_P^{1a}, Z_J^{1a})$ with $|Z_P^{1a}| = 2$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$; corresponding rate 4
- $Compute(Z_P^{1a}, Z_J^{1a}) + Compute(Z_P^{1a}, Z_J^{1b}) + Compute(Z_P^{1b}, Z_J^{1a}) + Compute(Z_P^{1b}, Z_J^{1b})$ with $|Z_P^{1a}| =$

1 and $Z_P^{1a} \in P_s$, $|Z_P^{1b}| = 1$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 1$ and $Z_J^{1a} \in J_s$, $|Z_J^{1b}| = 1$ and $Z_J^{1b} \in J_l$; corresponding rate 3.3

- $Compute(Z_P^{1a}, Z_J^{1a}) + Compute(Z_P^{1b}, Z_J^{1a})$ with $|Z_P^{1a}| = 1$ and $Z_P^{1a} \in P_s$, $|Z_P^{1b}| = 1$ and $Z_P^{1a} \in P_f$, $|Z_J^{1a}| = 2$ and $Z_J^{1b} \in J_s$; corresponding rate 2.6

There is a computationally less expensive but not always accurate method for computing lower/upper bounds for the elements of $\mathcal{P}^-$ and $\mathcal{P}^+$. Given a transition t and a tuple d of aggregate static subclasses, compute the Cartesian product D of the sets of original subclasses in the component aggregate subclasses. Associate with d the $min(max)_{d' \in D}\theta_t(d')$. In this way the computed $\mathcal{P}^-$ and $\mathcal{P}^+$ are bounds for $\mathcal{P}$, but they can be very inaccurate in some cases. In the previous example this simplified method would have predicted correctly the maximum rate ($= 4$), while the minimum ($= 0.6$) would have been less than the the correct one (0.9).

Observe that even if the computation of the aggregate transition rates has a certain cost, it could be performed once and for all in a way that the result is easily reusable for models differing only in the actual parameter values.

Complexity of bounds computation The main goal of the proposed method is to cope with the problem of state space explosion by allowing some aggregation of the state space. However since the aggregate chain is only a bound for the actual aggregate chain and since the computation of bounds of a chain given its lower (and/or upper) bound transition probability matrix is more expensive than just computing the steady state solution of a chain of the same dimension, in some cases the method proposed could be computationally more expensive than the solution of the whole model we started with.

Let's consider the queueing network example we have been using throughout the paper. The complete model has 50 states while the aggregate model has 18 states. If we assume that the computation of the steady state solution of a MC of dimension n is $O(n^2)$ (we are assuming that sparse matrix algorithms can be applied, if this is not the case, the worst case complexity would be $O(n^3)$) then the time required for solving the complete model is proportional to $50^2 = 2500$, while the time required for getting bounds from the aggregate model is proportional to $17 * (18^2) = 5508$. Apparently the bounding method for this case is not computationally convenient, however the situation changes completely when the dimension of the studied system grows. The size of this model was chosen small to be able to show both the complete and aggregate transition matrices. Similar models with different number of customers, servers in the multiple server station and static subclasses in the class of servers, show how convenient the method can be as the size of the problem increases. This is shown in the table of Figure 3.9 where the number of states in the lumped (column a) and complete (column b) Markov chain and the corresponding estimated computational time cost as function of the number of servers and customers are listed. Notice that the number of states in the complete model refers to the case in which each queue in the multiserver has a different speed.

In general the complexity for computing bounds is $k(n')^2$ where k is the number of aggregate columns modified in the original MC to make it lumpable, hence in the worst case $k = n'$. In order for the method to be computationally convenient the ratio n/n' must be such that $k(n')^2 < n^2$.

Finally observe that even when the ratio between the time complexity of exact solution computation versus that of bounds computation is less than one, the computation of bounds can still be convenient. In fact the computation of bounds on the lumped model can be easily parallelized because it requires the solution of several smaller and completely independent (lumped) models, thus resulting in a gain

# Serv.	# Aggr. States	# Non Aggr. States	(a) Bounds Comp. Time	(b) Exact Sol. Comp. Time	(b)/(a)
5 customers					
4	18	126	5832	15876	2.72
5	19	252	6859	63504	9.26
6	19	462	6859	213444	31.12
8 customers					
4	53	495	148877	245025	1.64
5	60	1287	216000	1656369	7.6
6	64	1716	262144	2944656	11.23
10 customers					
4	94	1001	830584	1002001	1.21
5	113	3003	1442897	9018009	6.25
6	125	8008	1953125	64128064	32.83

Figure 3.9: Comparing time complexity for the computation of bounds vs. exact solution.

in time complexity. Breaking up the problem into smaller ones may be also crucial from the space complexity point of view since the whole model could not fit into the available main memory, preventing the solution if no virtual memory is provided or causing a loss of performance due to frequent paging if virtual memory is available.

3.4 Improving bounds

The bounds found with the method described above may have a different degree of tightness depending on the sensitivity of the solution to the transitions replaced by lower bounds. In this section a technique is presented that can be used to improve the bounds when those obtained by directly applying the method presented in Section 3.3 are too loose. In the previous section we mentioned that it is possible to derive an upper bound $\mathcal{P}^+$ for the transition probabilities between aggregates; once we have this upper bound it is possible to use Theorem 3.3.3 to compute another set of bounds that, combined with those obtained from the lower bound $\mathcal{P}^-$, may lead to tighter bounds. The problem is that in this case the inverse $(I - P^+)^{-1}$ is not guaranteed to exist, and even if it does exist some further conditions must be verified in order to be sure that the results obtained are indeed bounds. In the next paragraph we will see how the upper bound on the aggregate transition probability can be used in conjunction with the the steady state probability bounds π^-, π^+ previously computed to get better bounds on any Markov reward function (and hence also on the state probabilities).

Exploiting information about the "probability redistribution" vector $\mathbf{x}$ The following method is aimed at improving the bounds on the performance measures we are interested in; we assume that each performance metric is defined by associating a reward function r on states with the model. In particular a possible performance metric could be the steady state probability of one specific state s_j; in this case all the states are assigned a null reward except s_j which is assigned a reward $r_j = 1$.

If the exact vector $\mathbf{x}$ of "redistribution" probabilities was known and substituted into matrix $\mathcal{P}_s$, then the mean cumulative reward (conditioned on being in the first n' states) computed from $\mathcal{P}_s$ would be the correct value we are looking for.

The behaviour of the Markov chain described by $\mathcal{P}_s$ is the following: the evolution starts from one state in $\mathcal{S}$ and goes on until it reaches the extra state s, then the evolution starts again from a state

State #	1	2	3	4	5	6	7	8	9
x^+	0.350781	0.840613	0.609799	0.258352	0.07127	0.434729	0.08 62158	0.0042123	0.100793
State #	10	11	12	13	14	15	16	17	18
x^+	0.0733268	0.160317	0.048524	0.000000	0.0162282	0.0163021	0.0400884	0.0421257	0.0135091

Figure 3.10: Upper bound for vector x.

in $\mathcal{S}$ probabilistically chosen according to the probability distribution defined by $\mathbf{x}$ until it reaches s again. The accumulated reward between two successive visits to s depends on which state is chosen to start with when exiting s. Assume we have the list $s_{j1}, s_{j2}, \ldots, s_{jn'}$ of states ordered in such a way that the accumulated reward before exiting to s, given that we started in $s_{jm}, m = 1, \ldots, n'$, is greater than or equal to the accumulated reward before exiting to s, given that we started in $s_{jk}, k > m$.

An upper bound on the mean cumulative reward can be computed from model $\mathcal{P}_s$ by substituting for $\mathbf{x}$ a modified vector $\mathbf{x}'$ such that $\mathbf{x}'_{jm} \geq \mathbf{x}_{jm}$ $(1 \leq m \leq k \leq n')$ and $\mathbf{x}'_{jl} \leq \mathbf{x}_{jl}$ $(k < l \leq n')$.

A lower bound on the mean cumulative reward can be computed if in model $\mathcal{P}_s$ we substitute for $\mathbf{x}$ a modified vector $\mathbf{x}'$ such that $\mathbf{x}'_{jm} \leq \mathbf{x}_{jm}$ $(1 \leq m \leq k \leq n')$ and $\mathbf{x}'_{jl} \geq \mathbf{x}_{jl}$ $(k < l \leq n')$.

If we were able to estimate an upper bound for $\mathbf{x}$ this could thus be used to compute better bounds on the desired performance figures. Recall that the correct $\mathbf{x}$ is defined as follows:

$$\mathbf{x} = \frac{\pi(\mathcal{P} - \mathcal{P}^-)}{\pi(\mathcal{P} - \mathcal{P}^-)\mathbf{e}^{\,T}}$$

Hence the following vector $\mathbf{x}^+$ is an upper bound for $\mathbf{x}$:

$$\mathbf{x}^+ = \frac{\pi^+(\mathcal{P}^+ - \mathcal{P}^-)}{\pi^-(\mathbf{e}^{\,T} - \mathcal{P}^-\mathbf{e}^{\,T})}$$

where π^+ and π^- are upper and lower bounds on the steady state probability distribution (e.g., that were computed in a first step using the method presented in the previous section) and $\mathcal{P}^+$ is an upper bound on the inter-aggregate transition probability matrix.

Applying the above method to the queueing network example we obtain for the vector $\mathbf{x}$ the upper bound estimate in Figure 3.10.

The new (much improved) bounds computed for the mean queue length and throughput using this method are:

	Lower bound	Exact value	Upper bound	Spread %
Mean Queue Length	1.250705	1.273430	1.295075	3.5%
Throughput	0.980303	0.98284	0.983502	0.3%

3.5 An Application Example

In this section we present a more complex example in which the presence of synchronization constraints does not allow us to easily obtain bounds on the performance figures of interest using some other method.

The modeled system consists of a parallel program organized according to a master-slave computation paradigm. The program is mapped on a parallel architecture: we assume there is a processor for each process, that the processors are homogeneous, and that due to the type of the interconnection network the communication time between the master and the slaves is not homogeneous (some slaves are "closer" to the master than others). This is the case for the example of a master-slave program mapped onto a mesh architecture depicted in Figure 3.11. The processors are represented by squares, while the processes are represented by circles. In the figure both the physical channels (connecting the processors) and the logical channels (connecting the processes) are depicted.

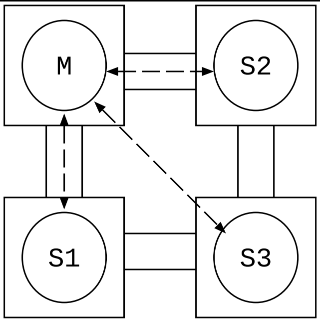


Figure 3.11: A Parallel Program Model.

The master divides the problem to be solved into several subproblems, and as long as there are free slaves it delivers the tasks to be executed to them. When no more free slaves are available, the master waits for some slave to complete its current task. The master also performs some computation in parallel with communicating, to process the results received and to prepare new data to be distributed.

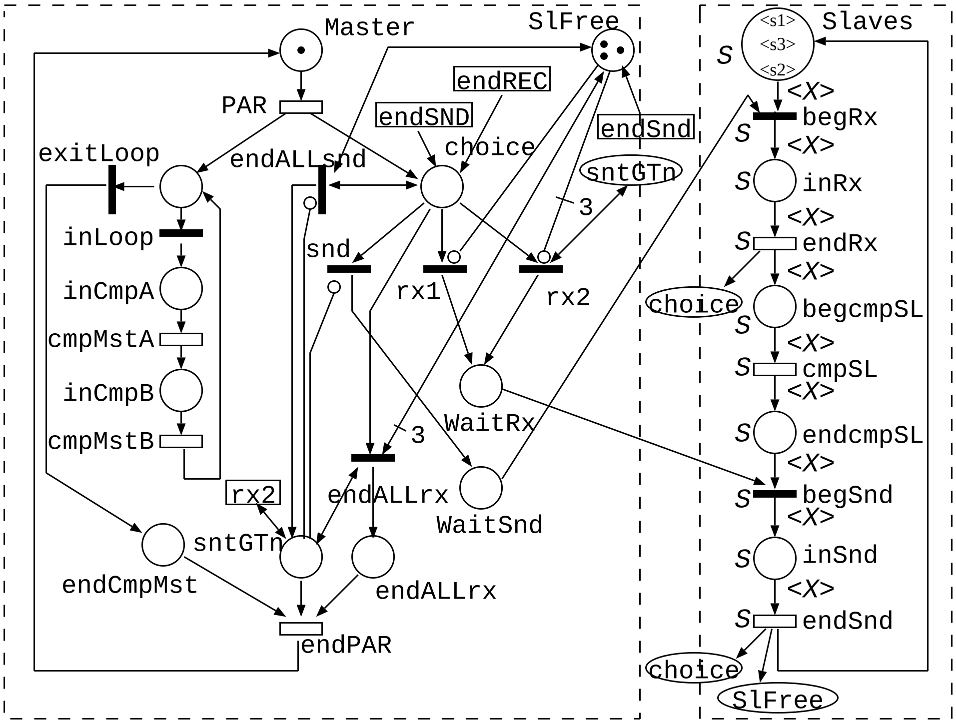


Figure 3.12: The SWN model of the Master-Slave Program.

We have modeled the master-slave system using the SWN formalism; in Figure 3.12 the resulting model is depicted. In this model the presence of a token in one place indicates the program counter position for the corresponding process, so that the next statement to be executed is represented by the transition pointed by the arc exiting the marked place.

The model is composed of two subnets, the first (leftmost) one represents the master's behaviour while the second (rightmost) one represent the common behaviour of the slaves. The distinction among the three slaves is obtained by using different basic colors. The behavioural model has been simplified as much as possible to make the picture readable. The master is composed of two parts that work in parallel: the computation part, consisting of a certain number of iterations of two procedures (transitions *inLoop*, *cmpMstA*, *cmpMstB*), and the synchronous communication part consisting of a certain number of iterations of send/receive operations (transitions *snd*, *rx1*, *rx2* plus the transitions representing actual communication, shared with the slaves net: *begRx* , *endRx* , *begSnd*, *endSnd*). The slaves behaviour can be described as repeated iteration of three operations: receive (transitions *begRx* and *endRx*), compute (transition *compSL*), send (transitions *begSnd* and *endSnd*). Only one color class is needed: the slaves class S. It has cardinality three and is divided into two static subclasses $S1$ of cardinality 2 and $S2$ of cardinality 1. Subclass $S1$ represents the slaves that are closer to the master while subclass $S2$ represents the farther slave. As a consequence the transitions representing communication between the master and the slaves have a rate that depends on the slave identity.

The detailed state representation for this model is a list of program counter values, one for each process in the program. The MC representing this model behaviour is quasi-lumpable with respect to the aggregation of states induced by considering the slaves as undistinguishable. In other words we aggregate all the states that are equal up to a permutation of slaves identities. For example we aggregate the state M_1 with slave $S1$ waiting to receive some data from the master, $S2$ sending a result to the master and $S3$ performing a computation, and state M_2 with slave $S2$ waiting to receive some data from the master, $S3$ sending a result to the master and $S1$ performing a computation. Observe that although this aggregation reflects a symmetry in the state space structure due to the homogeneous behaviour of the slaves, there is a perturbation in the transition rates, due to the fact that the communication between $S3$ and the master takes longer than the communication between the other two slaves and the master.

The quasi-lumpable model has 109 states and the number of aggregate states is 49.

The performance measures we have considered are three:

- throughput of the system, i.e. the number of completed computations for time unit (throughput of transition PAR representing the beginning of a new computation); acronym used in the table: THRUPUT;
- mean number of slaves waiting to receive a task from the master (i.e., mean number of tokens in place *Slaves*); acronym used in the table: MNSWRx;
- mean number of slaves waiting to send the result of their computation to the master (mean number of tokens in place *endcmpSL*); acronym used in the table: MNSWSnd.

Three experiments are reported in the following table changing the values of the difference between the rates associated with the communication between master and slaves (represented by transitions *endRx* and *endSnd*). We have fixed $\theta(endRx, S3) = 0.5$ and $\theta(endSnd, S3) = 3.5$ while the rates for $(endRx, S1-2)$ and $(endSnd, S1-2)$ that have been used in the three experiments are the following:

1. $\theta(endRx, S1) = 0.51$, $\theta(endSnd, S1) = 3.57$
2. $\theta(endRx, S1) = 0.525$, $\theta(endSnd, S1) = 3.675$
3. $\theta(endRx, S1) = 0.55$, $\theta(endSnd, S1) = 3.85$

Experiment #	Perf. Measure	Lower bound	Upper bound	Spread %
1	THRUPUT	0.012396	0.014595	18%
	MNSWSnd	0.079165	0.137257	73%
	MNSWRx	2.547911	2.726692	7%
2	THRUPUT	0.011936	0.016718	40%
	MNSWSnd	0.056813	0.180464	218%
	MNSWRx	2.436226	2.802309	15%
3	THRUPUT	0.011333	0.019553	73%
	MNSWSnd	0.039295	0.257986	557%
	MNSWRx	2.249367	2.861513	27%

The results shown in the table above are obtained from the direct method. We have also applied the bounds improvement method: it resulted in tightened bounds in the first experiment but it did not give any substantial improvement in experiments 2 and 3. As a matter of fact, the first case is also the only one that is eligible for the application of Theorem 3.3.3 of Section 3.3.

Clearly there is a strict correlation between the difference $P^+ - P^-$ and the spread in the bounds. It might be worthwhile to assess a priori the sensitivity of the system on variations on the transition probabilities that are positive in $P^+ - P^-$.

In this example we have also observed that some further bounds refinement could be obtained by the consideration of the high level model. If no contention for resources were present in the slaves subnet, then we would know for sure that the throughput of slave $S3$ is lower than the throughput of slaves $S1$ and $S2$. Since the slaves compete to communicate with the master, the throughput of both kind of slaves decreases with respect to the no contention case; moreover the difference between the throughput of the slave $S3$ and the other two slaves becomes smaller as the probability of contention increases. Our conjecture is that in the system we are studying, the throughput of slave $S3$ is less than or equal to the throughput of the other two slaves. Assuming that the conjecture is true (and it is indeed true for our experiment cases), we show how this information can be used to improve the estimate of the lower bound matrix $\mathcal{P}^-$. Of course, the kind of property to be proved about the high level model in order to obtain improved estimates of $\mathcal{P}^-$ and/or $\mathcal{P}^+$ and the technique used to prove it depend heavily on the kind of high level formalism adopted and in general cannot be automated.

Notice that in our example, every aggregate transition submatrix $P_{I,J}$ with different row-sums, refers to a communication transition (let's call this transition t_{comm}). Since only one slave at a time can communicate, each quasi-lumpable communication transition aggregate contains two kinds of ordinary transitions: one for the slave $S3$ and the other one for the slaves $S1$ and $S2$. Thus a lower bound for $\mathcal{P}_{I,J}$ is given by:

$Pr^+(\text{comm. by the slave in } S2)\ \theta(t_{comm}, S2)\ +\ Pr^-(\text{comm. by a slave in } S1)\ \theta(t_{comm}, S1)$

with the constraint $Pr^+(\text{comm. by the slave in } S2)\ +\ Pr^-(\text{comm. by a slave in } S1)\ =\ 1$.

The probability p that place $inREC$ contains a token representing slave $S3$ given that $inREC$ is not empty can be formulated as

$$p = \frac{\frac{throughput(endRx,S2)}{\theta(endRx,S2)}}{\frac{throughput(endRx,S1)}{\theta(endRx,S1)} + \frac{throughput(endRx,S2)}{\theta(endRx,S2)}}$$

An upper bound for this probability can be found by considering the throughput of the slave in

$S2$ to be equal to the throughput of both slaves in $S1$ (let's call the common throughput value t):

$$p^{+} = \frac{\frac{t}{\theta(endRx,S2)}}{\frac{2\ t}{\theta(endRx,S1)} + \frac{t}{\theta(endRx,S2)}}$$

and after a simplification

$$p^{+} = \frac{\theta(endRx, S1)}{2\ \theta(endRx, S2)\ +\ \theta(endRx, S1)}$$

A similar argument applies to place $inSnd$.

Thus a lower bound for the rate of an aggregate transition representing a communication is obtained by weighting by p^{+} the row-sum corresponding to a communication transition involving the slave in $S2$, and weighting by $1 - p^{+}$ the remaining communication transitions associated with the slaves in $S1$.

The results obtained by applying this technique to the model of experiment 3 are the following:

Perf. Measure	Lower bound	Upper bound	Spread %
THRUPUT	0.012154	0.015649	29%
MNSWSnd	0.069087	0.161423	134%
MNSWRx	2.507557	2.773172	11%

The bounds are definitely tighter than those obtained in experiment three ignoring the model semantics. This indicates that whenever some additional information about the system behaviour can be derived from the high level model, it should be exploited to get improved bounds.

Chapter 4

Structural Decolorization

4.1 Colours, symmetries, and redundancies

In this section the notion of *redundant colour component* is defined; it is based on the observation that some symmetries in the RG of WNs are due to the presence of redundancy in the colour specification i.e., some colour components exist that do not need so many distinguishable elements for a correct specification of the model in terms of possible transition firing sequences. Some sufficient structural conditions for the reduction of the cardinality of a colour component at the net level (i.e., independently of the initial marking) are given. Notice that when the number of distinct elements of a basic colour set becomes one, the corresponding colour component can be cancelled from the net. The RG aggregation obtained through the above decolourization procedure in general is not detected (hence not exploited) by the SRG generation algorithm (and vice-versa). These two RG aggregation methods are then complementary and can be combined to better exploit the behavioural symmetries of the model.

The concept of colour redundancy and colour simplification has already been introduced in [32]. Here these results are extended. In fact now the partial decolourization of color classes is introduced besides the total decolourization. Moreover less restrictive sufficient conditions for colour simplification are defined.

The presence of redundancies in the model description is quite natural in CPNs, and derives from the methodology followed in the model construction. Different approaches can be followed such as, e.g., device ontology, process ontology, etc. Once the desired point of view is chosen, the basic elements that flow in the model structure are selected. Two extreme choices may be considered "natural" in the definition of the basic colours to identify such elements: either each basic element has its own identity or no distinction is made between basic elements (like in the case of P/T nets). This is because a-priori it would be too difficult to conceive a representation that is both minimal and correct. The best approach would probably be that of considering all basic elements undistinguishable during the model construction process until the necessity arises to actually make some distinction (for example to identify higher and lower priority processes). Anyway the results presented in this paper allow an easy construction of a WN model without consideration for the problem of minimality of the representation. Unnecessary distinctions among basic colours can be identified and eliminated automatically, thus producing an equivalent simplified model that is easier to analyze.

We start by illustrating the ideas that will be more formally developed in the sequel by means of a trivial example. The formalism used in the example is described in the following sections and here we just reason intuitively on it. The aim is to give some insights in the proposed method rather than show a good case of application of the technique.

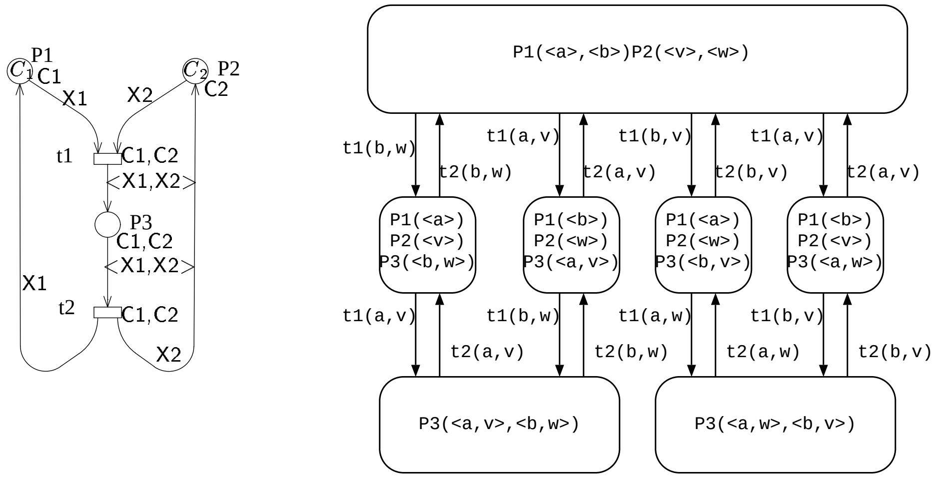


Figure 4.1: A simple WN system Σ_1 and its *RG*.

Let us consider the net in Figure 4.1; it has two basic colour domains C_1 and C_2. The net could be interpreted as a model of a production line where the elements of C_1 represent machines and the elements of C_2 represent raw pieces to be processed. We assume that any machine can process any piece and when a piece has been processed it is immediately replaced by a raw one.

In Figure 4.1 the complete RG for this model with $|C_1| = |C_2| = 2$ is also depicted. The reader is urged to imagine how huge the RG may become if we increase the cardinality of the basic colour classes. Observe however that, as far as the possible transition sequences are concerned, there is a high degree of redundancy in the RG. Indeed there are only two patterns of transition sequences from the initial marking and back to it, namely t_1, t_2 and t_1, t_1, t_2, t_2. The reason for the presence of redundancy is that we have introduced some unnecessary information in the model: the only thing we need to know is how many machines are working at any time, and this does not require the specification of the actual machine and the identity of the processed piece.

Figure 4.2.b (4.2.a) shows the reduction of the RG when two (one) colour classes are "decolourized", i.e. are dropped from the model. Even if trivial, this example can be used to understand how to proceed in the discovery and elimination of this kind of redundancies.

There are three concepts that we need to explain before presenting our "decolourization algorithm" in detail:

- *projection support*;
- *redundant colour component*;
- *structurally redundant colour component*;

The *projection support* describes the part of the colour structure that we are attempting to elim-

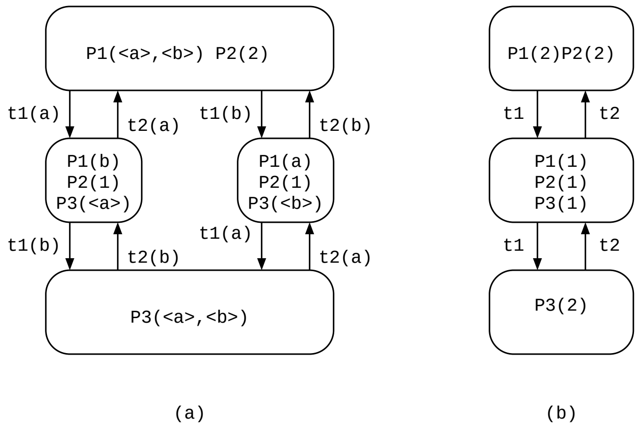


Figure 4.2: Decolourized *RG*s of Σ_1.

inate. We use the term *colour component* to indicate a part of colour structure identified by a given projection support. An example of "colour component" in the net of Figure 4.1 is:

- place P_1, colour domain component C_2
- transition t_1, colour domain component C_2
- place P_3, colour domain component C_2
- transition t_2, colour domain component C_2

A *colour component is redundant* if its elimination from the markings and firing instances of the underlying RG causes an aggregation of markings and arcs that satisfy the strong lumpability conditions. These conditions ensure that the possible firing sequences are preserved by the aggregation (hence by the decolourization).

In the RG of Figure 4.2.a we can observe that the lumping caused by dropping C_2 from both the markings and the firing instances is:

$$M_1, M_2 \rightarrow M_1'$$
$$M_3, M_4 \rightarrow M_2'$$

where

$$\begin{aligned}
M_1 &= P1(<a>), P2(<v>), P3(<b,w>) \\
M_2 &= P1(<a>), P2(<w>), P3(<b,v>) \\
M_3 &= P1(<b>), P2(<v>), P3(<a,w>) \\
M_4 &= P1(<b>), P2(<w>), P3(<a,v>) \\
M_1' &= P1(<a>), P2(1), P3(<b>) \\
M_2' &= P1(<b>), P2(1), P3(<a>)
\end{aligned}$$

Any transition fireable in each lumped marking was fireable also in the original markings and viceversa. Moreover the new marking reached by the transition firing in the original RG is contained [1] in the marking reached by the corresponding transition firing in the lumped RG.

Our final aim is to discover these redundant colour components by simply analysing the net structure, independently of the initial marking (and thus without computing the RG). In the net of Figure

[1] we say that a marking is the lumped RG *contains* a marking in the original RG if the former either has been obtained by lumping the latter with other markings or coincides with it.

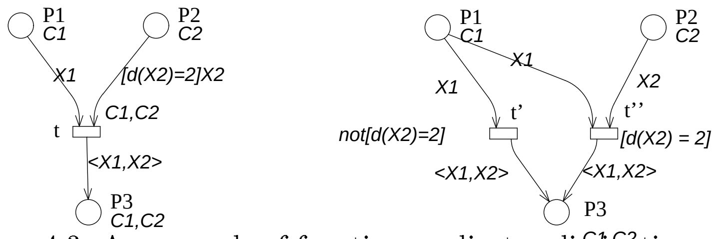


Figure 4.3: An example of function predicates elimination.

4.1 we can observe that the identity of the C_2 component of any transition colour instance is used neither to synchronize (wait for the same coloured tokens in different input places) nor to determine the truth value of a transition predicate. Hence the enabling of transitions is completely determined by the *presence or absence of tokens* rather than their colours. Therefore the C_2 component of the tokens flowing in the net is merely carried around the net but it never constitutes a crucial information to determine the enabling of a transition instance.

Intuitively, the algorithm to check if a colour component is structurally redundant should isolate the subnet of interest by "projecting" the whole net on the colour component, and controlling that no synchronization is present in it and no transition predicate requires the information we want to eliminate.

Finally we need precise rules to derive a correct uncolored net from the original one: the informal idea of "dropping" the colour component works with this example but in general it is too simplistic and must be refined as we shall see in the following sections.

4.2 A procedure for WN simplification

Let us first clarify the different effects that predicates can have depending on whether they label transitions or arc functions. Then we show how arc function predicates can be eliminated by a transformation of the net that in general increases the number of transitions.

Predicates on transitions are used to add some further condition on the transition enabling. Predicates in the function definitions are instead used to obtain variant functions, i.e., functions that depend on the particular colour instantiation of the corresponding transition. It may happen that an arc completely disappears for a particular instantiation of the corresponding transition (when all the predicates in the sum in Equation (1.1) evaluate to "false"). It is clear that in general this situation could prevent the application of any structure-based algorithm.

To overcome this problem it is possible to perform a transformation on the net structure that makes explicit all possible forms that a variant function can assume, and that moves the predicates from the arc functions to the corresponding transitions. Of course this transformation implies that each transition that has guarded functions on its arcs be replicated as many times as the possible different combinations of different definitions of variant function on the arcs. The example depicted in Figure 4.3 shows this kind of transformation. There is only one function predicate: $[d(X2) = 2]$. When the predicate is false, the arc disappears. When this predicate is true, the function on the arc connecting P_2 to t becomes simply $X2$. Hence two copies of t are needed that account for the two situations: t' with an associated predicate $[\neg\ d(X_2) = 2]$ and t'' with an associated predicate $[d(X2) = 2]$.

Hereafter we give an outline of the general transformation procedure to be performed for each transition t with at least one variant arc function connected to it.

- Let $\psi_t = \{pred_i\}$ be the set of all the predicates occurring on the arcs connected to t.
- Let $R \subseteq 2^{\psi_t}$ be the set of all possible non contradictory assignments of truth values to predicates $pred_i$.
- Add a replica t_r of transition t for each assignment $r \in R$ with the same input and output set of t and with the function f_i on each arc modified by substituting each predicate occurring in it with the corresponding truth value defined by r. If the result of this replacement is a constant null function, then the corresponding arc is removed. The predicate associated with t_r is equal to the predicate associated with t in "and" with the true predicates in r and with the negation of the false predicates in r.
- Delete transition t.

Notice that the above transformation is completely structural, i.e., it does not depend on the initial marking or the cardinality of the colour sets.

The structural colour redundancy detection and elimination algorithm that is described in one of the following subsections is based on the detection of possible colour synchronization patterns on the net structure. It may happen that WN models contain redundant structures that do not add information to the model itself, that have no influence on the state space size, and that introduce fictitious synchronization patterns in the model.

The following simplification procedures allow one to remove some of these redundancies. In the sequel of the chapter we assume that the WN models treated are simplified before any further consideration.

- Elimination of *implicit places*. The definition of implicit place for coloured nets is given in [33].
- Common coefficients simplification. The schemata of this structural simplification is shown in Figure 4.4. Before applying this rule, it is necessary to factorize all the coefficients within the function components as follows:

  $$\bigotimes_i \bigotimes_j (\lambda_{i,j} f_{i,j}) \quad = \quad \left(\prod_i \prod_j \lambda_{i,j}\right) \left(\bigotimes_i \bigotimes_j f_{i,j}\right)$$

  For example $6\langle X_1, X_2\rangle$ is the factorization of $\langle 6X_1, X_2\rangle, \langle X_1, 6X_2\rangle, \langle 3X_1, 2X_2\rangle, \langle 2X_1, 3X_2\rangle$, etc. Notice that this simplification is correct only if the initial marking of place p can be written in the form $M_0(p) = \{k_1 c\} : c \in C(p)$. In this case, it is always possible to translate the RG of the simplified net into that of the original net by multiplying all tuples in $M(p)$ by the factor k_1.
- Implicit colour components simplification. The schemata of this reduction rule is depicted in Figure 4.5 [2]. Also in this case the RG of the original net can always be retrieved from the RG of the reduced net as follows:
  - Case a: Cancellation of function $\sum S_{i,q}$. In this case the original marking $M(p)$ of place p can always be computed from the simplified marking $M'(p)$ as follows:

    $$\left(\bigcup D_{i,q}\right) \times M'(p)$$

[2]Functions f_l, f_l', g_l and g_l' in the lower figure are all assumed to have cardinality one.

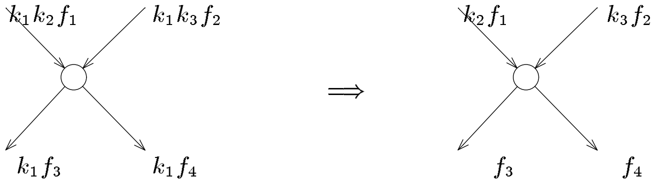


Figure 4.4: Schemata for common coefficient simplification.

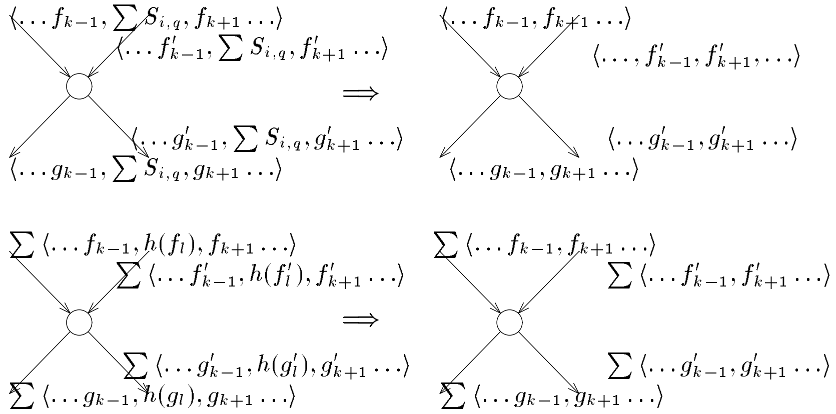


Figure 4.5: Schemata for implicit colour simplification.

Notice that this transformation is correct only if the initial marking of p can be written in the form $M_0(p) = \bigcup D_{i,q} \times M'_0(p)$

– Case b: Cancellation of a component f_k that can be computed as a function of another component $f_l = X_i$ or $\oplus X_i$, i.e., $f_k = h(f_l) = \alpha_j S_i + \beta_j X_i + \gamma_j \oplus X_i$. This can be done only if the above relation between components hold for each function tuple in the summation of each input and output arc of place p. It is required that component f_l be of cardinality one. Notice that this cancellation can be repeated on several different f_k for a given f_l. Also in this case there are some constraints on the initial marking of place p: it has to be partitionable into bags of tuples such that each tuple in the same bag has the same $l-th$ component c_l while the projection of the bag over the $k_t h$ component gives $h(c_l)$.

4.3 Colour redundancy elimination

In this section we consider only nets that do not have arc function predicates.

4.3.1 Decolourization of the Reachability Graph

The formal definition of RG decolourization is given over a set of colour components selected through a *projection support* for a given *colour shrinking function.* The concept of redundant colour component is also introduced.
Let $C_J = \bigotimes C_i^{e_i}$ be the Cartesian product of basic colour domains that define the colour of some place or transition.

Definition 4.3.1 (Projection support) *is a function from colours to components*

$$ps : \mathcal{C} \rightarrow 2^{(I\!N \times I\!N)}$$

where $\mathcal{C} = \{C_J : J \in Bag(I)\}$ *is the set of all possible colour domains that can be built on the set* C *of basic colour domains, the components are indicated by pairs* $\langle$colour domain index, occurrence index$\rangle$ *and* $ps(C_J) \subseteq \{\langle i,j\rangle : e_i > 0) \wedge (1 \le j \le e_i)\}$.

Definition 4.3.2 (Colour shrinking function) s *is a function from colour elements in the basic colour classes* C_i *to colour elements in the shrinked colour classes* C'_i*:*

$$s \; : \; \bigcup_{i\in I} C_i \rightarrow \bigcup_{i\in I} C'_i$$

such that $|C'_i| \le |C_i|$ *and* C'_i *has a number of static subclasses less than or equal to the one of* C_i *and*

$$\forall c_1, c_2 \in C_i, \quad d(c_1) = d(c_2) \quad \Rightarrow \quad d(s(c_1)) = d(s(c_2))$$

The function s *establishes a correspondence between a static subclass* $D_{i,q}$ *in* C_i *and a static subclass* $D'_{i,q}$ *in the shrinked colour class* C'_i *(we also use* $s(D_{i,q})$ *to denote the new static subclass corresponding to* $D_{i,q}$*). The shrinked static subclasses can be either of cardinality one or of the same cardinality of the originating static subclass* $D_{i,q}$*. If two or more static subclasses in the original model collapse into a single static subclass after the decolourization, then the new static subclass has cardinality one. The function must satisfy the property that each ordered subclass is either completely decolourized or not even partially decolourized.*

Observe that the definition of s does not require that this relation among static subclasses be injective. Moreover it allows that in a given basic colour domain some static subclasses are decolourized while others remain unchanged. However this does not apply to ordered classes, in fact it makes no sense to decolourize only some static subclasses of an ordered colour class because after the decolourization of a static subclass $D_{i,q}$ (of cardinality greater than one) we would be unable to compute correctly the function $\oplus X_i$ on the elements of the decolourized subclass.

Definition 4.3.3 *The decolourization through a colour shrinking function* s *of the components of a colour tuple* $c \in C_J$ *identified by a projection support* ps *is a function*

$$\mathcal{D}_{ps,s} : C_J \rightarrow C'_J$$

where

$$C'_J = \bigotimes_{i\in I} (C_i^{e_i - e'_i} \bigotimes C_i'^{e'_i})$$

and [3]

$$e'_i = \sum_{j=1}^{e_i} \Gamma_{ps(C_J)}(\langle i,j\rangle)$$

The decoulorization of a tuple $c \in C_J$ *with respect to a support* ps *and a shrinking function* s *gives another tuple of the same arity where each component* $c_{i,k}$ *selected in* ps *is substituted by* $s(c_{i,k})$*.*

Notice that when a colour class C_i that contains a single static subclass is shrinked so that $|C'_i| = 1$, the corresponding components in the tuple could be cancelled by the decolourization (in this case the arity of the tuple would decrease). The decolourization of a marking needs a projection support for each place and a unique colour shrinking function. We denote by ps_p the support corresponding to place p. Let $\mathcal{F}$ be the family of place projection supports:

$$\mathcal{F} = \{ps_p : p \in P\}$$

[3] $\Gamma_{ps(C_J)}$ is the characteristic function of set $ps(C_J)$

Definition 4.3.4 *The decolourization of a marking M through the colour shrinking function s on the components selected by the family $\mathcal{F}$ of place supports, denoted by $\mathcal{D}_{\mathcal{F},s}$, is a vector of $|P|$ functions; we denote by $\mathcal{D}_{ps_p,s}$ the $p-th$ component of $D_{\mathcal{F},s}$. $\mathcal{D}_{\mathcal{F},s}(M)$ is computed by applying the $p-th$ function to the corresponding place marking:*

$$\forall p \in P, forall c' \in C'(p) \quad \mathcal{D}_{ps_p,s}(M(p)(c')) \quad = \sum_{c:\mathcal{D}_{ps_p,s}(c)=c'} M(p)(c)$$

The subsequent step is the decolourization of the RG of a WN system $\Sigma = (\mathcal{N}, CS, M_0)$. It consists of the decolourization of its markings and of the transition instances labelling the arcs between markings. For this task the union of two projection support families is required: one for the places in the net and another one for the transitions.

Before defining the decolourization of RG(Σ) notice that such decolourization must be applied to place and transition components in a consistent way, taking into account the correlation between transition and place components induced by the functions.

Definition 4.3.5 *A family of projection supports is* consistent *with a WN iff the following conditions are satisfied*

1. *$\forall t \in T$, if $\langle i,j \rangle \in ps_t(C(t))$ and at least one predicate associated to the transition compares component $\langle i,j \rangle$ with another component $\langle i,j' \rangle$, then $\langle i,j' \rangle \in ps_t$*
2. *$\forall t \in T$, $\forall p \in {}^\bullet t \cup t^\bullet \cup {}^\circ t$, $\forall \langle i,j \rangle \in ps_t(C(t)) : X_i^j$ or $\oplus X_i^j$ occurs in component $\langle i,k \rangle$ of the function associated to the arc connecting p and t, $\langle i,k \rangle \in ps_p(C(p))$.*
3. *$\forall p \in P$, $\forall t \in {}^\bullet p \cup p^\bullet \cup {}^\circ p$, $\forall \langle i,k \rangle \in ps_p(C(p)) : X_i^j$ or $\oplus X_i^j$ occurs in component $\langle i,k \rangle$ of the function associated to the arc connecting p and t, $\langle i,j \rangle \in ps_t(C(t))$.*

Definition 4.3.6 (RG decolourization.) *Let Σ be a WN system and $RG(\Sigma)$ its reachability graph. let $\mathcal{F}'$ be a family of place and transition projection support functions*

$$\mathcal{F}' = \{ps_p : p \in P\} \cup \{ps_t : t \in T\} = \mathcal{F} \cup \{ps_t : t \in T\}$$

and s a colour shrinking function. The decolourization of $RG(\Sigma)$ onto $\mathcal{F}'$ through the shrinking function s denoted by $\mathcal{D}_{\mathcal{F}',s}(RG(\Sigma))$ is such that:

- *the markings of $\mathcal{D}_{\mathcal{F}',s}(RG(\Sigma))$ are obtained decolourizing the markings of $RG(\Sigma)$ with respect to $\mathcal{F}'$ and s, and then aggregating the identical decolourized markings;*
- *an arc $\langle \mathcal{D}_{\mathcal{F},s}(M_i), \mathcal{D}_{\mathcal{F},s}(M_j), t(\mathcal{D}_{ps_t,s}(c) \rangle$ exists in $\mathcal{D}_{\mathcal{F}',s}(RG(\Sigma))$ iff an arc $\langle M_i, M_j, t(c) \rangle$ exists in $RG(\Sigma)$; identical decolourized arcs are aggregated into a single arc.*

Definition 4.3.7 (Redundant colour component.) *A family $\mathcal{F}'$ of place and transition projection supports and a colour shrinking function s identify a* redundant colour component *for a given system Σ if the decolourization of $RG(\Sigma)$ onto $\mathcal{F}'$ through s satisfies the following* lumpability *conditions* [7]*:*

$$\begin{array}{l} \forall M_i, M_j \in RG(\Sigma) \;:\; \mathcal{D}_{\mathcal{F}',s}(M_i) = \mathcal{D}_{\mathcal{F}',s}(M_j) \\ \qquad \text{if } M_i[t,c\rangle M_k \text{ then} \\ \qquad\quad \exists c' \in C(t) \;:\; M_j[t,c'\rangle M_l \;\wedge\; \mathcal{D}_{ps_t,s}(c) = \mathcal{D}_{ps_t,s}(c') \;\wedge\; \mathcal{D}_{\mathcal{F}',s}(M_k) = \mathcal{D}_{\mathcal{F}',s}(M_l) \end{array}$$

The consequence of this property is that when a redundant colour component is eliminated from a system, the projected RG is reduced in size but still contains all the relevant information about the behaviour of the modelled system (in terms of possible transitions sequences).

We illustrate this concept by means of a simple example. In Figure 4.1 a WN system is shown together with its RG. Figure 4.2.a depicts the decolourized RG onto the family of projection supports and using the colour shrinking function s defined hereafter:

$ps_{P1} = \emptyset,\ ps_{P2} = \{\langle 2,1\rangle\},\ ps_{P3} = ps_{t1} = ps_{t2} = \{\langle 2,1\rangle\}$

$\forall c \in C_1 : s(c) = c;\ \ \forall c \in C_2 : s(c) = \epsilon,$ where ϵ denotes the neutral colour.

They make all elements of C_2 undistinguishable so that all C_2 components selected by the projection in places and transitions can be cancelled. Observe that the *strong lumpability* conditions of Definition 4.3.7 are satisfied, so that the whole colour domain C_2 (i.e., every occurrence of C_2 in the place and transition colours) is redundant in this system. The RG can be further aggregated by recognizing that C_1 is also a redundant basic colour domain.

4.3.2 Structural colour simplification

In the previous section we have defined the concept of redundant colour component based on the RG of a system. Since the aim of colour redundancy elimination is to reduce the size of the RG, our goal is to find some criteria that allow the detection of redundant colour components directly on the net structure, independently of the initial marking. The basic idea is to isolate the subnet which is candidate for decolourization by performing a *projection*. Then we check on this subnet whether some (sufficient) conditions are fulfilled. Finally we generate a new net from the original one by performing suitable transformations on the subnet under study.

In this section we define the projection of nets onto a family of projection supports and give some sufficient conditions on the net structure for the detection of redundant colour components. The family of projection supports used must satisfy the consistency constraints previously defined in order to obtain syntactically correct projected nets. In order to simplify the structural conditions that ensure colour redundancy, we propose to split the simplification procedure into subsequent "minimal simplification steps". This is achieved by applying iteratively the structural colour simplification using *minimal consistent projection support families.*

Definition 4.3.8 *A consistent projection support family is minimal iff no other consistent support familes is included in it. (We say that $\mathcal{F} \subset \mathcal{F}'$ if $\forall x \in P \cup T$, $ps_x \subseteq ps'_x$ and $\exists x \in P \cup T : ps_x \subset ps'_x$.)*

Observe that a minimal consistent support family will surely contain only components belonging to the same colour class. Anyway, given a colour class C_i, there may exist many minimal consistent projection support families containing disjoint sets of components of C_i. It is possible to find the set of minimal projection support families that cover the net and apply the procedures described in the sequel for each of them in some order. In general, different application orders of the decolourization procedure with respect to the set of minimal projection families lead to different final results and to different degrees of RG size reduction.

If more than one minimal projection support family refers to the same colour class C_i it may happen that C_i can be shrinked according to a given function s for one of these projection support family, but the same kind of reduction do not apply for the others. The solution to this apparent problem is in the definition of consistent projection support family; in fact since the components belonging to different consistent projection support families do not share objects (i.e. colour elements contained in the components selected by the first family cannot be "mixed" with those contained in the components selected by the second family) then we can use a different copy of C_i for each family and consider them as if they were different colour classes.

The procedure that checks whether a given projection support family $\mathcal{F}$ is structurally redundant works on the *projection over* $\mathcal{F}$ of the net $\mathcal{N}$. The definition of projection of $\mathcal{N}$ over $\mathcal{F}$ follows:

Definition 4.3.9 *The projection of a place (transition) colour domain* $C_J = \bigotimes_{i\in I} C_i^{e_i}$ *with respect to a projection support* ps *is:*

$$\mathcal{P}_{ps}(C_J) = C'_J$$

where $C'_J = \bigotimes_{i\in I} C_i^{e'_i}$ *and* $e'_i = \sum_{j=1}^{e_i} \Gamma_{ps(C_J)}(\langle i,j\rangle)$

Definition 4.3.10 *The projection a WN* $\mathcal{N}$ *onto a family* $\mathcal{F}$ *of place and transition projection supports, denoted by* $\mathcal{N}' = \mathcal{P}_{\mathcal{F}}(\mathcal{N})$*, is a net such that:*
$P' = P$; $\forall p \in P'$, $C'(p) = \mathcal{P}_{ps_p}(C(p))$
$T' = T$; $\forall t \in T'$, $C'(t) = \mathcal{P}_{ps_t}(C(t))$
$\forall t \in T$, *only the predicates that make reference to components selected by* ps_t *are kept in* $\mathcal{N}'$.
$\forall p \in P$, $\forall t \in T$, *if* $W^+(p,t)$ *(or* W^- *or* W^H*) is defined as*

$$\sum \delta \bigotimes_{i\in I} \bigotimes_{j=1}^{e_i} \left\{ \sum_{q=1}^{n_i} \alpha_{i,q}.S_{i,q} + \sum_{k=1}^{e_i} \left(\beta_{i,k}.X_i^k + \gamma_{i,k}. \oplus X_i^k \right) \right\} \tag{4.1}$$

the corresponding projected arc function is:

$$\mathcal{P}_{ps_p}(W^+(p,t)) = \sum \eta\delta \bigotimes_{i,j:\langle i,j\rangle \in ps_p(C(p))} \left\{ \sum_{q=1}^{n_i} \alpha_{i,q}.S_{i,q} + \sum_{k=1}^{e_i} \left(\beta_{i,k}.X_i^k + \gamma_{i,k}. \oplus X_i^k \right) \right\} \tag{4.2}$$

where the constant η *is computed on the excluded components of each element as the sum:*

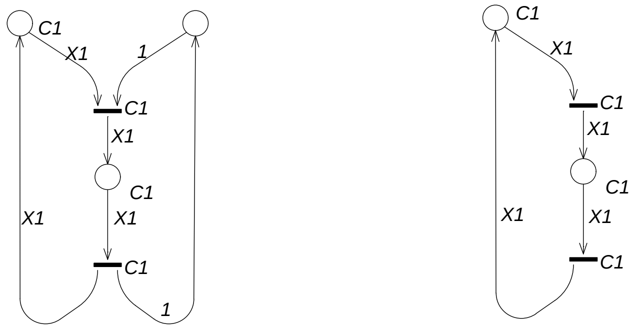
$$\eta = \prod_{i,j:\langle i,j\rangle \notin ps_p(C(p))} |W^+_{i,j}|$$

Examples of cardinality computed on functions:

- $|kS_i| = k|C_i|$
- $|S_{i,q}| = |D_{i,q}|$
- $|k(S_i - X_i)| = k(|C_i| - 1)$
- $|X_i + \oplus X_i| = 2$

Observe that the cardinality $|W^+_{i,j}|$ could not be computed in case of functions with predicates. For this reason we focused only on nets without function predicates.

Now let us describe a procedure that decides whether a given projection support family identifies a redundant colour. The procedure builds the appropriate shrinking function to be used in conjunction with the family of projection supports in order to obtain the decolourized net.



(a) (b)

Figure 4.6: Projection and cancellation of neutral subnet.

4.3.3 A sufficient condition for colour shrinking

The following procedure accepts a WN and a minimal consistent family $\mathcal{F}$ of projection supports as input and decides if the colour components selected by the projection supports are redundant. The procedure is made of two steps:

1. Projection of the net on $\mathcal{F}$ and cancellation of the neutral subnet; let $\mathcal{N}'$ be the net resulting from this step.

2. Redundancy check on $\mathcal{N}'$.

The first step should be clear. We illustrate it on the net and projection support of Figure 4.1. The projection gives the net in Figure 4.6.a. The resulting net after the cancellation of the neutral part is depicted in Figure 4.6.b.
The second step checks if the net $\mathcal{N}'$ resulting from the first step satisfies the properties listed below. The colour redundancy test is performed for each static subclass $D_{i,q}$ separately.

Definition 4.3.11 (Colour Redundancy) *Redundancy check for static subclass $D_{i,q}$. For each transition t in $\mathcal{N}'$ the following properties must be satisfied:*

1. *the set of functions that label the input and inhibitor arcs of t can only be functions of type X_i^j, $\oplus X_i^j$ and linear combinations of the two. If the class is not ordered and the place connected to the arc is colour-safe [4] this condition can be relaxed and diffusion functions (S_i or $S_{i,q}$) are allowed.*

2. *Let X_i^j be a function that can select one element in $D_{i,q}$ (i.e. the transition predicates do not imply that $d(X_i^j) \neq q$). No more than one occurrence of either X_i^j or $\oplus X_i^j$ is allowed in the labelling functions of all input and inhibitor arcs of t. Examples of forbidden situations are depicted in Figure 4.7.*

3. *If two or more functions X_i^j, X_i^k, X_i^l, etc. that can select elements in the same static subclass $D_{i,q}$ occur in the input or inhibitor arc functions of t, then the transition predicate ϕ_t must satisfy any one of the following constraints:*

   - *it is either a tautology (this comprises the particular case in which the predicate is omitted with the meaning that it is true);*
   - *or every predicate ϕ_t' such that $\phi_t \rightarrow \phi_t'$ and composed exclusively of standard predicates of equality type over X_i^j, X_i^k, X_i^l, etc., is always true.*

[4] A place is colour-safe iff its marking never contain two copies of the same tuple.

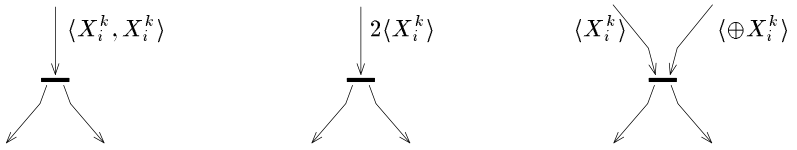


Figure 4.7: Some examples of function combinations that prevent decolouring.

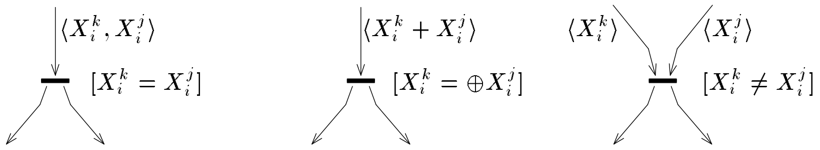


Figure 4.8: Some other examples of function combinations that prevent decolouring.

Examples of forbidden situations are depicted in Figure 4.8.

The third requirement can be relaxed in case of non ordered classes for the pattern in Figure 4.9 if place p is colour-safe. In fact in this case the predicate $[X_i^j \neq X_i^k]$ is always true and can be cancelled.

In the particular case of ordered classes, the above requirements should be verified for all static subclasses in which a colour domain is partitioned. Furthermore, predicates that refer to static subclasses should not occur on the selected transitions.[5]

If these properties are satisfied, then either the colour components under study are never used in a synchronization or into a conflict resolution (due to the conditions on the functions combination on input and inhibitor arcs), or the synchronizations can be solved just by counting the number of tokens in the place (due to the colour-safeness property of the place). Hence the static subclass $D_{i,q}$ can be decolourized by a function s that maps each element $c \in D_{i,q}$ onto the same element $c'_{i,q}$, the unique element of the shrinked subclass $D'_{i,q}$. Otherwise $D'_{i,q}$ is set equal to $D_{i,q}$ and the shrinking function s is the identity for the elements in $D_{i,q}$.

A further reduction in a colour class cardinality could be achieved by "merging" some decolourized static subclasses when possible. The presence of the predicate $[d(X_i^j) = d(X_i^k)]$ or its negation on any transition of the projected net prevents this operation. Provided that this kind of predicate does not occur, all static subclasses $D_{i,q}$ that satisfy the decolourization conditions and such that the predicate $[d(X_i^j) = q]$ does not occur on any transition in the projected net can be merged into a unique static subclass $D'_{i,q'}$ of cardinality one. Let Q be the set of mergeable static subclass indexes. The shrinking

[5]The reason for this further requirement is due to the rule of total decolourization of ordered classes that forces an ordered class to be either completely decolourized or remain unchanged.

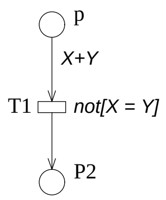


Figure 4.9: An exception to the third requirement for decolourization.

function s must be modified as follows:

$$\forall c \in \cup_{q \in Q} D_{i,q} : s(c) = c_{i,q'}$$

When the decolourization conditions are satisfied for some of the static subclasses of a class C_i the net $\mathcal{N}$ can be transformed using the following procedure. The decolourized net is denoted $\mathcal{D}_{\mathcal{F}',s}(\mathcal{N})$.

Definition 4.3.12 (Net decolourization) *with respect to a minimal consistent family $\mathcal{F}'$ of projection supports and a shrinking function s:*

- *The newly defined, shrinked colour class C_i' is added to the set of colour classes.*
- *For each place and transition colour components selected by the corresponding projection support the colour class C_i is substituted by the new class C_i'.*
- *Function components selected by the projection support of the corresponding place are transformed as follows:*

  X_i^j *remains unchanged*

  $S_{i,q}$ *is substituted by* $\frac{|D_{i,q}|}{|D'_{i,q}|} S_{i,q}$

  S_i *is substituted by* $\sum_q \frac{|D_{i,q}|}{|D'_{i,q}|} S_{i,q}$

- *Transition predicates that refer to completely decolourized classes are substituted by "true." Predicates referring to classes that are only partially decolourized remain unchanged, unless the predicate is of the form $X_i^j \neq X_i^k$ (that relate functions on input or inhibitor arcs —say X_i^j— to functions on output arcs, otherwise the decolourization would not be allowed). In that case it should be transformed taking into account the cardinality of the static subclasses in the original system. If all the static subclasses in the original net have cardinality greater than one then the predicate is changed to true otherwise it becomes*[6]

$$\left(\bigvee_{\forall q : |D_{i,q}| > 1 \ \wedge \ D_{i,q} \text{ has been shrinked}} [d(X_i^j) = q] \right) \ \vee \ [X_i^j \neq X_i^k] \tag{4.3}$$

Since the predicate in Equation (4.3) depends on the cardinality of the original static subclasses $D_{i,q}$, it is not completely parametric. The possible solution to this problem could be to consider for each static subclass that is involved in the predicate the two cases $|D_{i,q}| = 1$ and $|D_{i,q}| > 1$ and build the resulting reduced nets for all the possible cases. When the net is instantiated, it is possible to choose the correct reduced net among those generated by the reduction algorithm.

If the possibility of merging static subclasses is exploited a further transformation is needed on the reduced net predicates. In fact predicates of type $[d(X_i^j) = q]$ must be adjusted taking into account the new static subclasses indexes. Moreover, predicates of type $[X_i^j \neq X_i^k]$ must be transformed in the following boolean expression (where Q is the set of indexes of the merged subclasses)

$$[X_i^j \neq X_i^k] \ \vee \ \left(\bigvee_{\forall q \in Q} [d(X_i^j) = q] \right) \ \vee \ \left(\bigvee_{\forall q \notin Q : |D_{i,q}| > 1 \ \wedge \ D_{i,q} \text{ has been shrinked}} [d(X_i^j) = q] \right)$$

[6]The reason for using the above predicate is that different objects may belong to the same static subclass provided that it has a cardinality greater than one. When a static subclass of cardinality greater than one is shrinked, different colour elements in the original class could become equal in the decolourized class.

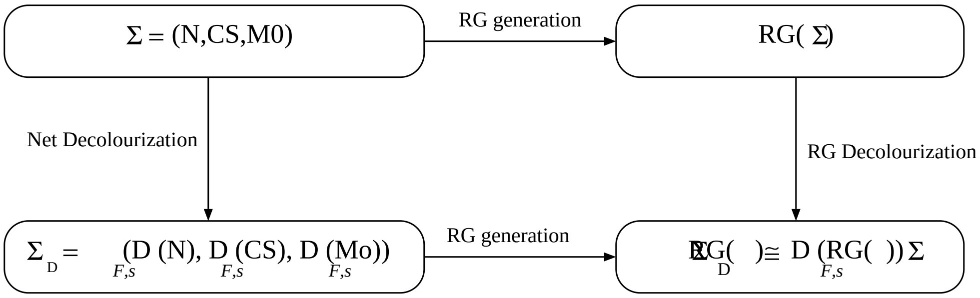


Figure 4.10: Illustration of Proposition 1.

.

instead of the one in Equation (4.3).

We have already mentioned that when a class is completely decolourized (i.e. $|C_i'| = 1$) the corresponding components can be "removed" from the net. When performing this cancellation however, the coefficients of the eliminated function component must be taken into account: the cardinality of each eliminated function component becomes a factor multiplying the function tuple. For example consider the following function tuple:

$$\langle f_{j_1}, \ldots, f_{j_{k-1}}, S_i - X_i, f_{j_{k+1}}, \ldots, f_{j_m} \rangle$$

after the complete decolourization of C_i the above tuple is transformed into:

$$\langle f_{j_1}, \ldots, f_{j_{k-1}}, |C_i|S_i - X_i, f_{j_{k+1}}, \ldots, f_{j_m} \rangle$$

(the "$|C_i|$" coefficient in front of function S_i derives from the decolourization rule for functions described previously). The component "$|C_i|S_i - X_i$" can be eliminated by multiplying the tuple by $|C_i| - 1$, that is the sum of the basic functions coefficients of the component. Indeed this gives the cardinality of the function since after the complete decolourization of the class both S and X return a single element. The resulting function tuple of reduced arity is:

$$(|C_i| - 1)\langle f_{j_1}, \ldots, f_{j_{k-1}}, f_{j_{k+1}}, \ldots, f_{j_m} \rangle$$

It is possible to prove the following statement:

Proposition 4.3.1 *Let $\mathcal{F}$ be a family of minimal projection supports, consistent with net $\mathcal{N}$, that satisfies the above properties. Let s be the corresponding colour shrinking function. Then the RG of every system $\Sigma = (\mathcal{N}, CS, M_0)$ satisfies the lumpability conditions and*

$$RG(\mathcal{D}_{\mathcal{F},s}(\mathcal{N}), \mathcal{D}_{\mathcal{F},s}(M_0)) = \mathcal{D}_{\mathcal{F},s}(RG(\mathcal{N}, CS, M_0)) \tag{4.4}$$

The above statement is schematized in Figure 4.10.

4.3.4 SRG aggregations versus decolourization

Let us discuss the differences between the aggregations obtained through the SRG generation and those obtained by the decolourization procedure proposed in this chapter.

The intuitive motivation for the aggregations achievable through decolourization and not through the SRG is the fact that the SRG aggregations are performed step by step, and preserve the minimal information necessary to solve every type of synchronization and/or conflict resolution that could be

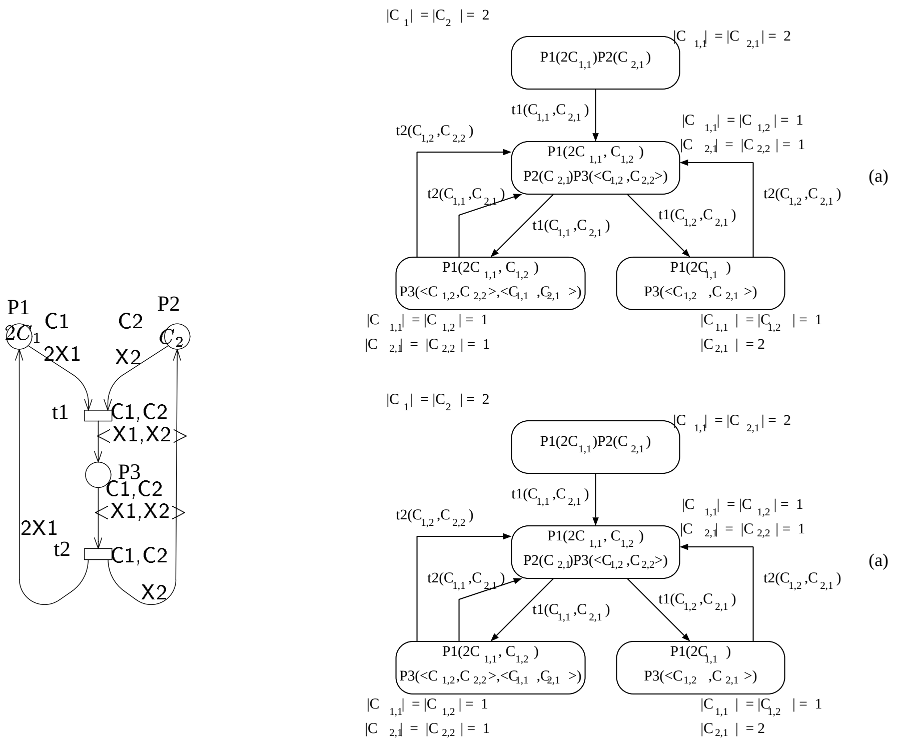


Figure 4.11: SRG aggregation versus decolouring aggregation: an example.

found in the future evolution of the net. The SRG construction algorithm has no knowledge of the global structure of the net, so that it cannot forecast whether certain pieces of information contained in the current state will ever be required to take decisions in future states.

On the contrary, the decolourization procedure considers whole subnets through which colour elements may flow and checks whether a colour specification is needed or not to take a decision about the enabling of any transition in the subnet. Therefore, in some sense, it looks ahead to the possible future needs of colour information. In Figure 4.11 a case is shown in which the decolourization aggregation is stronger than the corresponding SRG aggregation. In fact the SRG computed after the decolouring of C_2 (b) is smaller than that computed without decolourization (a). Notice that C_1 is redundant too, but remark that the SRG obtained after the decolourization of both C_2 and C_1 has the same number of states and arcs as the SRG in Figure 4.11.b after the decolourization of C_2 only.

4.4 An example

Let us now apply the decolourization method to a didactical example modelling a simplified skeleton of a message exchange communication protocol.

4.4.1 A message exchange system

The net in Figure 4.12.a models a skeleton of a packet-switching communication protocol in which the splitting of messages into data packets and the acknowledgement mechanism are represented. From the decolourization point of view this example is slightly more complicated than the previous one because of the presence of predicates and of static subclasses.

Two are the objects classes in this example: the sites and the packet types. The exchanged messages can be of two kinds: acknowledgement packets and data transfer packets. The message exchange protocol between any two sites is very simple: each site eventually transmits some data messages and waits for the corresponding acknowledgement. A site that has transmitted some data packets but has not yet received an acknowledgement cannot start another data message transmission to any site.

Tokens in place $Idle$ represent idle sites. Tokens in place $TxBuf$ represent messages contained in a local buffer; they can be either data packets that have to be transmitted to a site and acknowledged, or acknowledgement packets. The colour of tokens in this place is a 3-tuple: the first and second components are respectively the sender and receiver site identifiers; the last component is a packet type identifier. Transition $StartSnd$ represents the production of a set of data packets corresponding to the splitting of a message to be transmitted by a site. The predicate labelling this transition expresses the constraint that a site cannot send messages to itself. Transition Tx represent the transfer of a message from the transmitter buffer to the receiver buffer. Place $RxBuf$ contains data packets buffered by the receiver. Transition Rx represents the transfer of an acknowledgement packet to the sender site buffer. Finally transition $EndSnd$ represents the notification of the reception of the acknowledgement packet to the site.

There are two minimal consistent projection support families in this case: the first one contains the complete sites class; the second one contains the complete packets class.

Concerning the packets class, the two static subclasses have to be considered separately. The projected net satisfies the first decolourization requirement for neither of the two static subclasses: in fact there are diffusion functions S on the input arcs of transitions Rx and $EndSnd$. Nevertheless, if places $RxBuf$ and $TxBuf$ were colour-safe, the decolourization would still be possible.
This possibility depends on the initial marking as well as on the net structure. Assuming that the initial marking of place $Idle$ was the whole sites class and that the other places were initially empty, the colour-safeness condition would be satisfied (the statement could be proven by using the P-invariants, but this proof is out of the scope of this work). Under the additional condition that the net is colour-safe the two static classes can be decolourized but they cannot be merged into a unique class because of the predicate $[d(X_2) = 1]$ on transition Tx. The decolourized net is shown in Figure 4.13.a.

The net obtained by projection on the projection support family of the sites class does not satisfy the first decolourization condition unless $|D_{2,1}| = |D_{2,2}| = 1$. In fact the projected function on the arc from $RxBuf$ to Rx is $|D_{2,1}|\langle X_1^1, X_1^2\rangle$ and similarly the projected function on the arc from $TxBuf$ to $EndSnd$ is $|D_{2,2}|\langle X_1^2, X_1^1\rangle$. Of course if $|D_{2,1}| = |D_{2,2}| = 1$ then the packets class is already minimal; furthermore the sites class can be completely decolourized because the synchronizations on transitions Tx,Rx, and $EndSnd$ do not depend on the colour of the two elements X_1^1 and X_1^2. From a semantic point of view, the condition $|D_{2,2}| = 1$ is quite natural to enforce since there is only one type of acknowledgement packets. Instead the condition $|D_{2,1}| = 1$ means that no splitting of messages into smaller data packet fragments is performed and it is usually not verified. The decolourized net obtained in this case is depicted in Figure 4.13.b. Notice that in this case the predicate associated

$C = \{C_1, C_2\} \quad C_2 = D_{2,1} \cup D_{2,2}$

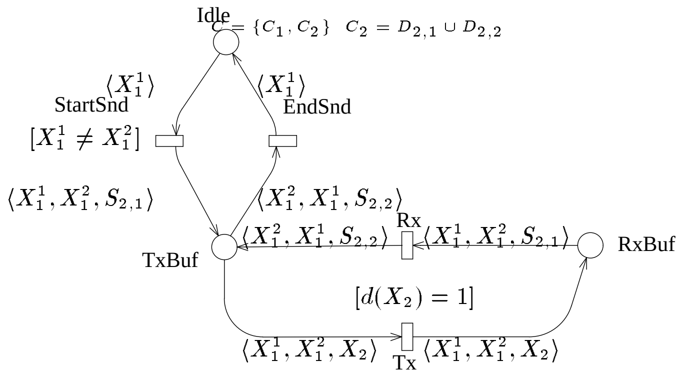


Figure 4.12: A message exchange protocol.

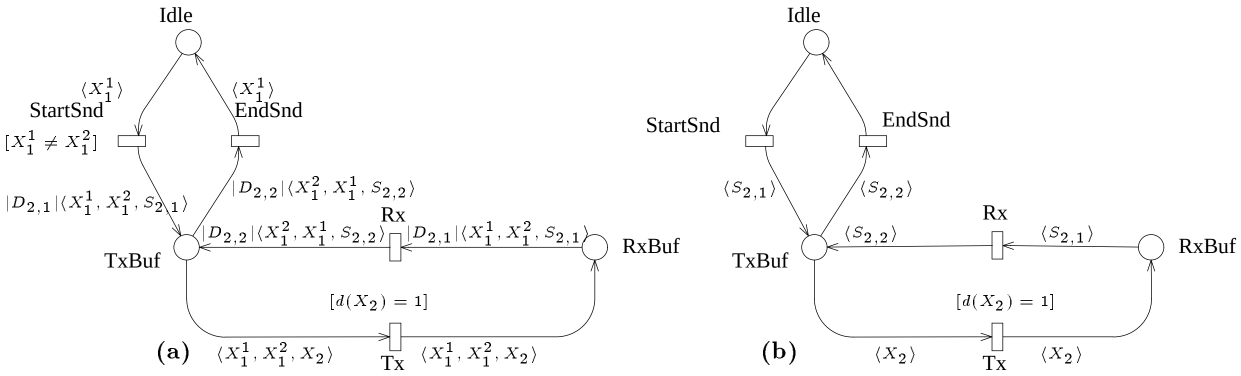


Figure 4.13: Decolourized models of the message exchange protocol.

with transition *StartSnd* has been removed because the sites class has been completely decolourized.

4.5 Observations

The possibility of defining redundant WNs is important from a modelling point of view since it simplifies the task of setting up a model. However, the presence of redundancies in the marking description may hamper the practical possibility of model analysis. Hence the development of algorithmic model simplification techniques is crucial for a practical utilization of a CPN formalism.

In this section the concept of colour structure simplification has been introduced for WNs. The simplification technique implies a reduction of the size of the reachability graph while preserving the behavioural properties of the net. This work significantly extends the initial results on colour simplification presented in [32]. The reduction is obtained by exploiting some colour symmetries of the net.

Unlike the symbolic reachability graph technique, the symmetries are detected and eliminated at the net structural level, so that the simplification obtained is parametric on (independent of) the cardinality of the basic colour sets. The decolourization and the SRG techniques are independent and complementary, so that they may be combined to optimize the analysis of WNs.

Chapter 5

Tool Support

5.1 History of *GreatSPN*

GreatSPN 1.7 is a tool for the modeling and analysis of systems, based on the Petri nets formalism. We first describe shortly the historical evolution of the package, which explains the reasons for some implementation choices as well as the intended purpose of the package. In Section 5.2 the functionalities of the current version of the tool are listed. In Section a few examples are presented to show the coordinated use of the different tool functionalities in a modeling/evaluation effort. In Section 5.4, several analysis modules are discussed in terms of algorithms used and of implementation issues that may significantly affect their efficiency.

The first impulse to the development of the *GreatSPN* package stemmed from the research pursued by the Torino group on generalized stochastic Petri nets (GSPN). GSPNs were initially developed as a tool for the specification and performance evaluation of computer architectures at the Dipartimento di Elettronica of the Politecnico di Torino and at the Dipartimento di Informatica of the Università di Torino [10], in the frame of the Progetto Finalizzato Informatica of the Italian Consiglio Nazionale delle Ricerche, MUMICRO project. The development of GSPNs was stimulated by the results on SPNs described in the Ph.D. thesis of M. K. Molloy. In GSPNs a new class of transitions (called *immediate*) that fire in zero time with priority over timed transitions was introduced. A solution algorithm that exploits the reduction of the size of the Reachability Set (RS) due to the presence of immediate transitions was first described in [10].

Several computer programs were developed as part of graduation thesis to implement the steady-state numerical solution of GSPN models, eventually leading to a first documented software package for their analysis [34]. This package allowed to experiment with the new modeling tool and gain insight into the memory and CPU time requirements of the solution algorithms as functions of the size of GSPN models. The weak points of this package were scarce portability and flexibility of the programs, and the lack of a graphic interface, which is the most natural type of support for the definition of GSPN models. Subsequent efforts were devoted to designing a package with the following characteristics: (1) user friendliness – in particular the availability of a graphic interface for the model definition was considered a must to satisfy this requirement –, (2) portability, (3) modularity and easy upgradability, and (4) efficiency of the analysis modules.

The first step in this direction was the implementation of the software tool described in [35]. A decomposition was pursued both of the software tool and of the analysis steps. Several intermediate results were identified and stored in form of separate files. Several independent programs cooperate to the production of final result files by taking as input intermediate result files produced by running other modules of the tool. Thanks to this modular software architecture [26], the tool was easily upgraded and adapted to different uses as soon as new theoretical results provided new analysis algorithms. From the functional point of view, earlier versions of *GreatSPN* included a graphical interface (based on the SunView package and on the PixRect utilities for basic graphics), all the algorithms for the generation and steady-state or transient solution of the "underlying Markov chain" of a GSPN, and a new algorithm for the analysis of a class of models containing a mix of exponentially distributed and deterministic timed transitions (DSPN) [36]. A Monte Carlo simulation program with confidence level estimation was also introduced for two main reasons: (1) to provide a tool for performance evaluation in the general case of Timed Transitions Petri Nets (TTPN) that are not analytically solvable and (2) to provide a tool for the validation of models when numerical solutions cannot be implemented due to the size of the Reachability Graph (RG), which is equal to the number of states of the underlying Markov chain[1].

GreatSPN started to become an interesting and useful support for the performance modeling, and many research and education institutions asked for permission to have it, so that the Dipartimento di Informatica of the Università di Torino began its free distribution. A major check-point was undertaken with the release 1.3 [37], which included all the structural analysis techniques for the validation of the underlying Petri net structure of a GSPN model. Major improvements in the validation capabilities of the package were achieved with the implementation of algorithms for the computation of Place- and Transition- invariants, and of the specific structural analysis techniques for the qualitative validation of GSPNs proposed in [23].

At this point the weakest part of the package was the simulation feature, which used a very straightforward (and inefficient) Monte Carlo non event-driven technique for the generation of the sample paths. The same structural properties computed for the model validation were then reconsidered from a different perspective: they were used for the optimization of the data structures of the analysis and simulation programs [38]. This idea led to a completely new set of solution and simulation programs, and eventually to the implementation of the interactive simulation facilities [39] of *GreatSPN* 1.4.

One of the drawbacks of versions 1.3 and 1.4 was that they comprised both modules written in Pascal and modules written in C. In version *GreatSPN* 1.5 all the Pascal modules were rewritten in C in order to increase portability. Moreover, new algorithms and techniques were implemented for the efficient and direct construction of the Tangible Reachability Graph (TRG) [40, 41], further reducing the space and time requirements of this phase with respect to the technique proposed in [38]. In version 1.6, the graphic interface (see Figure 5.1) was rewritten based on XView, a public domain toolkit (included in the MIT distribution tape of X11R5). The basic structure and features of the graphic interface of *GreatSPN* 1.3 were retained. Minor changes were introduced in order to present the control items in a more rational way. Some features have been added in a straightforward way in order to allow the visualization of the new structural, behavioural, and performance results obtained by the new analysis modules.

GreatSPN 1.7 represents a new major check-point for the package. New algorithms have been added for the fast computation of performance bounds based on linear programming techniques [42], working at a purely structural level. The computed bounds depend only on the average firing delay of the transitions while they do not depend on the p.d.f. of such delays. Algorithms have also been added for the analysis of high-level Petri net models providing the user with the possibility of designing

[1] In any case simulation usually requires costly and long computations.

models of complex systems in a more compact way. The chosen high-level formalism is Stochastic Well-Formed nets (SWNs), for which efficient algorithms have been defined, that automatically generate the Symbolic RG exploiting the model symmetries.

Research is still going on, and will eventually produce new implementations to be introduced in *GreatSPN* on hierarchical modeling [43] and on the exploitation of parallel processing techniques for the efficient analysis of large models [44, 45]. Concerning the graphic interface, porting under OSF/Motif is being considered.

5.2 Current Tool Functionalities

GreatSPN is composed of many separate programs that cooperate to the construction and analysis of PN models by sharing files. Using network file system capabilities, different analysis modules can be run on different machines in a distributed computing environment. All modules are written in the C programming language to guarantee a good portability and efficiency on different Unix machines. All solution modules use special compact-storing techniques to save memory both for intermediate result files and for program data structures. The more restrictive constraints are on the Reachability Graph construction phases; the current practical limit is a few hundred thousands markings on SUN SPARC machines with 12–16 MBytes of main memory. The main functionalities of the tool can be outlined as follows:

graphic interface running under OpenWindows 3.0 on SPARC workstations with SunOS 4.1.3 (Solaris 1.0) allows:

- graphical model editing including layering, cut and paste, selection, rotation, rescaling, spline, zooming, printing in PostScript, etc.
- graphical representation of structural properties
- interactive token game for PNs with priorities and inhibitor arcs
- access to the model data base, with possibility of cut and paste of submodels
- definition of timing and stochastic specifications, parameters, and performance measures
- menu-driven interaction with analysis modules
- graphical representation of performance results
- graphical interactive simulation of timed and stochastic models

structural properties for nets with priorities and inhibitor arcs

- Place- and Transition- invariants
- deadlocks and traps
- implicit places
- causal connection and structural conflict
- mutual exclusion
- Extended Conflict Set (ECS) and structural confusion
- structural boundedness and unbounded transition sequences

linear programming performance bounds for timed P/T nets (upper and lower bounds for transition throughputs and average place markings)

reachability graph generation and analysis programs

- home space, deadlocks, and livelocks

- transition liveness degree

Markovian solvers for steady-state as well as transient performance evaluation exploiting efficient, sparse matrix numerical techniques.

simulation modules for interactive event-driven simulation. In cooperation with the graphic interface, they provide

- interactive simulation, with graphical animation of the model and movement of tokens, with step-by-step, and automatic run, forward and backward time progression, real time update of performance figures, arbitrary rescheduling of events.
- batch simulation, with periodic sampling and real time update of performance figures on the graphic interface and computation of confidence intervals

The simulation state is periodically saved on a file, so that previous simulation runs can be resumed at any time. Several probability distributions are allowed for transition timings (including uniform, discrete, Cox, Erlang, etc.).

well-formed colored nets analysis prototypes

- construction of symbolic reachability graph, conversion into lumped Markov chain, steady state and transient Markovian analysis
- colored and symbolic simulation (currently integrated only in input, i.e. producing results in output files that are not displayed on the graphic interface)

Let u

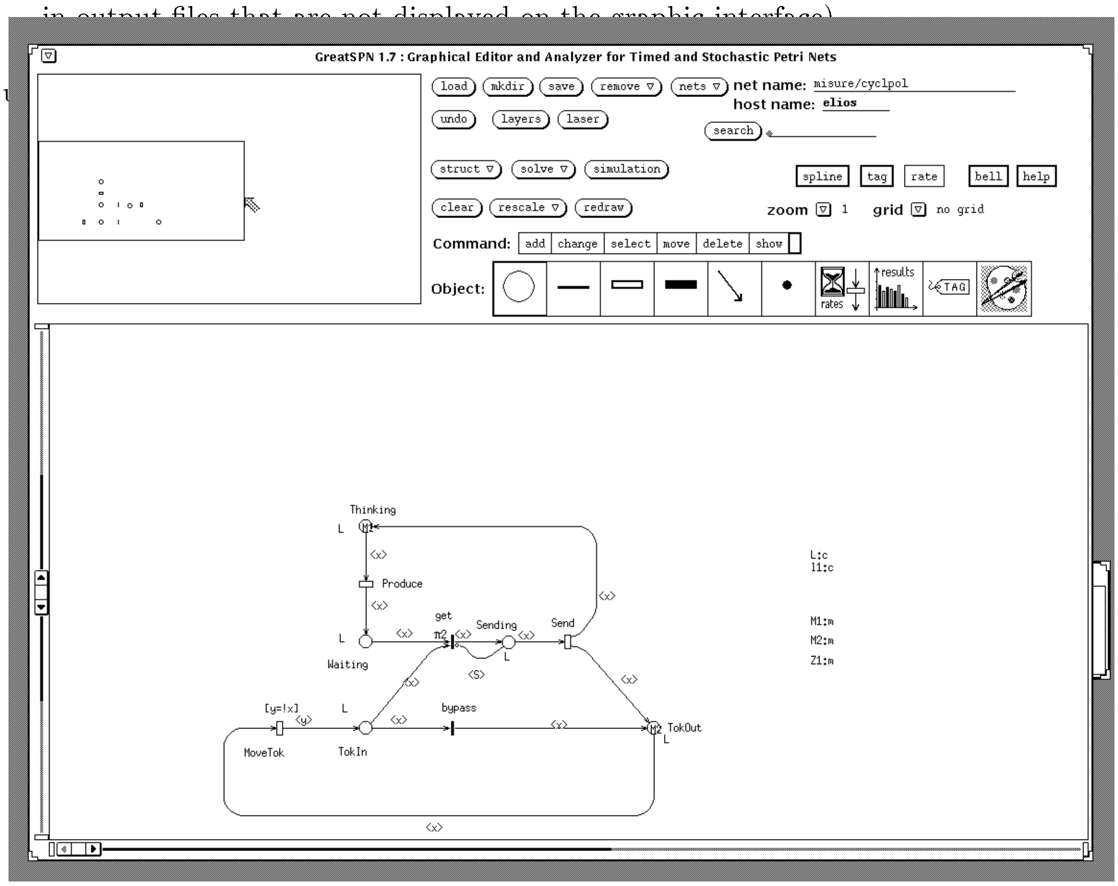


Figure 5.1: Appearance of the GreatSPN1.7 graphic interface.

The window in the upper left corner shows an overview of the whole net currently edited. The big window below is the working space of the editor; only a portion of the drawing space is visible in this window: it corresponds to the box visualized in the overview window. The user can move the visible part of the drawing space by either moving the box in the overview window or using the scrolling bars associated with the drawing window. The window in the upper right corner is a panel governing the editor. It contains *buttons* (e.g., the ovals labeled “load”, “mkdir”, and “laser”) used to invoke commands, *toggles* representing *on/off* switches (e.g., “spline” and “bell”) and *cycles*, containing lists of values that “roll over” (e.g., “zoom” and “grid”). In the interface there are two menus of choices, the *Command* and the *Object* menus used to select different possible options. For instance the “*add place*” option of the graphical editor can be selected by clicking on the “add” choice and then on the place icon choice. There are also three *text items* used to visualize the name of the currently edited net, the name of the machine onto which the analysis programs should run and the text to be searched for in the net (it is possible to ask the editor to search for a place or transition with a given name).

5.3 Well-formed Colored Net Example

In this section the implementation of the SWN formalism is described through an example, then we shall describe the facilities provided by the *GreatSPN* package for the analysis of SWN models.

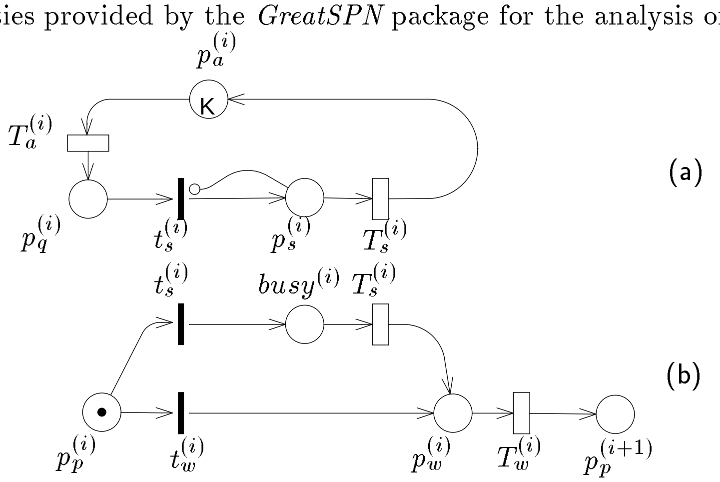


Figure 5.2: GSPN representations of a queue and a server.

Using the SWN formalism it is possible to build more compact and parametric representations of symmetric systems by “folding” similar GSPN subnets such as the one depicted in Figure 5.2. Figure 5.3 shows the SWN model of a polling system example that has been obtained by folding all the replicas of the submodels corresponding to different stations. When several similar subnets are folded, some additional annotation is needed to be able to distinguish tokens that end up to be in the same folded place. These annotations constitute the color structure of the net. The BNF grammar of the syntax used for the definition of the color structure of SWNs models is shown in Figure 5.4. In the SWN model of Figure 5.3 tokens carry information to distinguish the customers associated with different queues. The type of data associated with tokens in a given place — the *place color domain* — can be structured, i.e. it can comprise several “fields” associated with basic data types. The basic data types are finite and non empty sets called *basic color classes*. In our example we need only one basic color class defined as $L = \{q_1, q_2, \ldots, q_M\}$ that represents the M queues. The ring connection induces a circular order relation among queues, characterized by the “next station” relation. L is thus defined to be an *ordered* color class with the possibility of applying the *successor* function (denoted “!”) to any of its element (i.e., $!q_i = q_{(i+1) mod M}$).

The initial marking of place p_a is $K\langle S\rangle$ where K represents the maximum capacity of each queue

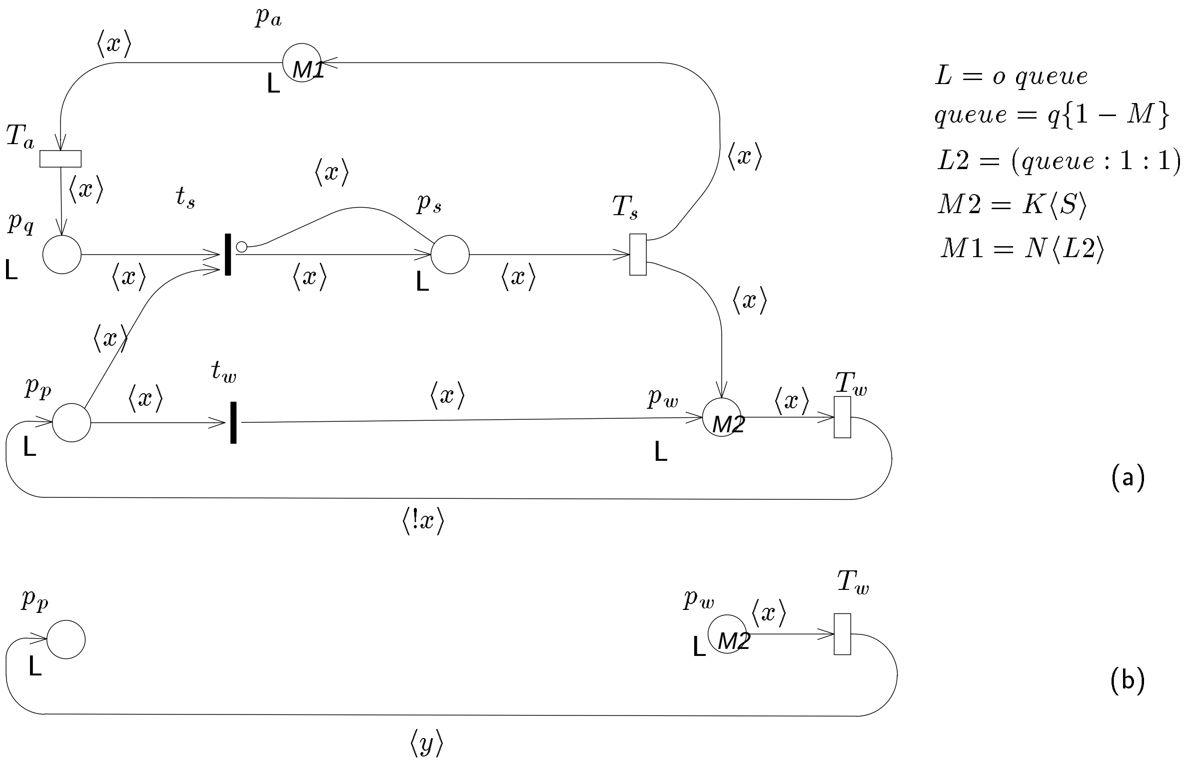


Figure 5.3: SWN representations of the cyclic and random polling systems.

and $\langle S \rangle$ is a special symbol denoting the diffusion function, i.e. *all* the colored tokens belonging to the color domain; the initial marking of place p_w represents the initial position of the N servers and is equal to $N\langle L2 \rangle$ ($L2$ is a subset of class L of cardinality one containing q_1; we are assuming that all servers are initially polling the first queue of the ring).

A transition in a SWN model actually represents several transitions in the corresponding "unfolded" GSPN model: for example the firing of transition T_a represents an arrival to any queue. In SWNs transitions are parameterized: the parameter types are chosen among the set of basic color classes. Transition T_a has a parameter x of type L. A *transition instance* is obtained assigning actual objects of proper type to the transition parameters; enabling and firing is defined only for transition instances. The set of "colored" tokens that are withdrawn from (added to) the input (output) places of a given transition when one of its enabled instances fires, are defined through *functions* (see Figure 5.4 for functions syntax) labeling the net arcs. The enabling of a transition instance depends both on the multiset of colored tokens contained in its input and inhibitor places and on the evaluation of an optional *predicate* associated with the transition.

The transition instance of T_a that assigns q_2 to parameter x is enabled in the initial marking. Its firing deposits the colored token $\langle q_2 \rangle$ in place p_q and removes it from place p_a thus modeling an arrival to the second queue. This is due to the fact that function $\langle x \rangle$ labeling both the input and output arcs of T_a is an identity function. When one of the N servers polls the second queue (i.e., when place p_p contains the colored token $\langle q_2 \rangle$) the service can be provided and the immediate transition instance of t_s binding the parameter x to q_2 can fire. The inhibitor arc connecting t_s and p_s prevents the enabling of transition t_s for any color q_j present in place p_s. A server that polls a queue in which another server is working will bypass that queue (i.e., transition t_w will fire) because in the modeled system only one customer can be served in each queue. A server moving to the next queue is modeled by means of the combined presence of the projection function $\langle x \rangle$ and successor function $\langle !x \rangle$ labeling the input and the output arcs of transition T_w.

The SWN model in Figure 5.3 is parametric in the queue capacities (K), in the number of servers (N) and also in the number of queues in the system ($|L|$). An important feature of this colored model is that the service policy may be easily modified; for example in order to model a random service policy instead of the cyclic one, it is sufficient to replace the successor function labeling the output arc of transition T_w with a projection function $\langle y \rangle$. Observe that such a variation in a GSPN model would require much more complex structure manipulation.

Once the SWN model of the polling system is drawn we can perform its analysis. In the pop-up menu activated by clicking on button "solve" we can find the option "SWN solution" that comprises the following modules:

1. "Symbolic Reachability Graph (SRG) computation"
2. "Symbolic simulation"
3. "Colored simulation"

The SRG computation module generates the Symbolic Reachability Graph and the lumped Markov chain underlying the model. The SRG of the net shown in Figure 5.3 when $|L| = 4$ and $K = 1$ has 30 symbolic tangible markings (representing 312 ordinary tangible markings of the colored model) and 30 symbolic vanishing markings (corresponding to 384 ordinary vanishing markings). No deadlocks are present in the model and the initial marking is a home state (hence the underlying Markov chain is ergodic). A portion of the file describing the SRG is depicted in Figure 5.5. For each symbolic marking the file contains its description and the number of ordinary markings that the symbolic marking represents. For each enabled transition instance the file contains the firing rate and the new symbolic marking reached after the transition firing. The symbolic marking description consists of two parts: the list of the symbolic markings for non empty places and the list of *dynamic subclasses* with their cardinalities. The symbolic representation of the initial marking is

$$p_a(1\langle L1\rangle 1\langle L2\rangle)p_w(2\langle L2\rangle) \quad |L1| = 3, |L2| = 1$$

$L1$ and $L2$ denote two *dynamic subclasses*[2] of L of cardinality three and one, respectively. The *symbolic token* $1\langle L2\rangle$ in place p_a represents a single queue while the *symbolic token* $1\langle L1\rangle$ represents three ordinary tokens corresponding to the other three queues. The reason for having two dynamic subclasses in the initial marking is due to the fact that we need to distinguish between the queue where the servers start their polling cycle (represented by the dynamic subclass $\langle L2\rangle$) and the other queues. Notice that we are not interested in the identity of the objects within a given dynamic subclass but only in the number of objects each dynamic subclass contains. Each symbolic marking in fact represents a class of ordinary markings that can be obtained assigning objects identities to the elements in the dynamic subclasses. For instance, the initial marking represents four ordinary markings because we have four different ways of associating a queue to the dynamic subclass $L2$ and the remaining three queues to the dynamic subclass $L1$.

Two *symbolic instances* of transition T_w are enabled in the initial marking. Let us consider the symbolic instance $T_w(\langle L2, L1\rangle)$; variables x and y are bound to subclasses $L2$ and $L1$ respectively. The firing of this symbolic instance of T_w leads to symbolic marking 2 as shown in Figure 5.5: it represents the set of ordinary firings that can be obtained by assigning any possible subset of L to

[2] These subclasses are called *dynamic* because they change from one symbolic marking to another.

dynamic subclasses $L1$ and $L2$, and then binding parameter x to any element of $L2$ and parameter y to any element of $L1$.

These concepts of symbolic marking and symbolic firing should give an idea of the fact that the SRG has a smaller number of states with respect to the ordinary RG. In Figure 5.6, the SRG size versus the RG size of both the cyclic and the random polling SWN models are shown. As stated before, the SRG computation module computes also the lumped Markov chain underlying the SWN model plus some intermediate files that are necessary for the computation of the probability distribution of the number of tokens (independently of their colors) in places. These latter measures can be visualized on the graphic interface selecting the "show results" option on the panel.

If the cardinality of class L is so large that the SRG generation module requires enormous amounts of time and space then we can analyze the system using a simulation module. The result of the simulation is written into a file in ASCII form and it cannot be visualized on the graphic interface for the moment[3].

5.4 Implementation issues

The modular structure of *GreatSPN* 1.7 is shown in Figure 5.7. Rectangles represent program modules while ovals represent the intermediate files generated by the modules. Dotted arcs are used to highlight how easily the addition of colored analysis algorithms could be accommodated thanks to the package modularization. The MC solution part is common to GSPN and SWN solutions. In this section the most relevant implementation aspects of the various modules are briefly discussed.

5.4.1 Symbolic RG for Well-formed colored nets

The module that implements the state space generation for SWN models exploits the symmetries that may be present in the modeled system by applying the Symbolic Reachability Graph (SRG) generation algorithm. The technique leads to remarkable reductions in the state space size while allowing the study of any property that could be checked on the whole RG. The same module produces also the CTMC isomorphic to the SRG: it has been shown that this Markov chain can be obtained by lumping the CTMC isomorphic to the whole RG. The steady state (or transient) probability distribution of the symbolic markings is computed by the same modules used for the analysis of CTMCs generated from GSPN models.

The reduction obtained in the number of states, usually does not lead to a reduction of the same magnitude in the computation time because the symbolic firing operation is inherently more complex than the "ordinary" firing operation. However, as the color class sizes increase, the symbolic firing overhead is largely compensated by the orders of magnitude difference between the RG size and the SRG size.

Several techniques can be used to improve the efficiency of the SRG construction implementation. Some of them could be inherited from the GSPN TRG construction implementation. Currently the lack of tools for the structural analysis of SWN models prevents the full application of the optimization techniques applied in the TRG generation module. However some structural property that can be computed on the net structure disregarding colors are exploited in the current version.

Considerable increase in the efficiency can be obtained by carefully implementing the procedure that generates the list of transition instances enabled in a given symbolic marking. The straightforward

[3] This feature will be included in following versions of the tool.

implementation consists of generating all possible instances for each transition and then performing the enabling test for each one. A much more efficient approach consists of directly generating the set of enabled instances starting from the current marking of the input places. This technique is successfully applied when the transition input arcs are labeled with simple functions, which usually is the case for most transitions in SWN models. The optimized procedure implements the following rules: (1) arcs connecting places with neutral domain and arcs labeled by diffusion functions pose conditions independent of the transition color instance and therefore their test must be performed first, (2) a tuple of "easy" functions (i.e. projection or successor functions) can only match tokens that are present in the marking of the connected place. If all input places have arcs labeled with such functions, we obtain the set of enabled color instances by intersecting the marking of these places; (3) a tuple of "mixed" functions (i.e. diffusion functions and projection/successor functions) can only restrict the set of possible instantiation for the transition instance elements; this allows to perform the actual enabling test on fewer color instances. These functions are the most complex to evaluate, so that their test must be delayed after the test of "easier" functions has already reduced the set of possible instantiations that should be checked.

5.4.2 Embedded Markov chain construction and analysis

These modules are used for the performance evaluation of a SWN with finite (and sufficiently small) SRG. The relevant steps for the construction and analysis of the EMC are: (1) the parsing of marking dependent transition firing rate definitions and of performance measure definitions using LEX and YACC; (2) the computation of the EMC and initialization of the marking probability vector; (3) the computation of either the steady-state solution of the EMC using an iterative numerical method based on Gauss-Seidel algorithm or the transient solution at a given time of the EMC using an optimized, numerically stable version of the randomization method; (4) the computation of the performance measures based on the marking probability vector representing either the steady-state or the transient solution of the EMC.

The EMC underlying a SWN model is constructed starting from SRG of the SWN, and labeling the arcs with the corresponding transition firing probabilities; the EMC matrix is stored in a sparse and compact form. An entry in the EMC may correspond to the firing of one timed transition followed by an arbitrary number of firings of immediate transitions.

The numerical computation of the steady-state marking probability distribution is implemented by a variation of the Gauss-Seidel iterative solution of the EMC. Partial results are periodically check-pointed, and the iteration procedure can start from the last check-point stored on file (in this way the solution can be restarted after an interruption of the program, without losing the previous results). At the end of the iteration procedure, the EMC probability distribution is renormalized according to the mean sojourn times in each marking in order to obtain the steady-state probability distribution of the continuous time Markov process.

The numerical computation of the transient marking probability distribution at a given instant of time with respect to the starting time is computed by a variation of the randomization method [46]. Even extremely stiff matrices can be integrated with a time complexity which is sublinear with respect to variations in the integration time. Also in this case partial results can be check-pointed, and the iteration procedure can start from the last check-point. The marking probability distribution at time zero is the initial marking with probability 1.

Starting from the steady-state or transient marking probability distribution, a number of default as well as user defined performance measures are automatically computed in *GreatSPN* 1.7. The default performance indexes are the probability distribution of the number of tokens in each place, and the throughput of each transition. In addition, other performance results may be defined as "result" objects of a GSPN model in terms of the probability distribution of the markings and the timing parameters of the model. A formal grammar is used to compose result definitions. A module which contains the parser generated according to the YACC specification of the definition grammar, produces the optimized data structures for the computation of the performance measures.

5.4.3 Efficient simulation of colored nets

Two SWN simulation modules have been implemented: colored simulation and symbolic simulation. The former module implements an event driven simulator of SWN models, that uses the "ordinary" colored marking for the state representation and "ordinary" colored transition instances for the events denotation.

The latter module implements an event driven simulator of SWN models that uses the symbolic marking for the state representation and the symbolic transition instances to denote *sets of scheduled events* in the event list. This approach allows a reduction of the average and maximum length of the event list [47] since there is only one event scheduled per symbolic firing instead of as many events as the different (but equivalent) firing colors represented by the symbolic firing. The average length of the event list depends on the number of dynamic subclasses of the most probable markings, hence it is influenced by the transition rates. The maximum reduction is achieved when the most probable markings have a minimum number of dynamic subclasses.

In the polling system example, the maximum number of dynamic subclasses has an upper bound due to the small number of places in the net. For this reason, both the maximum number of dynamic subclasses in a symbolic marking and the average length of the event list tend asymptotically to a constant value when the cardinality of class L increases beyond a given threshold. On the contrary the average length of the event list in the colored simulation program increases linearly with the cardinality of class L.

The current prototype does not provide yet any interactive simulation facility: only a batch simulation method is implemented. A single run is divided in several periods of observation (batches) to gather the statistics about the mean number of tokens in some places and the throughputs of certain transitions. A transitory period (of the same constant length for each period of observation) is discarded before considering each batch. The batch and the transitory length are expressed in terms of number of transitions firings. Both colored and symbolic simulation modules take advantage of the optimized enabling test used by the SRG module.

5.5 Perspectives and foreseen evolution

The *GreatSPN* package will continue to evolve in the future in order to follow the state of the art of the analysis techniques for timed and/or stochastic Petri nets. The three major points that have already been studied at the research level and that will be implemented in the near future are:

- Hierarchical modeling based on the concept of interfaces and equivalent subnet substitution [43];
- Parallel Processing techniques applied to the Markovian solution of large state spaces [45];
- Distributed simulation of large nets [44] using classical as well as new protocols [48].

```
class → classtype staticlist
staticlist → staticname | staticlist , staticname
classtype → O | U

static → STRING { NUMBER – NUMBER } | { objectlist }
objectlist → STRING | objectlist , STRING

placedomain → classlist | ε
classlist → classname | classname , classlist

function → ε | coefficient ID | ordinfunction
ordinfunction → coefficient optionalpredicate < functionlist >
                | ordinfunction plusop coefficient optionalpredicate < functionlist >
functionlist → functionkernel | functionkernel , functionlist
functionkernel → term | functionkernel plusop term
term → syncterm | projterm | succterm
syncterm → coefficient S | coefficient S staticname
projterm → coefficient functionname
succterm → coefficient ! functionname

marking → shortmarking | ordinarymarking
shortmarking → coefficient S
ordinarymarking → coefficient < markinglist > |
                  ordinarymarking plusop coefficient < markinglist >
markinglist → markingitem | markinglist , markingitem
markingitem → dynamicsubclassname | S staticname | S
cardinality → NUMBER | VBAR staticname VBAR
        | VBAR classname VBAR

dynamicsubclass → ( staticname : cardinality optionalorder)
optionalorder → : NUMBER | ε

predicate → predicate OR pterm | pterm
pterm → pterm AND pfatt | pfatt
pfatt → ( predicate ) | D ( STRING ) eqop doperand
         | STRING eqop stroperand
doperand → D ( STRING ) | staticname
stroperand → STRING | ! STRING

classname → STRING
functionname → STRING
staticname → STRING
dynamicsubclassname → STRING
coefficient → NUMBER | VBAR staticname VBAR
              | VBAR classname VBAR | ε
plusop → + | -
eqop → = | <>
```

Figure 5.4: SWN syntax for the GreatSPN1.7 package.

INITIAL MARKING: $p_a(1\langle L1\rangle 1\langle L2\rangle)p_w(2\langle L2\rangle) \quad |L1| = 3, |L2| = 1$

MARKING 1 (tangible 1)
$T_w(\langle L2, L1\rangle) -- >$ (rate 6.000000)
MARKING 2 # ordinary markings : 12
$p_a(1\langle L1\rangle 1\langle L2\rangle 1\langle L3\rangle)p_p(1\langle L3\rangle)p_w(1\langle L2\rangle) \quad |L1| = 2, |L2| = 1, |L3| = 1$

MARKING 2 (vanishing)
$t_w(\langle L3\rangle) -- >$ (rate 1.000000)
MARKING 3 # ordinary markings: 6
$p_a(1\langle L1\rangle 1\langle L2\rangle)p_w(1\langle L2\rangle) \quad |L1| = 2, |L2| = 2$

MARKING 1 (tangible 1)
$T_w(\langle L2, L2\rangle) -- >$ (rate 2.000000)
MARKING 4 # ordinary markings : 4
$p_a(1\langle L1\rangle 1\langle L2\rangle)p_p(1\langle L2\rangle)p_w(1\langle L2\rangle) \quad |L1| = 3, |L2| = 1$

MARKING 4 (vanishing)
$t_w(\langle L2\rangle) -- >$ (rate 1.000000)
MARKING 1 # ordinary markings: 4
$p_a(1\langle L1\rangle 1\langle L2\rangle)p_w(2\langle L2\rangle) \quad |L1| = 3, |L2| = 1$

Figure 5.5: A portion of the SRG for the random polling system SWN model.

	Cyclic polling				Random polling			
	SRG		RG		SRG		RG	
$\|L\|$	# tang.	# van.	# tang.	# van.	# tang.	# van.	# tang.	# van.
4	81	96	312	384	30	30	312	384
5	192	240	960	1200	39	39	960	1200
6	462	576	2736	3456	48	48	2736	3456
7	1056	1344	7392	9408	57	56	7392	9408
8	2412	3072	19200	24576	66	66	19200	24576
9	5376	6912	48384	62208	75	75	48384	62208
10	11928	15360	119040	153600	84	84	119040	153600

Figure 5.6: SRG versus RG size for the random polling SWN model.

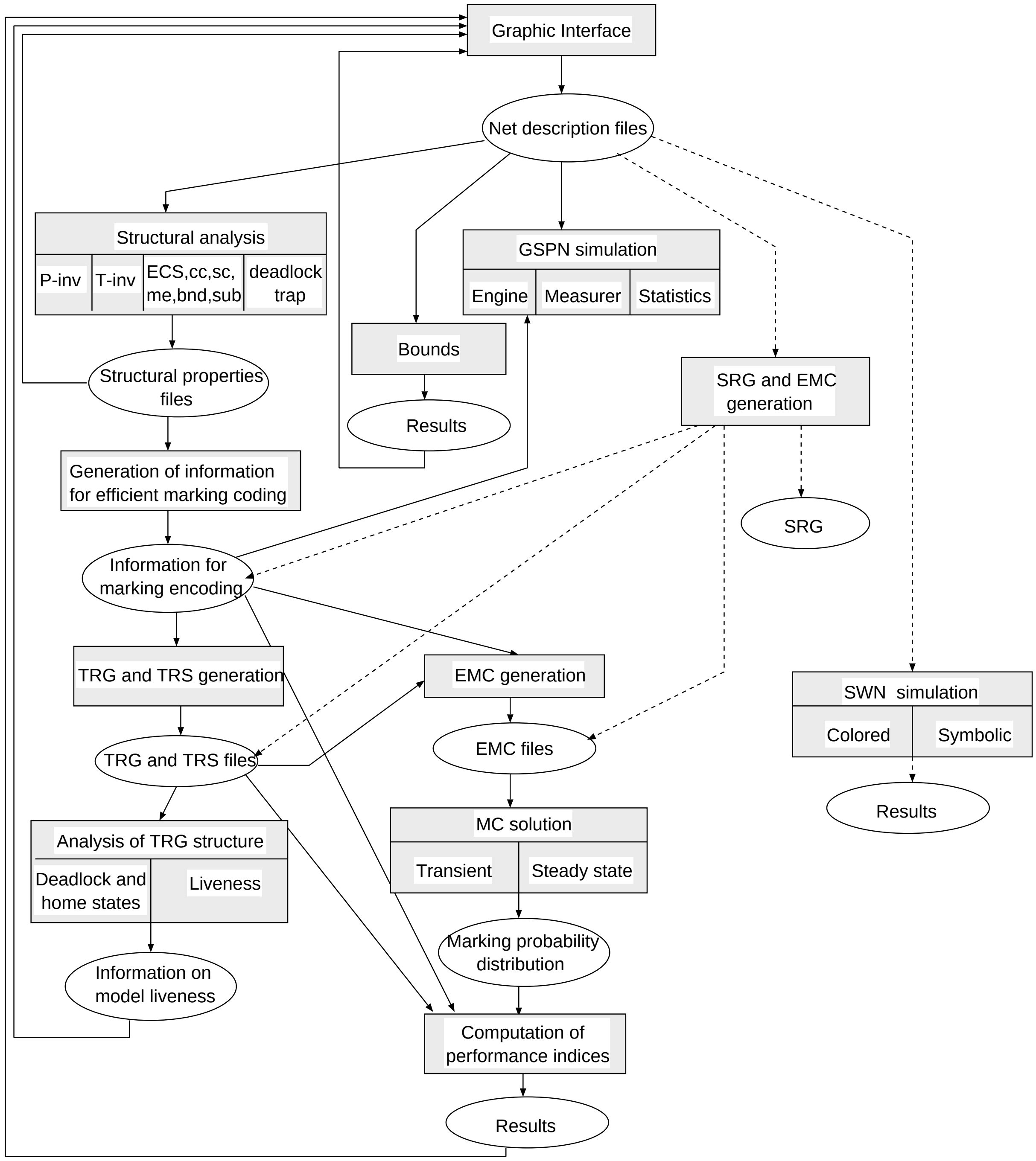


Figure 5.7: Graphic representation of the modular structure of *GreatSPN* 1.7.

Chapter 6

Applications to Parallel Computer Architectures

6.1 Introduction

An ideal modeling tool for the study of complex systems such as parallel computer architectures has at least three main requirements that partially contradict one the other:

1. it should be based on a very simple and intuitive formalism, so that it can be used by people that are not expert modelers; a graphic representation is usually preferred by users if the formalism has the capability of conveying the main ideas in a few small pictures;
2. it should be powerful enough so that all basic problems and mechanisms in the selected application domain may be represented in a concise and efficient way;
3. it should be amenable to efficient analysis or (at least) simulation.

High level Petri nets have been pushed by several academic researchers as well as profit organizations as a good trade-off among these main requirements. In particular, the idea of folding the behaviour representation expressed in net form by identifying several token types in the places is most effectively exploited for the representation of symmetric systems.

Although the proposed formalisms (Pr/T, CPN, etc.) have the power to handle the folding of inhomogeneous process behaviour, up to the extreme case in which the complete system state is encoded in the data structure associated with a single token in a single place (manipulated by a single transition) [49], this power is never fully exploited in practice. The practical reason for stopping the folding process at a given point is that this folding in general shifts the complexity from the graphic structure to the textual description. Usually the only practical case in which the modeler feels him/herself confident to be able to master the complexity of color inscriptions in high level Petri net model is when a substantial amount of symmetry is inherent to the color definition part [14, 19]. Hence, well-formed colored nets (WNs) [5, 6] where only two basic forms of symmetries give the basis of color structures are not usually perceived as a severely restricted formalism for practical applications. On the contrary, the simplicity of their basic constructs might be seen as an advantage from the point of view of understanding the model specification by non experts. On the other hand, the syntactic restrictions of the formalism provide the condition for efficient analysis as well as simulation algorithms [47, 25] to deal with models of large systems.

Timed and stochastic WNs appear then as a very good modeling formalism for highly symmetric systems: their graphic representation can be kept small and easy to understand, their color structures do not "hide" complexity of the model behaviour, the actual size of the system in terms of number of similarly behaving module components may be parametrized in the color class cardinality and may (in several cases) affect only marginally the complexity of the model analysis and/or simulation.

Most parallel computer architectures, on the other hand, tend to be extremely regular interconnections of (small or large quantities of) uniformly behaving components. The design and implementation is heavily based on such symmetry properties, so that it appears quite natural to base also the modeling techniques used for their behavioural and/or quantitative evaluation on the same symmetry properties for better efficiency.

The purpose of this last chapter is to assess the adequacy of the SWN formalism for behavioural and performance modeling of typical problems and mechanisms found in parallel architectures. We carry out this assessment by developing and studying models of classical problems in this domain. The rest of the chapter is organized as a set of independent sections, each one devoted to the development of efficient SWN models for the study of different aspects and mechanism related to parallel processing. Each considered example is characterized by some intrinsic symmetry that is exploited using the SWN formalism in order to simplify both the model structure and its analysis/simulation. The last section resumes the main results obtained on these case studies and contains an organic discussion of the practical modeling power of the formalism for the particular application domain.

6.2 A Classical NUMA Multiprocessor

In this section we apply the WN formalism to the modeling and performance evaluation of a multiprocessor system already studied in [50, 9, 12]. 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., [14, 17]. Second, the solutions based on Markov or Stochastic Petri net techniques presented in [50, 9] 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 [12], 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 behavioural 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 [50, 9]. On the other hand we compare the aggregations automatically performed in the SRG generation phase with those devised in [12] at the price of a thorough (human) analysis of the system behaviour 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

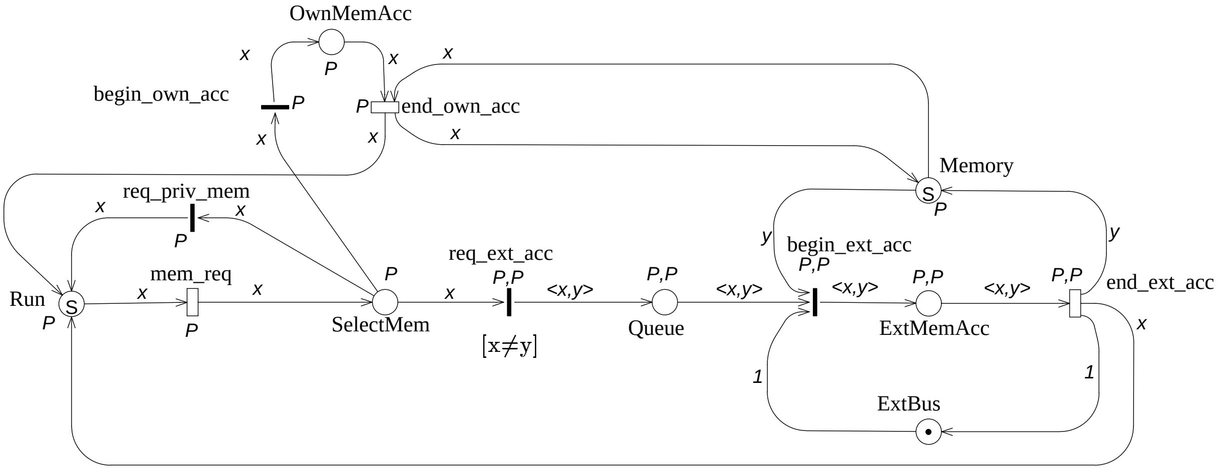


Figure 6.1: An intuitive WN model of the multiprocessor architecture.

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.

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 behaviour 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 behaviour of the system is described in a straightforward manner by the WN model in Figure 6.1. 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

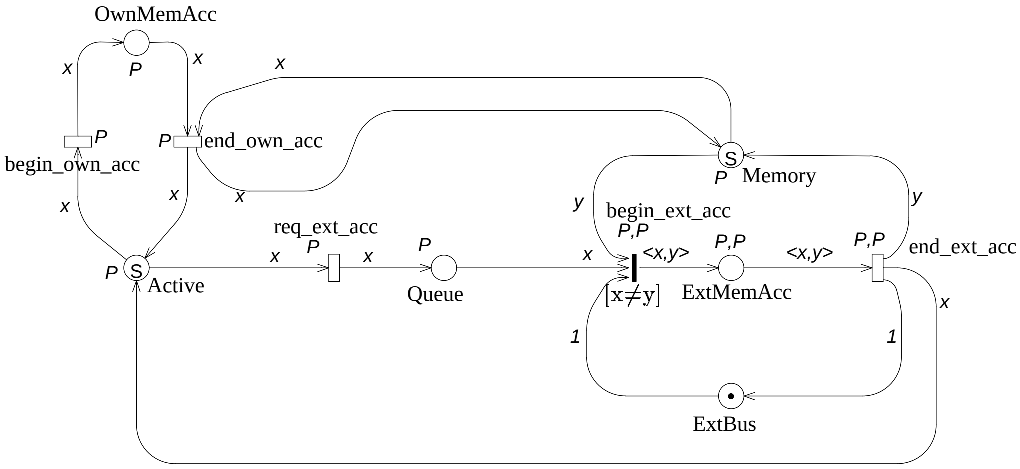


Figure 6.2: A more compact WN model of the multiprocessor architecture.

preserve all behavioural properties of the net [33] as well as the timing semantics of the model [40]. Moreover, the timed transition resulting from the fusion of transitions 'mem_req' and 'req_priv_mem' in the WN in Figure 6.1 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 6.2. 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[1]; 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.[2]

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 *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 [50, 9, 12], we define the system load factor ρ as the ratio λ/μ. Some performance figures

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

[2] 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 6.1: Results.

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 6.1 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 6.1 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 [12]. 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 behaviour 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 model shown in Figure 6.3. Notice that the construction of the WN description, as compared to the folded GSPN model originally presented in [12], 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.

6.2.1 A decolourization example

The net in Figure 6.4 models the same NUMA multiprocessor system. The WN is a variation of the WN model already depicted in Figure 6.2 that is behaviourally equivalent in teh sense that it generates the same SRG structure.

Tokens in place *Active* represent processes in the active state. Tokens in place *OwnMemAcc* represent processes that are in the "own memory access" state. Transitions *bgn_own_acc* and *end_own_acc*

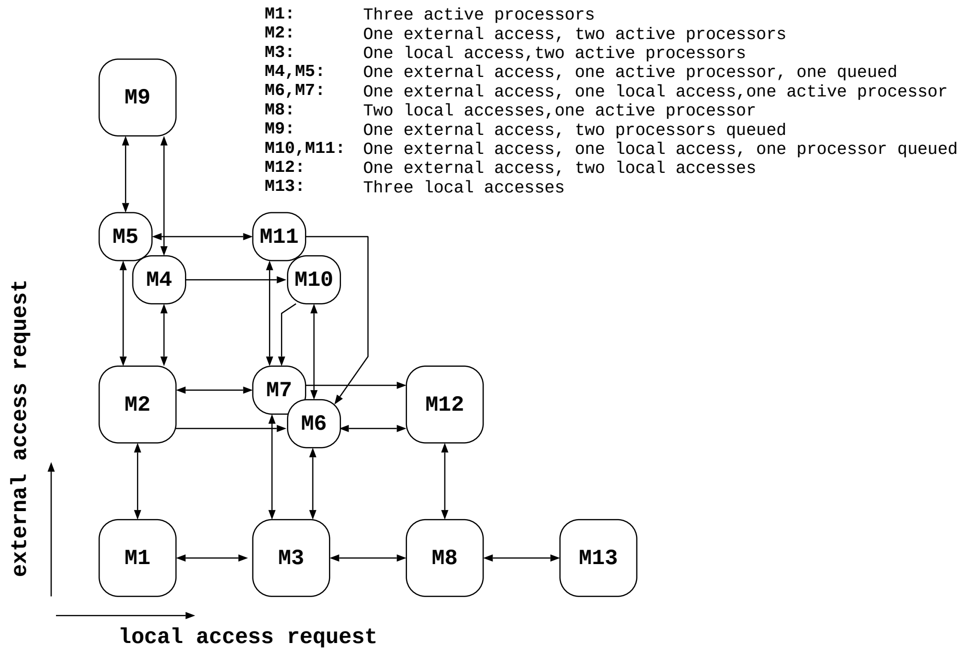


Figure 6.3: SRG of the multiprocessor WN.

represent the state transitions between the above two states. Tokens in place *Queue* represent processes waiting for an external memory access. Transition *req_ext_acc* represents the transition from the active to the "waiting for an external access" state. Tokens in places *Preempt*, NPY, and NPP represent processes performing an external memory access; their colour is a pair with the first component representing the identity of the processor that is performing the access and the second component representing the identity of the processor owner of the accessed memory. A token is in *Preempt* when the memory owner processor is preempted by the external access; it is in NPY when the memory owner processor is not preempted at the moment but could try to perform a local memory access (i.e., it is currently in the active state) and become blocked by the external access; it is in NPP if the memory owner processor is not preempted and it cannot try to perform a local memory access any more till the end of the current access to its own memory (i.e., it is waiting to perform an external access). Transitions *req_own_bis* and *req_ext_bis* represent the state transitions between the three cases of external memory access described above. The state transitions from the "waiting for an external memory access" state to the "accessing external memory" state are represented by transitions s_P, s_NPY, and s_NPP. The state transitions from the "accessing external memory" state back to the active state are represented by transitions e_P, e_NPY, and e_NPP.

This net is covered by a unique, minimal consistent projection support family that includes all colour components. It is easy to verify that all decolourization requirements are satisfied: indeed no synchronization depends on the identity of the tokens since there is no relation between functions x and y.

In this case the unique colour set is not partitioned into static subclasses and it can be completely decolourized. All colour annotations on places, transitions, and arcs can be cancelled, so that the decolourized net is a P/T net. All arcs in the decolourized net have weight one except the arcs from *Queue* to s_NPP and from e_NPY to *Active* that have weight two. The decolourized net happens to be identical to the GSPN model proposed in [12].

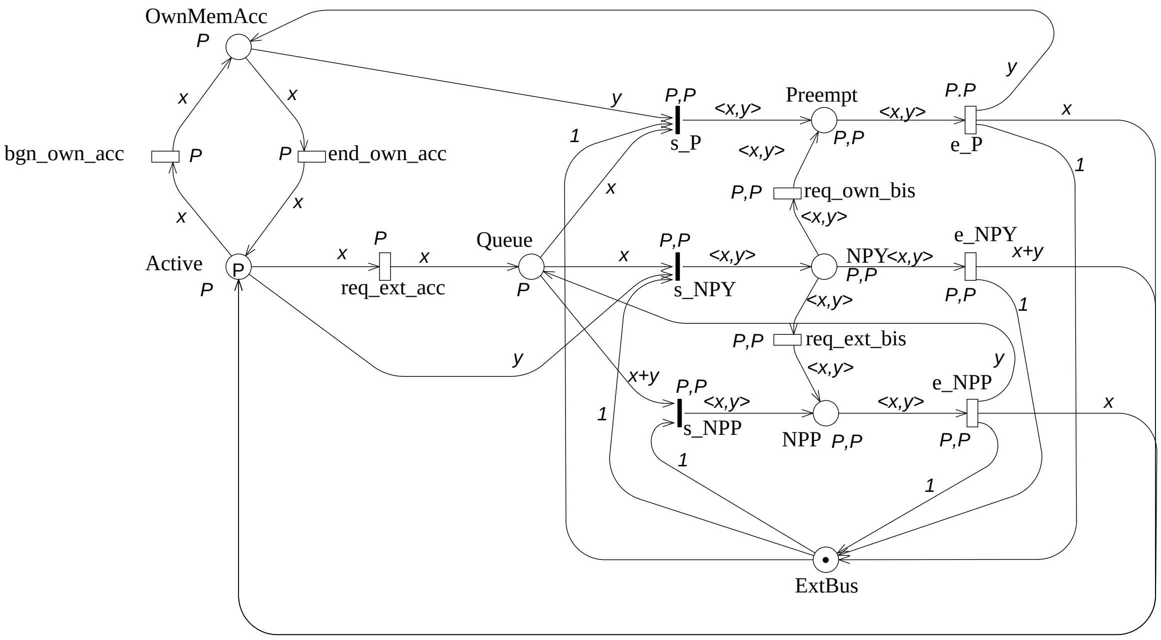


Figure 6.4: A multiprocessor system WN model that can be completely decolourized.

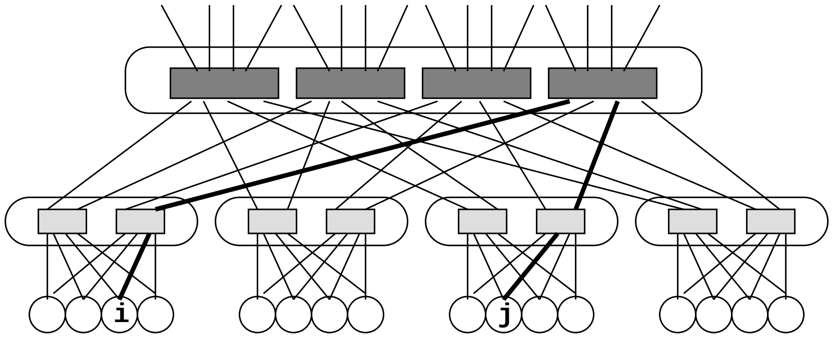


Figure 6.5: A 4-ary fat tree (from [52], Fig. 8.29).

6.3 Multilevel Fat Trees

We consider now an interconnection network similar to that used in the Connection Machine CM-5. This interconnection network, introduced by Leiserson in [51], is called *fat-tree*. The net structure is a k-ary tree, whose leaves are the processors while the internal nodes are switching elements. The tree structure is called *fat* because the branches of the tree closer to the root are "thicker" (i.e., they may support a larger number of communications in parallel).

The CM-5 data network is implemented with a 4-ary fat tree. Figure 6.5 depicts such interconnection network: each internal node is made up of several router chips; when a process mapped on processor i wants to communicate with a process mapped on processor j, the message climbs up towards the root of the tree until the closest common ancestor is found, then it descends towards the destination processor leaf.

The hierarchical structure of the network naturally suggests a hierarchical naming schema for the tree nodes: each node is uniquely identified by a pathname. Since we are dealing with a $4-$ary tree, we can label all the nodes with common father with four different labels, e.g., a, b, c, and d, and then identify a generic node "n" with the sequence of labels associated with the nodes on the path from the root to "n". Consider the three-levels tree in Figure 6.6, the pathname for processor j is $\langle a, a, d\rangle$,

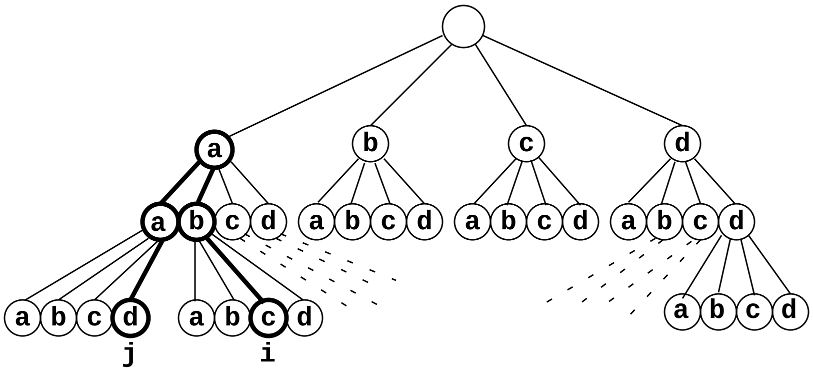


Figure 6.6: Naming nodes through pathnames in a 3 level 4-ary tree.

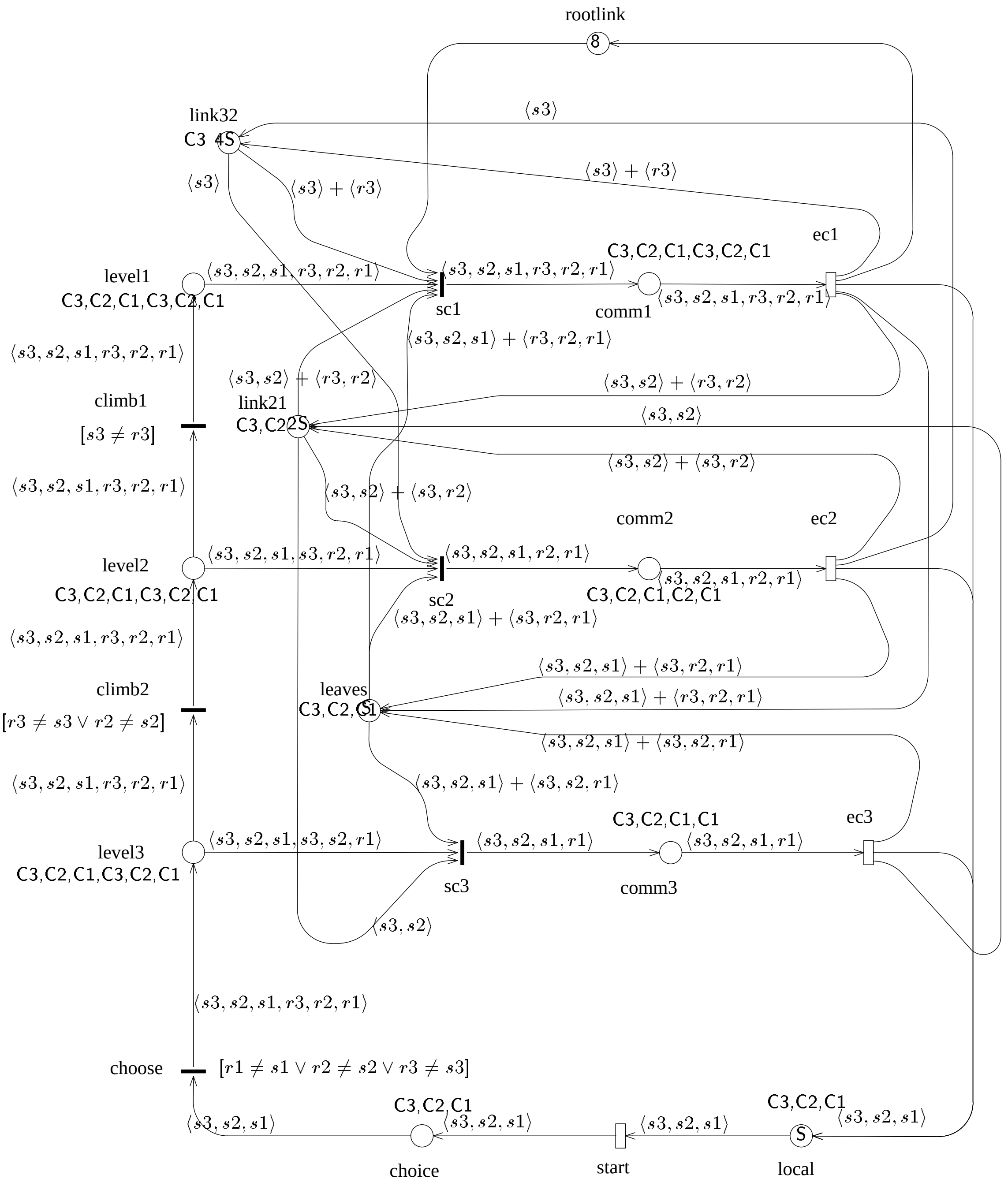
while the pathname for processor i is $\langle a, b, c\rangle$. Their closest common ancestor is identified by the pathname $\langle a\rangle$. Notice that the pathname of the closest common ancestor of two nodes corresponds to the longest common prefix in their pathnames. This encoding of the net nodes easily translates in a specification for the color structure of the SWN model of the interconnection network: k-ary trees of depth D require a single (non ordered) color class C of cardinality k, and a token representing a node at depth level[3] j $(1 \leq j \leq D)$ is a tuple of arity j.

The SWN in Figure 6.7 represents a 4−ary fat tree of depth three. The model includes a very simple specification of the messages workload: each processor "p" runs one process that performs local computation for an average time of $1/\lambda$ (transition *start* represents the end of a local computation), then it sends a message to a randomly chosen destination (the firing of immediate transition *choose* represents the choice of a destination); the probability of chosing a given destination is uniformly distributed among all remaining processors in the system.

The presence of a 6-tuple $\langle s_3, s_2, s_1, r_3, r_2, r_1\rangle$ in place $level3$ means that a message is being sent from processor $\langle s_3, s_2, s_1\rangle$ to processor $\langle r_3, r_2, r_1\rangle$. The routing of a message is modeled by a three-level subnet. The level i subnet contains two places $level_i$ and $comm_i$, and two transitions sc_i and ec_i. Transitions $climb_j$ represent the search for the sender-receiver common ancestor (predicates are associated with these transitions that compare the prefixes of the two nodes up to the considered level). Places $rootlink$, $link32$, $link21$ and $leaves$ represent the resources needed to set up the communication between the sender and receiver processors: the initial marking of these places is set to the maximum number of concurrent communications that can pass through a switch at a given level. For example the initial marking of place $rootlink$ is 8, hence at most 8 communications that need to be routed through the root may be concurrently served. Place $link21$ initially contains two copies of each identifier $\langle s_3, s_2\rangle$ representing a switch at level two meaning that at most two communications can be concurrently routed through the switch $\langle s_3, s_2\rangle$. Finally, place $link32$ initially contains four copies of each identifier $\langle s_3\rangle$ representing a switch at level one meaning that at most four communications can be concurrently routed through the switch $\langle s_3\rangle$.

Observe that any k-ary fat tree with three levels can be modeled by the same net structure, changing only the definition of the basic color class C, so that $|C| = k$. On the contrary, k-ary fat trees with $d \neq 3$ levels require that the place and transition color domains and the arc functions are changed to reflect the fact that processor pathnames have length d; moreover the routing subnet should comprise d submodels, each representing the climbing of one tree level, to model the fact that messages may need to climb up to d levels towards the root.

[3] The depth level of the root is 0, while for the leaves it is D.



$$C1 = \{a, b, c, d\}; \quad C2 = \{a, b, c, d\}; \quad C3 = \{a, b, c, d\}$$

Figure 6.7: First SWN model of the messages routing in a 3 level 4-ary fat-tree.

The model is thus fully parametrized with respect to the arity k of the tree, while its structure changes when the number of levels d changes. Notice that the net structure is highly regular with respect to parameter d as well, and a folding of the subnets representing the tree climbing could be obtained at the price of using much more complex arc functions (guarded functions are needed to model the fact that a different number of resources —i.e., switches— have to be acquired depending on the number of climbed levels). In the SWN in Figure 6.7 there are actually three color classes: $C1$, $C2$, $C3$ all of cardinality four: class Ci is used to represent labels of nodes at level $d-i+1$. This is to stress the fact that the algorithm has to consider the colors used to identify objects at different levels

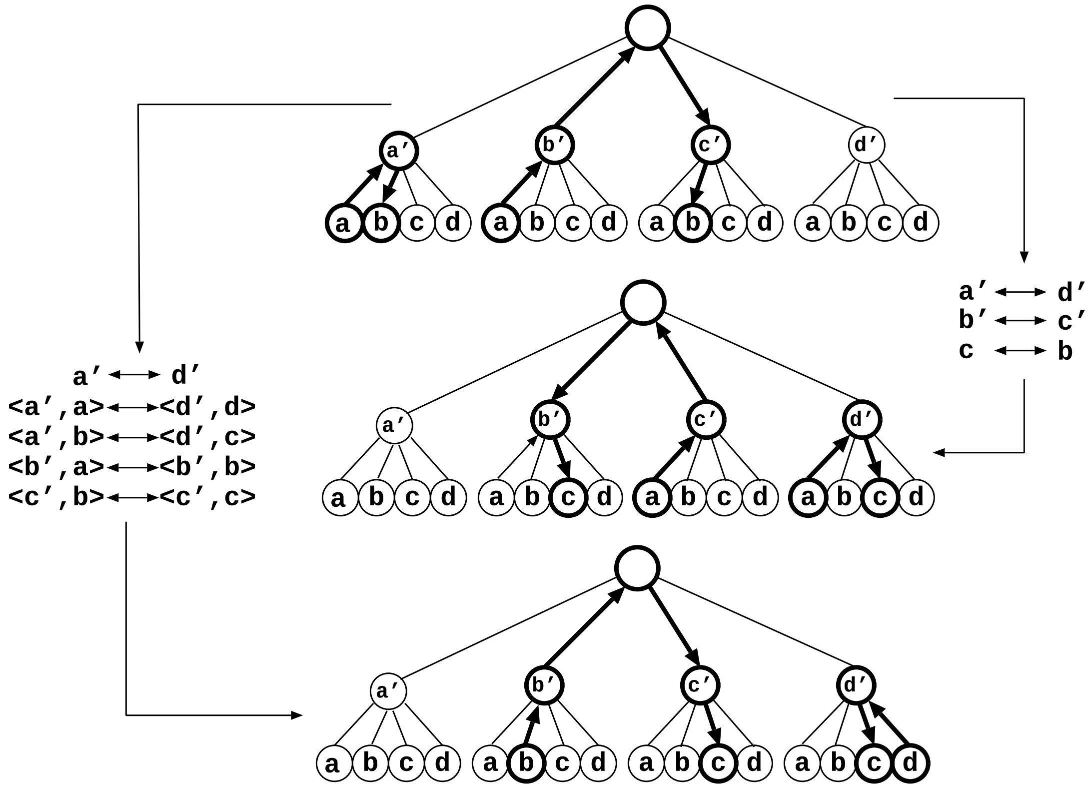


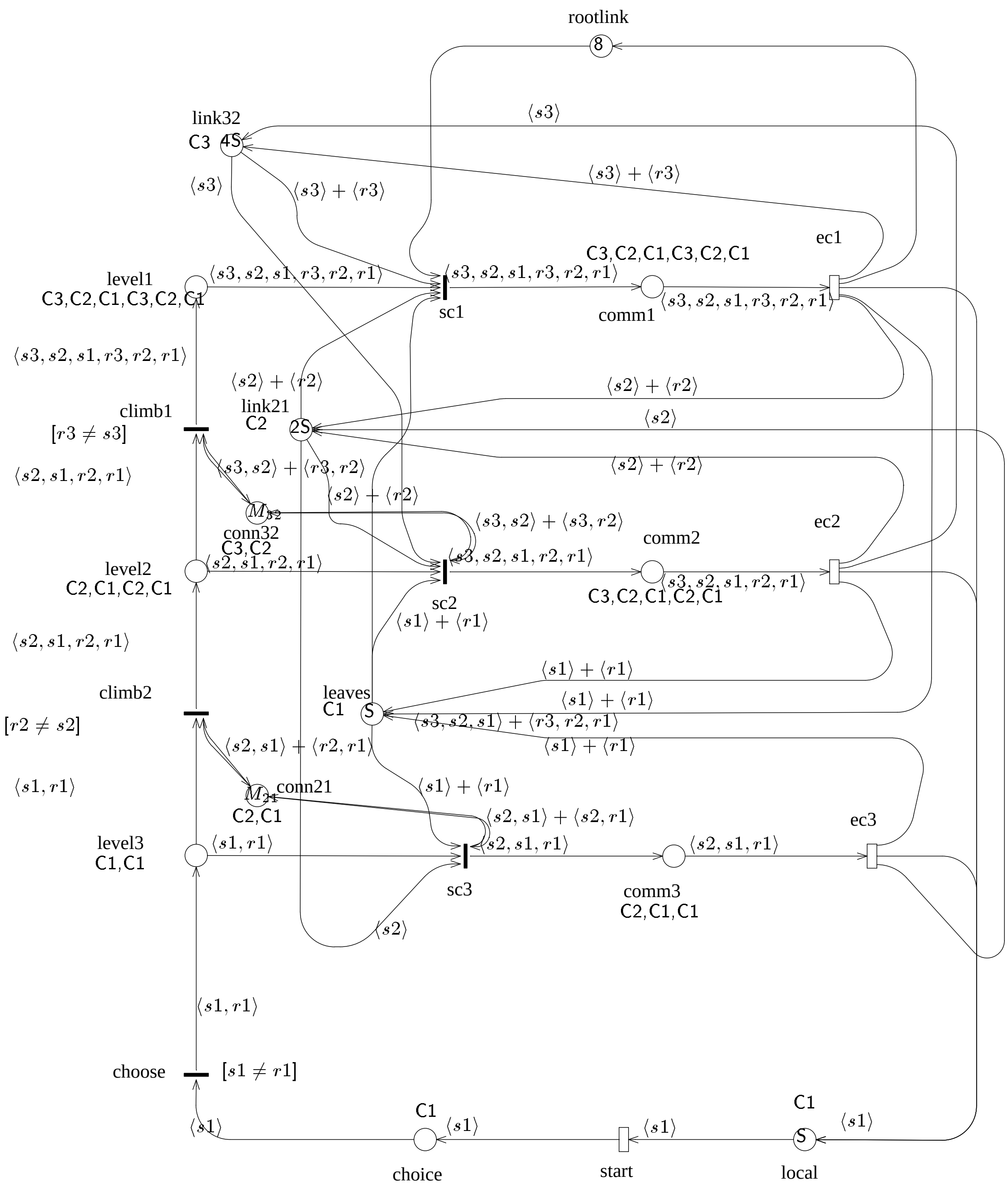
Figure 6.8: Equivalent markings.

of the tree as independent. If the three color components were not considered as independent by the SRG construction algorithm, the degree of aggregation achieved would be substantially decreased.

We first ran our prototype SRG computation program on a model representing a 2-level binary tree, obtaining a SRG with 184 tangible and 731 vanishing symbolic markings. The size of the ordinary RG for this model was 505 tangible and 2020 vanishing markings (thus obtaining an average compaction factor of about 3 by using the SRG instead of the RG technique in this case).

By analyzing the symbolic markings we observed that the degree of reduction achieved was lower than expected. This was due to the fact that the specified color structure introduces an unnecessary dependence between nodes that have different fathers but same associated color characterizing the postfix of the pathname. Let us explain the problem on the following example. Consider the three states depicted in Figure 6.8: all of them represent a situation in which two concurrent communications are in progress, the former involving two leaves l_1 and l_2 with a common father f_1, while the latter traverses two levels and involves two leaves with father different from f_1. Intuitively we would like to consider these markings as equivalent. Instead the SRG keeps the first two in the same symbolic marking while it associates the third with a different symbolic marking. The equivalence relation that we have in mind relates any two states that can be obtained one from the other by permutation of the nodes at the same level with the only constraint of preserving the father-children connection. The equivalence relation used by the SRG generation algorithm, instead, adds a further constraint due to the fact that the permutations used to define the symbolic markings operate at the basic color class level. Since leaves with different fathers are associated with labels from the same class $C1$ in the model, an example of allowed permutation is the exchange of objects a and c in class $C1$, that has the effect of simultaneously permuting nodes $\langle x, y, a\rangle$ and $\langle x, y, c\rangle$ for all $\langle x, y\rangle \in C3, C2$, thus preventing the possibility of exchanging at the same time $\langle x', y', a\rangle$ and $\langle x', y', b\rangle$ for some $\langle x', y'\rangle$ in $C3, C2$.

An easy solution to this problem is obtained as follows. Define class Ci (representing the nodes



$$C1 = \{aaa, aab, \cdots, ddc, ddd\}; \quad C2 = \{aa, ab, \cdots, dc, dd\}; \quad C3 = \{a, b, c, d\}$$

Figure 6.9: SWN model of 3 level 4-ary "fat tree" interconnection capturing all permutation symmetries.

at level $d-i+1$) as a color set of cardinality k^{d-i+1}, and then represent the father-children relation of the tree structure in the initial (symbolic) marking. For example in the SWN model of a 3 level 4-ary tree depicted in Figure 6.9, color classes are defined as follows: $C1 : |C1| = 64$, $C2 : |C2| = 16$, $C3 : |C3| = 4$. Moreover places $conn21$ represents the father-children relation at the leaves level (i.e., a pair $\langle x, y\rangle, x \in C2, y \in C1$ is in this place iff x is father of y) while place $conn32$ represents the father-children relation one level above. The initial marking of these places is defined in a symbolic form:

class $C1$ is partitioned into 16 *dynamic subclasses* ($Z_1^i, i = 1, \ldots, 16$) of cardinality four[4] (objects in the same dynamic subclass represent siblings); 16 dynamic subclasses ($Z_2^i, i = 1, \ldots, 16$) of cardinality one are defined for class $C2$; 4 dynamic subclasses ($Z_3^i, i = 1, \ldots, 4$) of cardinality one are defined for class $C3$. The initial marking of place $conn21$ is then defined as follows:

$$\mathcal{M}_0(conn21) = \sum_{i=1}^{16} \langle Z_2^i, Z_1^i \rangle$$

meaning that the object represented by subclass Z_2^i is father of the four objects in subclass Z_1^i. Place $conn32$'s initial marking is defined as:

$$\mathcal{M}_0(conn32) = \sum_{i=1}^{4} \sum_{j=1}^{4} \langle Z_3^i, Z_2^{4(i-1)+j} \rangle$$

meaning that the object represented by subclass Z_3^1 is father of the four objects represented by subclasses $\{Z_2^1, \ldots Z_2^4\}$, etc.

The initial symbolic marking represents several initial ordinary markings corresponding to all the possible ways of connecting labeled nodes in a father-children relation consistently with the desired tree structure. As a consequence, the SRG represents $|\mathcal{M}_0|$ ordinary reachability graphs (the different RGs are not connected with each other).

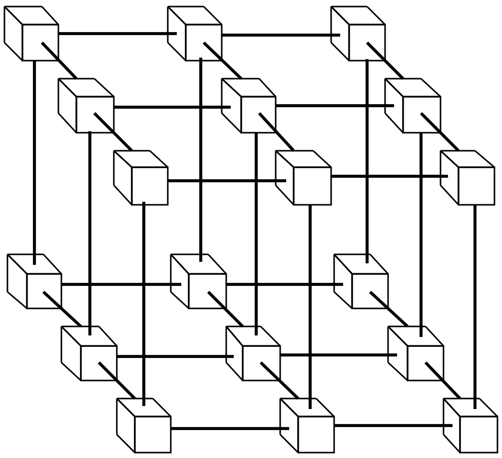
This new representation allows us to achieve the goal of allowing independent permutations within sets of siblings associated with different fathers. Applying the SRG computation algorithm to an SWN model of a 2-level binary tree represented in the new proposed form, 113 tangible and 301 vanishing symbolic markings are generated. The model can be further refined to reduce the number of vanishing markings by "fusing" immediate transitions *choose*, *climb*2 and *climb*1 into timed transition *start* using the transformations described in [40]. The reduced model of the two-level binary tree has 113 tangible and 73 vanishing symbolic states, thus yielding a compaction factor of about 4.5 on tangible markings.

The solution of a 2-level ternary tree was already beyond the possibilities of our SRG construction program prototype (the current implementation is rather inefficient). Even if the SRG generation did not complete for time and space limits, we could analyze the SRG portion generated (that included 9232 tangible and 5447 vanishing symbolic states) to evaluate the level of aggregation achieved. The initial symbolic marking represents 1680 ordinary markings, while the symbolic markings average cardinality is in the order of one million of states. Hence the average compaction factor (i.e., the ratio between number of states in the ordinary RG and number of symbolic states in the SRG), obtained by dividing the average symbolic marking cardinality by $|\mathcal{M}_0|$ gives about 700 (please compare with the compaction factor of about 4.5 for the 2-level binary tree). Similar levels of improvement can be obtained in simulation by using the symbolic simulation technique instead of the usual event-driven simulation for colored Petri nets.

6.3.1 Discussion

In this section we have developed a modeling exercise with the aim of highlighting an aspect of SWN color structure definition that may influence the degree of state space reduction achieved using the SRG construction algorithm. The equivalence relation that defines symbolic markings is given in

[4] The cardinality completely characterizes a dynamic subclass; a partition of a class in dynamic subclasses denotes *any* partition satisfying the given subclass cardinality constraints.



```
Computation task:
  while TRUE do
      local_computation;
      neighbour:= select(random);
      send_request(neighbour);
      receive(neighbour);
 od

Server task:
  while TRUE do
      receive(neighbour);
      compute;
      send_answer(neighbour);
   od;
```

(a) (b)

Figure 6.10: (a) $3 \times 3 \times 2$ mesh (b) workload description.

terms of permutation of objects within the same color class, as a consequence the reduction factor is proportional to the product over the set of color classes of the number of possible permutations in each class. Hence it is important to distinguish classes used to denote different objects even if they have the same cardinality. Since it is possible to automatically check on the model structure for the color classes independence, it would be helpful to implement this feature in the SRG generator to overcome a source of incomplete reduction.

It is also important to avoid creating unnecessary dependence among objects belonging to the same set. In the model of a k-ary tree with d levels described in this section, the first encoding chosen for describing nodes used classes of cardinality k for each level. As a consequence only a maximal potential reduction proportional to $(k!)^d$ is achievable. Indeed, this encoding introduced an artificial dependence between nodes at the same level. By defining a cardinality k^i class to describe the set of nodes at level i, a much higher potential reduction degree can be achieved. In our example, the degree of reduction achieved on a ternary tree with two levels was about 36 using the first model and 700 using the second model. For larger tree configurations (4-ary trees of depth ≥ 2) the compaction factor becomes enormous, so that its exploitation becomes the only hope to attack the analysis of such models, not only in the case of state space enumeration but also in the case of discrete event simulation.

6.4 Multidimensional Mesh interconnection

Let us consider MIMD distributed-memory architectures with n-dimensional mesh interconnection structure, possibly having a different number of processors in each dimension. Figure 6.10 depicts a 3-dimensional structure with a number of nodes in the vertical direction different from the number of nodes in the other two directions.

6.4.1 Modeling assumptions

Let us assume that the workload is evenly distributed across all the processing elements and the computations assigned to each processor entail interaction with the local neighbors only [53]. The computation assigned to each processor consists of a set of $N \geq 1$ concurrent tasks performing a cyclic activity and one server task. The cyclic activity of the identical tasks includes a local operation phase (whose duration is a random, exponentially distributed variable with parameter λ), followed by a request for service addressed with equal probability to one of the neighbors. Service time is also a random, exponentially distributed variable with parameter μ. Furthermore let us consider two interaction policies:

- *preemptive interaction*: requests coming from the neighbors are served with higher priority than the local activity, preempting any computation task; no particular local task scheduling policy is specified: only the priority of server processes over local tasks is assumed;
- *non-preemptive interaction*: requests are not preemptive, but are served at the end of the local operation with higher priority with respect to the local computation tasks; this policy implies that local tasks are run by the processor up to their completion, so that for instance round-robin scheduling is excluded; on the other hand, no further scheduling policy assumption is made besides the fact that local tasks are run one at a time, without interruption up to their completion.

We assume that communication delays are negligible in our model. The addition of two delay transitions to take them into account would be however straightforward.

As an example of performance measure we can consider the node utilization for different levels of multiprogramming N.

6.4.2 Modeling the interconnection symmetry

The mesh structure depicted in Figure 6.10 looks extremely regular. Despite its regularity it is not fully symmetric. Internal nodes in the structure show indeed the same interconnection pattern with their nearest neighbor nodes. However part of the regularity is lost when we consider the "boundary" nodes, which show a reduced number of connections compared to the internal nodes.

A "slight" modification of this interconnection schema is the folding obtained by addition of one link connecting the last node to the first one in each dimension. In this way we obtain an interconnection schema called (n-dimensional) torus, which is instead completely symmetric.

For the sake of simplicity let us first consider the case of 1-dimensional torus, i.e., a set of m processing units connected in a ring. Let us consider a very simple WN model that represent the choice of one neighbor from a node that wants to communicate. This (trivial) WN model is depicted in Figure 6.11. Each processing node is uniquely associated with one element of the ordered basic class C. The predicate associated with the enabling of transition "choose" restricts the identity of the receiver to be either the previous or the next element with respect to the sender.

The symmetry of the structure is captured by the WN model that allows a reduction of $1/|C|$ in the number of symbolic markings with respect to the number of ordinary markings. The use of SWN models for n-dimensional torus topologies (composed as explained later on by Cartesian product of the color class C) yields a compaction factor of $1/\prod_1^n |C_i|$, and can thus be considered natural and efficient at the same time; indeed it results in models that are parametric on the number of nodes, whose color structure is very easy to understand, and that are efficient to analyze using either the SRG or the symbolic simulation technique.

The correct treatment of boundary conditions for mesh interconnections requires the introduction of predicates that distinguish inner node identities from boundary node identities. A careless and direct

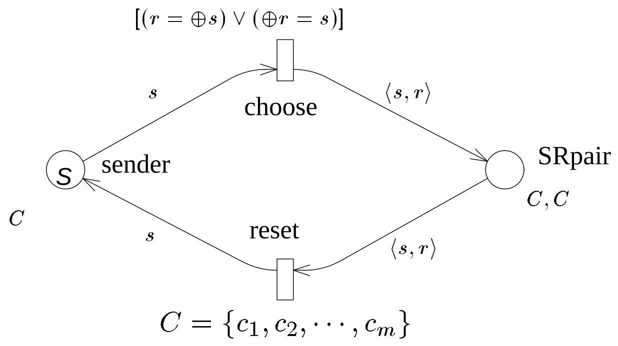


Figure 6.11: Basic color definitions for nearest neighbor identification in 1-dimensional torus (ring).

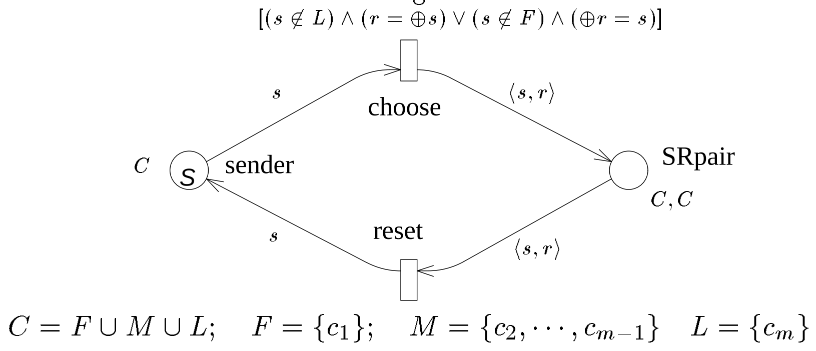


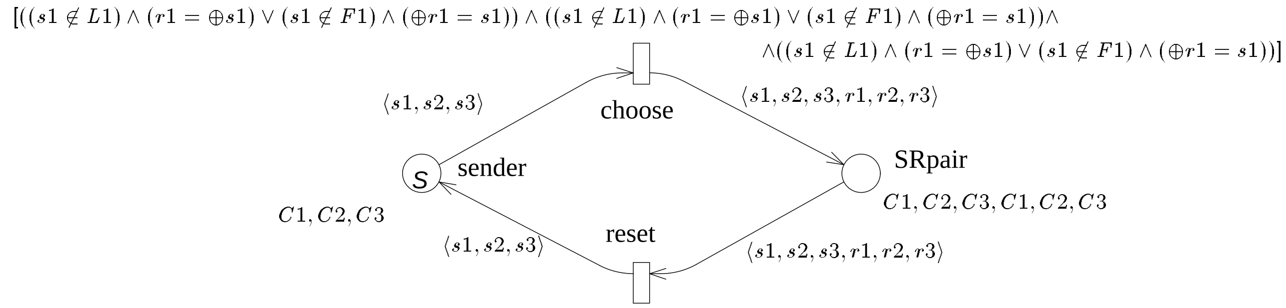
Figure 6.12: Basic color definitions for nearest neighbor identification in 1-dimensional mesh.

introduction of such predicates completely destroys all symmetry properties of the WN model, thus giving rise to models that are still easy to draw and describe, somewhat cumbersome to understand for what concerns the treatment of boundary conditions, and impossible to analyze efficiently. Of course, one might "approximate" the behaviour of a mesh interconnection by ignoring the anomalies due to the boundary nodes and using the torus model instead of the mesh model. In this example, however, we are interested in representing such boundary anomalies in an accurate way in order to compare the performance of mesh versus torus interconnection. We first introduce a direct representation of the structure in terms of color structure that has all the problems just mentioned, and then refine the model in order to be able to capture and exploit those symmetries that are anyhow present in the system.

The case of 1-dimensional mesh can be obtained by refining the predicate associated with transition "choose" in order to distinguish the two boundary nodes from the others. This WN model is depicted in Figure 6.12. The ordered class C is now partitioned in three subsets, so that it is possible to identify the first (the only element of subclass F) and the last (the only element of subclass L) elements. The predicate associated with the enabling of transition "choose" restricts the identity of the receiver to be either the previous or the next element with respect to the sender identity for internal nodes. Boundary nodes have only one neighbor (the next for node c_1 and the previous for node c_m).

The extension to higher dimension levels is trivial both from a conceptual and a practical point of view. Figure 6.13 depicts the color structures and functions for the 3-dimensional case. The predicate restricting the enabling of transition "choose" increases in complexity in order to handle the boundary condition in each dimension. The mapping of the interconnection structure into color structures remains however straightforward to understand.

Unfortunately, the partitioning of the ordered color classes (that is necessary to distinguish boundary conditions by means of predicates) destroys all useful model symmetries: symbolic marking rep-

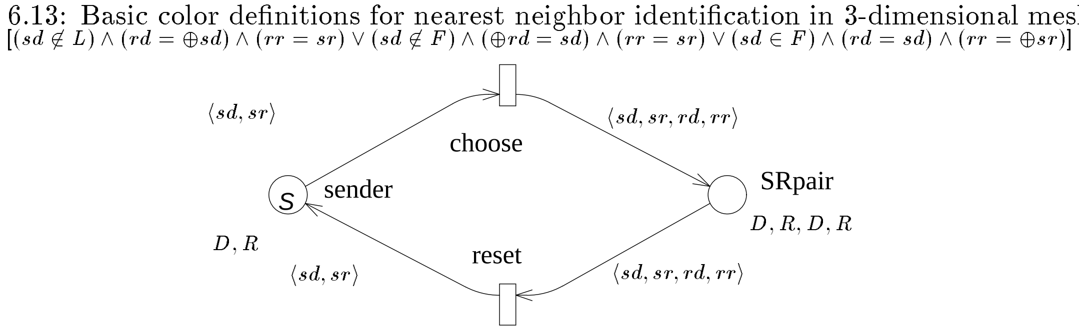


$$C1 = F1 \cup M1 \cup L1; \quad F1 = \{a_1\}; \quad M1 = \{a_2, \cdots, a_{m_a-1}\} \quad L1 = \{a_{m_a}\}$$

$$C2 = F2 \cup M2 \cup L2; \quad F2 = \{b_1\}; \quad M2 = \{b_2, \cdots, b_{m_b-1}\} \quad L2 = \{b_{m_b}\}$$

$$C3 = F3 \cup M3 \cup L3; \quad F3 = \{c_1\}; \quad M3 = \{c_2, \cdots, c_{m_c-1}\} \quad L3 = \{c_{m_c}\}$$

Figure 6.13: Basic color definitions for nearest neighbor identification in 3-dimensional mesh.



$$D = F \cup M \cup L; \quad F = \{d_1\}; \quad M = \{d_2, \cdots, d_{m-1}\} \quad L = \{d_m\}; \qquad R = \{l, r\}$$

Figure 6.14: Basic color definitions for nearest neighbor identification in 1-dimensional mesh capturing specular symmetry.

resentations collapse to ordinary marking representation, thus making the complexity of the model analysis to increase dramatically with both the number of dimensions and the size in each dimension. On the other hand, some kind of symmetries are still enjoyed by our interconnection schema, even if they are not directly captured by the WN formalism in Figures 6.12 and 6.13. The kind of symmetry that we should model is, in each dimension, the symmetry with respect to the central point of a segment.

The "mirror" symmetry that we need in order to efficiently handle this case is not directly supported by the WN formalism, but it can be "simulated" as a special case of "rotation" (modeled by ordered classes) in which only two basic objects are considered. Figure 6.14 depicts a more sophisticated way of modeling the linear connection using the WN formalism. The "coordinate" of each processing node is encoded using a pair of ordered basic classes, the first one representing the distance from the center of the segment, and the second one the direction (left or right) with respect to the center. The class representing the distance is still partitioned in subclasses to identify the first and last elements. The class representing the direction is instead not partitioned, thus inducing actual symmetric folding of the symbolic marking representation.

The extension from 1-dimensional to n-dimensional mesh interconnections may be as straightforward in principle as in the first model in which symmetries were not captured: one may just replicate the pair of basic classes as many times as the number of dimensions (thus also expanding the complexity of the predicate associated with transition "choose").

The degree of compaction of the symbolic marking representation is only a factor of 2 in the

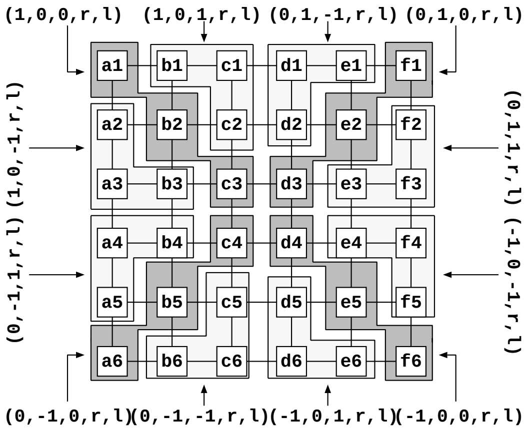


Figure 6.15: Encoding of nodes of a 2-dimensional structure.

case of 1-dimensional mesh. It increases however exponentially with the dimension, thus providing a factor 2^n of compaction in the case of n-dimensional mesh. Notice that the case of "n-dimensional hypercube" in which the number of nodes per dimension is fixed to 2 is particularly advantageous from the symmetric folding point of view: the color class D can be omitted and the coordinate of a node reduces to a set of n colors of type R for which complete symmetry is available.

Notice also that, even in the case of completely homogeneous local computation workload that we are examining, the compaction factor 2^n is the best result that we may obtain. Indeed, processing nodes at different distances from the center (or, equivalently, from the nearest boundary) are subject to different communication loads (due to the different number of neighbors on the boundary), thus making the exact aggregation of nodes at different distances impossible from a performance evaluation point of view.

In the case of equal number of components in two or more dimensions, however, another form of permutation symmetry with respect to the ordering of dimensions may be observed. In other words, if we focus on a 2-dimensional case with $2m$ homogeneous nodes in each dimension, we can observe a new form of symmetry corresponding to a flip of the structure with respect to either main diagonal. This additional symmetry can be properly taken into account at the price of some additional complication in the encoding of the coordinates of each node and in the predicate that establishes the nearest neighbor connection. In particular, the identity of a node may be encoded by means of a 5-tuple of type R, R, R, D, D of the two following basic color classes:

$$D = F \cup M \cup L; \quad F = \{d_1\}; \quad M = \{d_2, \cdots, d_{m-1}\} \quad L = \{d_m\};$$

$$R = Z \cup NZ; \quad Z = \{0\}; \quad NZ = \{-1, 1\};$$

the first one ordered (with a successor function defined among elements), and the second one not ordered. The first two elements in the encoding identify one among four "quadrants" in the mesh. The third element is zero for the nodes laying on the main diagonals, and either 1 or -1 for the other nodes (see Fig. 6.15). Hence the three components of type R are used to encode one out of eight "sectors". The last two elements of type D identify a specific node in a sector, by encoding its "latitude" and "longitude"; for instance in Fig. 6.15 the encoding of node **a1** is $\langle 1, 0, 0, 3, 1 \rangle$, that of node **f2** is $\langle 0, 1, 1, 2, 2 \rangle$, and finally that of node **b6** is $\langle 0, -1, -1, 3, 2 \rangle$.

A valid identification of a node is subject to the following restrictions: call $\langle dx, dy, dd, r, l \rangle$ a color instantiation corresponding to a node,

$$dx \neq 0 \text{ iff } dy = 0 \quad \wedge$$

$$\text{if } dd = 0 \text{ then } l = d_1 \quad \wedge$$

$$l \leq r \text{ (i.e. if } r = d_i \text{ then } l = d_j \ \wedge \ j \leq i \text{)}$$

The resulting model is cumbersome to draw and to explain, so that it is not reported here. The compaction factor in this case of 2-dimensional mesh varies with number of nodes per dimension from a minimum of 4 in the case of $m = 1$ (i.e., a 2-dimensional hypercube) to a limiting value of 8 in the case $m \to \infty$ (which corresponds to the number of symmetric sectors defined in the color encoding).

6.4.3 Modeling non-preemptive interaction

The first interaction policy considered is the non-preemptive one, in which external requests are not preemptive, but are served at the end of the local operation with higher priority with respect to computation tasks. As already pointed out, this model assumes a non interruptible, single server type scheduling of computation tasks.

In Fig. 6.16 the SWN model corresponding to the non-preemptive policy for a $2m$ nodes, 1-dimensional (1D) mesh is depicted. The processor identities are modeled as shown in the previous

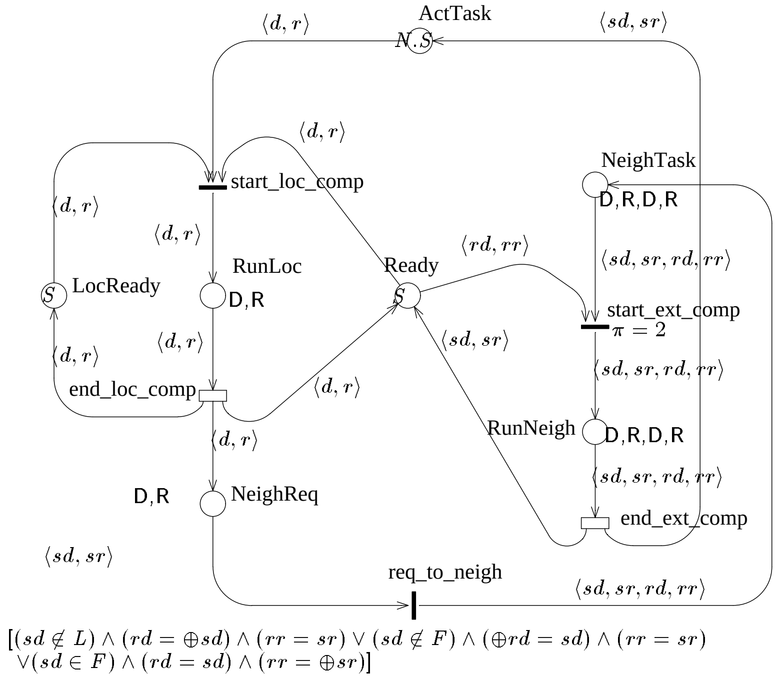


Figure 6.16: $2m$, 1D mesh with non-preemptive policy.

section using a pair of order classes (one without symmetry, and the other one with 2 symmetric elements).

Place ActTask models the presence of N indistinguishable tasks on the different processors, each one characterized by a pair representing the identity of the processor on which it is allocated. Immediate transition start_loc_comp models the scheduling of such tasks one at a time on each processor by synchronizing the tokens in place ActTask with the ones in place LocReady, initially containing only one token per coordinate pair. Place Ready initially containing one token per processor identity represents the availability of processors for the execution of computation and server tasks. Transition end_loc_comp (whose rate is λ) models the activity of processors that are not currently involved in a communication. Its firing represents issuing a communication request to another processor. The selection of the destination processor is made at random through the firing of immediate transition req_to_neigh, that "invents" the identity of the destination by means of the additional components rd and rr in the function labeling its output arc. The values that can be bound to functions rd and rr are restricted by the transition predicate in order to correctly implement the nearest neighbors connection on the 1D mesh topology as illustrated in Figure 6.14. Immediate transition start_ext_comp models the scheduling of the activity related to the message handling at the destination processor. The completion of the activity related to the communication is modeled by the timed transition end_ext_comp (whose rate is μ), The non-preemptive priority policy is simply obtained by giving immediate transition start_ext_comp higher priority than immediate transition start_loc_comp. Indeed once a processors has chosen to start a local computation phase, the token with processor identity is removed from place Ready and it is not available for the enabling of transition start_ext_comp until the local operation completes.

Structural model analysis and validation

By checking the incidence functions of the net and the initial marking, one may easily realize that place LocReady is behaviourally as well as structurally implicit. Indeed its marking may never reduce the enabling degree of its only output transition start_loc_comp since it is always greater than or equal to the marking of the implicant place Ready. It may then be deleted together with its input and output arcs without affecting the behaviour of the model in any way.

After the erasure of the implicit place LocReady the net exhibits two colored, minimal support P-semiflows that cover all places. The fact that all places are covered by P-semiflows is sufficient to guarantee boundedness of the model, while an intelligent use of the two invariant relations is sufficient to prove liveness and reversibility (i.e., the ergodicity of the corresponding Markovian process). The details of the formal proofs are omitted here for the sake of brevity. Notice that all these formal proofs are parametric with respect to the cardinality of the color classes (i.e., are valid $\forall m \geq 1$).

6.4.4 Modeling preemptive interactions

The second model proposed is relative to the preemptive interaction policy. Please remember that in the preemptive interaction policy requests coming from the neighbors are served with higher priority than the local activity, immediately preempting any computation task. However nothing is assumed concerning the scheduling of computation tasks.

The model is not reported here for the sake of brevity. The definition of the basic color class is exactly the same as for the model of the non-preemptive case. The main difference with respect to the SWN depicted in Figure 6.16 is that the arc from place Ready to transition start_loc_comp has been moved to transition end_loc_comp. In this way the token representing the processor is used for the enabling of timed transition end_loc_comp, but is not removed during the execution of computation tasks. Immediate transition start_ext_proc may be enabled by an incoming communication request (if no other communication request is currently being served) even during the execution of a computation task. The fact that start_ext_proc is immediate forces its firing right away, thus possibly disabling end_loc_comp. The memoryless property of the firing time distribution of transition end_loc_comp makes it unnecessary the specification of an age memory policy to correctly handle the resume of activity after preemption. The completion of the activity related to the communication is still modeled by the timed transition end_ext_comp (whose rate is μ). Its firing may resume the enabling of transition end_loc_comp by returning the corresponding processor identity token in place Ready (if no other communication requests are pending).

In this case, place LocReady is not behaviourally redundant with respect to the enabling degree of transition start_loc_comp. However, the latter is an immediate transition, for which the enabling degree is not semantically relevant. The enabling degree of timed transition end_loc_comp that is semantically crucial in the behaviour of the timed model, instead is not affected by the presence of place LocReady, since it is already limited to the maximum value 1 by the marking of place Ready. Therefore, removing place LocReady does not change the timed measurable behaviour of the model.

From the model point of view, after the erasure of place LocReady the net may be further reduced without affecting its qualitative and performance properties by removing immediate transition start_loc_comp and merging places ActTask and RunLoc into a single place.

Boundedness, liveness, and reversibility may be formally proven in a parametric way as in the non-preemptive case. The details of the proofs are omitted here.

6.4.5 Discussion

The formalism provides a direct mapping of the interconnection structure into the model color structure. In case of torus and hypercube topologies the direct mapping into the color structures and functions captures all the symmetries of the model, while in the case of mesh interconnection it does not. Less trivial WN representations may be found that capture system symmetries of mesh interconnection at different levels at the expense of an increasing model complexity. Some conceptually simple extensions of the formalism such as the direct support of the "mirror" type symmetry or the "flip" across color components (permutation of elements in a tuple) would allow the treatment of such more difficult cases in a natural way.

6.5 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 behaviour, 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 behaviour 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.*

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 developed a set of examples of use of the SWN formalism for the description of problems arising in the study of parallel computer architectures. The common denominator of these cases that makes the use of SWNs practically advantageous is their intrinsic high degree of symmetry. In some cases the system symmetry directly matches the basic ones provided by the formalism, while in other cases the basic color and function definitions must be composed in a non trivial way in order to capture the desired system symmetries.

In this sense, the WN formalism can be seen as an intermediate level of abstraction in which some more complex symmetric structures can be reproduced only at the cost of some model complexity. From a practical point of view, it would be better to have the possibility of adding (possibly problem or domain specific) color structures and associated manipulation functions in order to match more closely the modeling needs.

On the other hand, the basic principle of restricting the color complexity so that it is mainly related to the symmetry of the system to be modeled seems to be the key for the success of this modeling approach. Future works on the subject should include the development of a clean method to define a library of ready-to-use symmetry structures together with their appropriate manipulation functions that allow efficient implementation of the symbolic firing technique.

Bibliography

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

[2] H.J. Genrich. Predicate/transition nets. In W. Brawer, W. Reisig, and G. Rozenberg, editors, *Advances on Petri Nets '86 - Part I*, volume 254 of *LNCS*, pages 207–247. Springer Verlag, 1987.

[3] 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.

[4] 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.

[5] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed coloured nets and their symbolic reachability graph. In *Proc.* 11^{th} *International 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.

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

[7] J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Van Nostrand, Princeton, NJ, 1960.

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

[9] 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.

[10] 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.

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

[12] 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} *Int. Conf. on Modeling Techniques and Tools for Performance Analysis*, Sophia Antipolis, France, June 1985. ACM.

[13] 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.

[14] Chuang Lin and Dan C. Marinescu. Stochastic high level Petri nets and applications. *IEEE Transactions on Computers*, 37(7):815–825, July 1988.

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

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

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

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

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

[20] G. Chiola and G. Franceschinis. Colored GSPN models and automatic symmetry detection. In *Proc. 3rd Intern. Workshop on Petri Nets and Performance Models*, Kyoto, Japan, December 1989. IEEE-CS Press.

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

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

[23] 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.

[24] 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.

[25] 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, *Proc.* 6^{th} *Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 391–395, Edinburg, UK, September 1992. Antony Rowe Ltd.

[26] G. Chiola. GreatSPN 1.5 software architecture. In *Proc.* 5^{th} *Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, Torino, Italy, February 1991.

[27] 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).

[28] P.J. Courtois and P. Semal. Computable bounds on conditional steady-state probabilities in large Markov chains and queueing models. *IEEE Journal on Selected Areas in Communication*, SAC-4(6):926–937, Sept. 1986.

[29] P.J. Courtois and P. Semal. Bounds for positive eigenvectors of nonnegative matrices and their approximations. *Journal of the ACM*, 31(4):804–825, Oct. 1984.

[30] W. H. Sanders and J. F. Meyer. Reduced base model construction methods for stochastic activity networks. In *Proc. International Conference on Petri Nets and Performance Models*, pages 74–84, Kyoto,Japan, December 1989.

[31] V. F. Nicola. Lumping in Markov reward processes. In *Proc.* 1^{st} *International Conference on Numerical Solution of Markov Chains.* Marcel Dekker, Inc., January 1990.

[32] G. Chiola and G. Franceschinis. A condition for behaviour-preserving colour simplification in Well-Formed coloured nets. In *Proc. ISCAS'91*, Singapore, June 1991. Special session on Net-based analysis and Design of Concurrent Distributed Event Systems.

[33] 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.

[34] M. Ajmone Marsan, G. Balbo, G. Ciardo, and G. Conte. A software tool for the automatic analysis of generalized stochastic Petri net models. In *Proc. International Conference on Modelling Techniques and Tools for Performance Analysis*, Paris, France, May 1984.

[35] G. Chiola. A software package for the analysis of generalized stochastic Petri net models. In *Proc. Int. Workshop on Timed Petri Nets*, Torino, Italy, July 1985. IEEE-CS Press.

[36] M. Ajmone Marsan and G. Chiola. On Petri nets with deterministic and exponentially distributed firing times. In *Proc.* 7^{th} *European Workshop on Application and Theory of Petri Nets*, pages 151–165, Oxford, England, June 1986. reprinted in G. Rozenberg, ed., Advances on Petri Nets '87, LNCS 266, pp.132–145, Springer Verlag, 1987.

[37] G. Chiola. A graphical Petri net tool for performance analysis. In *Proc.* 3^{rd} *Int. Workshop on Modeling Techniques and Performance Evaluation*, Paris, France, March 1987. AFCET.

[38] G. Chiola. Compiling techniques for the analysis of stochastic Petri nets. In R. Puigjaner and D. Potier, editors, *Proc.* 4^{th} *Int. Conf. Modeling Techniques and Tools for Computer Performance Evaluation*, Palma de Mallorca, Spain, September 1988. Plenum Press, New York.

[39] G. Balbo and G. Chiola. Stochastic Petri net simulation. In *Proc. 1989 Winter Simulation Conference*, Washington D.C., December 1989.

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

[41] G. Balbo, G. Chiola, G. Franceschinis, and G. Molinar Roet. On the efficient construction of the tangible reachability graph of generalized stochastic Petri nets. In *Proc. Int. Workshop on Petri Nets and Performance Models*, Madison, WI, USA, August 1987. IEEE-CS Press.

[42] G. Chiola, J. Campos, J.M. Colom, M. Silva, and C. Anglano. Operational analysis of timed Petri nets and applications to the computation of performance bounds. In *Proc. 5th Intern. Workshop on Petri Nets and Performance Models*, Toulouse, France, October 1993. IEEE-CS Press.

[43] G. Chiola and S. Donatelli. A framework for studying sets of related Petri net models. Technical Report 90/51, Université Paris 6, 4 Place Jussieu, 75252 Paris Cedex 05, France, July 1990. IBP Tech. Report.

[44] G. Chiola and A. Ferscha. Distributed simulation of Petri nets. *Parallel and Distributed Technology*, 1(3):33–50, August 1993.

[45] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In *Proc. 15^{th} International Conference on Application and Theory of Petri Nets*, Zaragoza, Spain, June 1994.

[46] G. Chiola and M. Sereno. An efficient and numerically stable implementation of the randomization formula. submitted for publication, May 1994.

[47] G. Chiola, G. Franceschinis, and R. Gaeta. A symbolic simulation mechanism for well-formed coloured Petri nets. In *Proc. 25th SCS Annual Simulation Symposium*, Orlando, Florida, April 1992.

[48] G. Chiola and A. Ferscha. Self adaptive logical processes: the probabilistic distributed simulation protocol. In *Proc. 27^{th} Annual Simulation Symposium*, La Jolla, California, April 1994.

[49] H.J. Genrich. Equivalence transformations of Pr/T nets. In *Proc. 9^{th} Europ. Workshop on Application and Theory of Petri Nets*, Venezia, Italy, June 1988.

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

[51] C.E. Leiserson. Fat-trees: Universal networks for hardware efficient supercomputing. *IEEE Transactions on Computers*, C-34(10), October 1985.

[52] Kay Hwang. *Advanced Computer Architecture with Parallel Programming.* Mc Graw Hill, 1993.

[53] S. Caselli and G. Conte. GSPN models of concurrent architectures with mesh topology. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 280–289, Melbourne, Australia, December 1991. IEEE-CS Press.