2009 American Control Conference
Hyatt Regency Riverfront, St. Louis, MO, USA
June 10-12, 2009

Efficient State-Based Analysis by Introducing Bags in Petri Nets Color Domains

Serge Haddad, Fabrice Kordon, Laure Petrucci, Jean-François Pradat-Peyre and Nicolas Trèves

Abstract— **The use of high-level nets, such as coloured Petri nets, is very convenient for modelling complex controllable systems in order to have a compact, readable and structured specification. However, when coming to the analysis phase, using too elaborate types becomes a burden.**

A good trade-off between expressiveness and analysis capabilities is then to have only simple types, which is achieved with symmetric nets. These latter nets enjoy the possibility of generating a symbolic reachability graph, which is much smaller than the whole state space and still allows for exhaustive analysis.

In this paper, we extend the symmetric net model with bags on arcs. Hence, variables can be bags of tokens, leading to more flexible models. We show that symmetric nets with bags also allow for applying the symbolic reachability graph technique with application to deadlock detection and more generally for safety properties.

I. Introduction

Managing large specifications is a challenge to tackle industrial size problems. This is particularly true when using Petri Nets (PN). Since having a good expressiveness is important, Coloured Petri Nets (CPN) [1] have been proposed as a high-level model derived from PNs.

The trade-off between expressiveness of the specification formalism and the analysis power (and automation) is a crucial and recurrent problem one must face: the more expressive the specification language, the more difficult the verification.

CPNs [2], [3] provide an excellent expressiveness through an association with the ML programming language to use elaborate functions in arc expressions. As a counterpart, verification can be automated for models relying on complex ML functions, only by generating the state space. Reduction techniques may hence become difficult to apply, or even impossible, thus hampering the verification capabilities for complex systems.

Other extensions, such as algebraic nets [4] or predicate/transition nets [5] also provide more comfortable notations to model complex systems. However, the verification capabilities are also tempered with, for reasons similar to those for CPNs.

S. Haddad is with LSV, ENS Cachan, Cachan, France Serge.Haddad@lsv.ens-cachan.fr

F. Kordon and J-F. Peyre are with LIP6, Université Pierre et Marie Curie, Paris, France Fabrice.Kordon@lip6.fr, Jean-Francois.Pradat-Peyre@lip6.fr

L. Petrucci is with LIPN, Université Paris 13, Villetaneuse, France Laure.Petrucci@lipn.univ-paris13.fr

N. Trèves is with CEDRIC, CNAM, Paris, Nicolas.Treves@cnam.fr

On the contrary, Symmetric Nets[1] [7] only provide a limited set of colour functions. Even though the expressive power is the same as CPNs, practical use is slightly less amenable. Nevertheless, symmetric nets benefit from the definition of the symbolic reachability graph [8], a very condensed way to store the system state space. The use of the symbolic reachability graph allows for analysing very large systems via a model checker.

Furthermore, this graph can be applied in order to perform an efficient control analysis. This can be done in two ways: either the model already represents the controlled system and then safety properties such as deadlock detection can be directly detected by reachability analysis over the graph; or (more interestingly) the model represents the yet uncontrolled system with a partition of states between the environment states and the controller states.

Thus, viewing the graph as a (finite) game between the controller and the environment, the standard algorithms derived from game theory allow for finding a strategy for the controller (or decides that there is none). The objective of this game could be to avoid bad states or more sophisticated ones based on parity, Büchi, Street, ... conditions. Observe that these algorithms are polynomial w.r.t. the size of the model [9] and thus remain tractable with the help of the reduction provided by the symbolic approach.

Our modelling and verification experience on complex systems leads to modelling techniques that still permit a similar expressiveness as in CPNs. For example, discretisation of functions into the initial (and stable) marking of a place have been experimented to represent operations and behaviour of physical systems such as a braking function in a transportation system [10], [11]. However, these techniques may transform an atomic operation into several ones, thus generating complexity in the state space.

The aim of our contribution is to enhance the symmetric nets formalism so as to gain more expressiveness by providing bags manipulation functions. This extension does not sacrifice the underlying symbolic reachability graph and its benefits for model checking and control analysis.

Related work

The construction of a reduced state graph based on symmetries of high level nets was introduced by K. Jensen et al. [12]. However, this technique suffers two drawbacks. On the one hand, the definition of symmetries is left to the

[1]Symmetric Nets were formerly known as Well-Formed Nets, a subclass of high-level Petri nets. The name "Symmetric Nets" has been chosen in the context of the ISO standardisation [6].

978-1-4244-4524-0/09/$25.00 ©2009 AACC

modeller leading to miss some symmetries, and on the other hand the transition firing is still managed as the ordinary one.

In order to combine the advantages of automatic symbolic verification of symmetric nets and of the expressiveness of coloured nets, T. Junttila proposed in [13] a class of nets including a set of constructors for coloured functions still allowing for automatic detection of symmetries. However, this approach is also based on the ordinary firing rule which, in the case of complex operators such as the powerset constructor, leads to an exponential number (w.r.t. the size of the high-level net) of ordinary firings from a single marking whereas in similar cases our technique reduces it to a polynomial number.

Numerous works on different exploitations of symmetries have been developed. Let us cite the main contributions. Symmetries are the support for model checking general temporal logic formulas rather than safety properties (e.g. [14]). Detection of symmetries within ordinary Petri nets is also possible (e.g. [15]). Efficient verification and evaluation procedures are also possible in partially symmetric systems (e.g. [16], [17]).

The paper is structured as follows. Section II formally defines symmetric nets with bags and illustrates their benefits with an example that points out the interest of the formalism for deadlock detection due to the management of different kinds of resources. Then, Section III shows that the symbolic reachability graph technique still applies. Finally, section IV concludes and gives some perspectives to this work.

II. Symmetric nets with bags (SNB)

A. Definitions

Based on the Symmetric Nets (SN) from [7], [13], Symmetric Nets with Bags (SNB) are formally defined. An example is presented in section II-B.

1) Colour domains: In symmetric nets, the colour domains are structured.

- Colour domains are called *classes* and generally represent primitive objects like processes, jobs, files, resources, etc. Classes are finite sets. For some models, it is interesting to define a (total) order between colours of a class. In such a case, a class is said to be *ordered*. In the example of figure 5, there are three classes: `Count`, `Jobs` and `Cores`.
- The colours of a class are objects of the same kind but they may have different behaviours. For instance, a class of jobs may include interactive and batch jobs. In order to represent such differences, a class is partitioned into *static subclasses*. In the example of figure 5, the `Cores` (resp. `Jobs`) class is not partitioned since all cores (resp. jobs) have the same potential behaviour, while the `Count` class is completely partitioned since each different element of this class may explicitly be checked by a transition.
- When modelling, associations between objects are quite usual. For instance, a core executes a thread of a job and then one needs to memorise such an association. So more general colour domains are built by cartesian product of classes. Note that the same class may occur several times in a colour domain (e.g. a network connection between two machines). Also note that the null product corresponds to a domain reduced to a single colour $\{\bullet\}$.

This leads to the following formal definitions.

Definition 1 (Class and subclass): The set of classes of a symmetric net is denoted by $\{C_1, \ldots, C_k\}$. The partition of a class C_i is denoted $C_i = \biguplus_{q \in 1..s_i} C_{i,q}$ where s_i is the number of static subclasses of C_i.

In order to alleviate notations and emphasise the meaning of a class, renamings such as $\texttt{Jobs} \equiv C_1$ are permitted.

We now introduce the Bag notion.

a) Notations: Let C be a set, then a *bag* (or *multiset*) over C is a mapping a from C to $\mathbb{N}$ such that the set (called the *support* of a) $\|a\| = \{c \mid a(c) \neq 0\}$ is finite. Let $a, b \in Bag(C)$. Then $a \cup b$ is defined by $(a \cup b)(c) = a(c) + b(c)$ and $a \geq b$ holds iff $\forall c \in C, a(c) \geq b(c)$. When $a \geq b$, $a \setminus b$ is defined by $(a \setminus b)(c) = a(c) - b(c)$. The size of a, denoted $size(a)$, is defined by $size(a) = \sum_{c \in C} a(c)$. A bag a is denoted by the symbolic expression $\sum_{c \in C} a(c).c$. In this sum, we elide scalars $a(c) = 1$ and terms when $a(c) = 0$ whatever c. For instance, c denotes the bag reduced to the single item c and $\sum_{c \in C} c$ denotes the bag equivalent to the whole set C. Let C and C' be two sets, then $Bag(C) \times Bag(C')$ may be viewed as a subset of $Bag(C \times C')$ by mapping $\langle b, b' \rangle$ onto $\sum_{c \in C, c' \in C'} b(c)b'(c')\langle c, c' \rangle$. This embedding can be generalised to any cartesian product of sets of bags. For instance, $\langle 2c + 3c', 4c'' \rangle \equiv 8\langle c, c'' \rangle + 12\langle c', c'' \rangle$.

Definition 2 (Colour domain): A colour domain is a cartesian product of classes *and sets of bags over classes*. More precisely, a colour domain D can be written $D = \bigotimes_{i \in 1..k} (C_i)^{e_i} \times \bigotimes_{i \in 1..k} Bag(C_i)^{e'_i}$ where e_i is the number of occurrences of class C_i in D and e'_i the number of occurrences of $Bag(C_i)$ in D.

An item d of a colour domain $D = \bigotimes_{i \in 1..k} (C_i)^{e_i} \times \bigotimes_{i \in 1..k} Bag(C_i)^{e'_i}$ will be denoted by $d = \bigotimes_{i \in 1..k, j \in 1..e_i} c_i^j \times \bigotimes_{i \in 1..k, j \in 1..e'_i} b_i^j$ with $c_i^j \in C_i$ and $b_i^j \in Bag(C_i)$.

Most definitions in this section can easily be restricted to Symmetric Nets by leaving out the Bags part.

Note that $D = \bigotimes_{i \in 1..k} (C_i)^{e_i} \times \bigotimes_{i \in 1..k} Bag(C_i)^{e'_i}$ is infinite as soon as some $e'_i \neq 0$. If D is the colour domain of a place, this does not raise any difficulty. Indeed, in a Petri net (resp. a CPN with finite domains) $m(p) \in \mathbb{N}$ (resp. $m(p) \in Bag(C(p))$) and $\mathbb{N}$ (resp. $Bag(C(p))$) is infinite. The key point w.r.t. effectiveness is that a marking must have a finite representation which is also the case for bags of tuples of bags. However if D is the colour domain of a transition, then the firing rule cannot be applied. Thus, with the help of transition guards (see definition 7), we restrict *the colour domain of a transition to be a finite subset of* D.

2) Colour functions: In high-level Petri nets, arcs are labelled by colour functions which select tokens in adjacent places depending on the instantiation performed for the

firing.

The simplest colour functions are the projections, denoted $X^j_{C_i}$, $i \in 1..k$, $j \in 1..e_i$, that select one component of a colour ; the successor functions, denoted $X^j_{C_i}$++, $i \in 1..k$, $j \in 1..e_i$, that select the successor of a component of a colour ; and the "global" selections $C_i.all = \sum_{c \in C_i} c$ that map any colour to the "sum" of colours in class C_i.

New colour functions are defined, that operate both on the tokens and on the "bag" part of the colour domain.

Definition 3 (Basic colour functions): Let C_i be a class and $D = \bigotimes_{i \in 1..k} (C_i)^{e_i} \times \bigotimes_{i \in 1..k} Bag(C_i)^{e'_i}$ a colour domain. Let $d = \bigotimes_{i \in 1..k, j \in 1..e_i} c_i^j \bigotimes_{i \in 1..k, j \in 1..e'_i} b_i^j$. The **basic colour functions** deal with colour domains and are defined from D to $Bag(C_i)$ by :

i: $X^j_{C_i}(d) = c_i^j$ (for all j such that $1 \leq j \leq e_i$);

ii: $X^j_{C_i}(d)$++ = the successor of c_i^j in C_i (C_i is supposed to be ordered and j is such that $1 \leq j \leq e_i$) ;

iii: $C_i.all(d) = \sum_{x \in C_i} x$ and $C_{i,q}.all(d) = \sum_{x \in C_{i,q}} x$.

Let $b_i^j = \sum_{x \in C_i} \alpha_x.x$. Then the **basic colour functions for bags** produce a single element which is a bag and are defined from D to $Bag(C_i)$ by:

B-i: $Y^j_{Bag(C_i)}(d) = b_i^j$ (for all j such that $1 \leq j \leq e'_i$) which denotes the function that *dispatches* items of a bag;

B-ii: $\tilde{}Y^j_{Bag(C_i)}(d) = \sum_{x \in C_i | \alpha_x = 0} 1.x$ (for all j such that $1 \leq j \leq e'_i$) which denotes the function which produces the *complementary* of a bag;

B-iii: $(Y^j_{Bag(C_i)} \cup Y^{j'}_{Bag(C_i)})(d) = \sum_{x \in C_i} (\alpha_x + \alpha'_x).x$ (for all j, j' such that $1 \leq j, j' \leq e'_i$) which denotes the *union* of two bags;

B-iv: $(Y^j_{Bag(C_i)} \setminus Y^{j'}_{Bag(C_i)})(d) = \sum_{x \in C_i} \max(0, (\alpha_x - \alpha'_x)).x$ (for all j, j' such that $1 \leq j, j' \leq e'_i$) which denotes the *difference* between two bags.

Note that the all function (iii) is a constant function. It can thus be viewed as a constant bag and used to define (initial) markings.

Basic colour functions are those of SN. The colour functions ranging over a class are obtained by linear combinations of basic colour functions (note that some constraints are required to ensure that the colour functions select a *positive* number of tokens). Functions labelled by B are those of SNB.

In order to manipulate *bags* we also use the operator $whole$ which, applied to a bag, produces a single bag containing it.

Definition 4 (Whole mapping): Let C be a finite set, $whole_C(c)$ is the mapping from C to $Bag(C)$ defined by: given c in C, $whole_C(c) = 1.\{1.c\} \in Bag(C)$. We extend $whole$ to a mapping from $Bag(C)$ to $Bag(Bag(C))$ by: given $b \in Bag(C)$, $whole_C(b) = 1.b \in Bag(Bag(C))$.

Remark 1: As for SN, when no confusion is possible, the co-domain of colour functions may be omitted ; for instance, the mappings $Y^j_{Bag(C_i)}$ will frequently be denoted Y_i or <Y> as in figure 6, and the mappings $X^j_{C_i}$ will be denoted X^j_i, X_i, <X> or by any name (different from all) as in the model of figure 5 where `j` is used instead of X^1_{Jobs} and k instead of X^2_{Count}. Furthermore, the *whole* colour function will frequently be used in colour functions composition like $whole \circ Y^j_{Bag(C_i)}$, and in this case will be denoted $whole(Y^j_{Bag(C_i)})$.

The effect of these functions is illustrated in the net of figure 1 and one example of firing for transition **t** shown in figure 2. Places **Px** and **Pa** are typed by C while places **Ps** and **Pw** are typed by $Bag(C)$ and thus hold tokens containing a bag. We also provide a comparison between functions <C.all> and <whole(C)>.

Figure 3 also illustrates a simple net using the $\cup$ and $\setminus$ functions on bags. A possible transition firing is shown in Figure 4, after which **p1** is unmarked.

Definition 5 (Class colour functions): Let C_i be a class and $D = \bigotimes_{i \in 1..k} (C_i)^{e_i} \times \bigotimes_{i \in 1..k} Bag(C_i)^{e'_i}$ a colour domain.

A *class colour function* $f : D \longrightarrow Bag(C_i)$ is a linear combination of basic colour functions and colour functions for bags such that $\forall d \in D, \forall c \in C_i, f(d)(c) \geq 0$.

We now define the *tuple* colour functions of a SNB. To do so, we denote $C(x)$, where x is either a transition or a place, the color domain associated with it (see definition 9).

Definition 6 (Tuple colour functions): A colour function labelling an arc between a transition t and a place p is:

i: either a natural number n when $\mathcal{C}(p) = \{\bullet\}$ with $\forall c \in \mathcal{C}(t), n(c) = n.\bullet$;

ii: or a tuple $f \equiv \langle f_1, \ldots, f_k \rangle$, when $\mathcal{C}(p) = C_{\alpha_1} \times \ldots \times C_{\alpha_k}$ where every f_i is a colour function from $\mathcal{C}(t)$ to $Bag(C_{\alpha_i})$. Then $\forall c \in \mathcal{C}(t), f(c) = \langle f_1(c), \ldots, f_k(c) \rangle$

iii: or a tuple $f \equiv \langle f_1, \ldots, f_k, f'_1, \ldots, f'_{k'} \rangle$, when $\mathcal{C}(p) = C_{\alpha_1} \times \ldots \times C_{\alpha_k} \times Bag(C_{\alpha'_1}) \ldots \times Bag(C_{\alpha'_{k'}})$ where every f_i is a class colour function from $\mathcal{C}(t)$ to $Bag(C_{\alpha_i})$ and f'_i is the composition of a class colour function from $\mathcal{C}(t)$ to $Bag(C_{\alpha_i})$ and the $whole_{C_{\alpha'_i}}$ mapping. Let $c \in \mathcal{C}(t)$: $f(c) = \langle f_1(c), \ldots, f_k(c), f'_1(c), \ldots, f'_{k'}(c) \rangle$.

3) Guards: Guards are predicates defined over a colour domain. When applied to a transition, they restrict the corresponding colour domain. They can also be combined with a tuple colour function as follows. Either the instantiating colour fulfils the guard and the new colour function behaves as the tuple function whilst in the other case, the new function returns the empty bag. For instance the colour function $\langle X, Y \rangle$ produces a token with two components but we cannot require that X should be different from Y (see definition 7.i). Similarly, the colour function X selects an item in a class but we cannot require this item to be selected in a given static subclass (see definition 7.iii). We also want to restrict the instantiation of a bag variable to be an ordinary set (see definition 7.B.i) or to constrain the size of the bag instantiation (see definition 7.B.ii).

Guards express such requirements.

Definition 7 (Guards): A (basic) *guard* for bags is a boolean mapping defined on a colour domain $D = \bigotimes_{i=1..k} (C_i)^{e_i} \times \bigotimes_{i=1..k'} Bag(C_i)^{e'_i}$ with $b_i^j = \sum_{c \in C_i} \alpha_c.c$. Let $\bowtie$ be either the $=$ or the $<$ relation. SNB guards are syntactically built with:

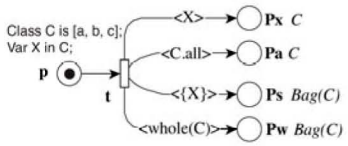

Fig. 1. Model illustrating functions

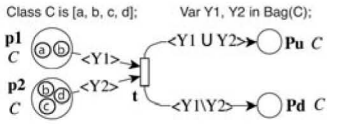

Fig. 3. Model illustrating functions

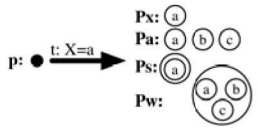

Fig. 2. Result of firing for transition **t** with $X = a$ in the model of figure 1

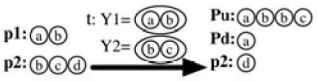

Fig. 4. Result of firing for transition **t** with $Y1 = 1.a + 1.b$ and $Y2 = 1.b + 1.c$ in the model of figure 3

i $[X^{i_1}_{C_i} \bowtie X^{i_2}_{C_i}](c)$ equals `true` iff $c_i^{i_1} \bowtie c_i^{i_2}$;

ii $[X^{i_1}_{C_i} = X^{i_2}_{C_i}{+}{+}](c)$ equals `true` iff $c_i^{i_1}$ is the successor of $c_i^{i_2}$ in C_i;

iii $[X^{i_1}_{C_i} \in C_{i,q}](c)$ equals `true` iff $c_i^{i_1}$ belongs to the static sub-class $C_{i,q}$.

B.i $[Unique(Y^j_{Bag(C_i)})](c)$equals `true` iff $\forall c \in C_i, \alpha_c \leq 1$;

B.ii $[card(Y^j_{Bag(C_i)}) \bowtie n](c)$ equals `true` iff $size(Y^j_{Bag(C_i)}(c)) \bowtie n$;

B.iii $[Y^j_{Bag(C_i)} \bowtie Y^{j'}_{Bag(C_i)}](c)$ equals `true` iff $Y^j_{Bag(C_i)}(c) \bowtie Y^{j'}_{Bag(C_i)}$.

More generally a guard is inductively defined by:

- Let g be a basic guard, then g is a guard;
- Let g_1, g_2 be guards, then $g_1 \vee g_2$, $g_1 \wedge g_2$ and $\neg g_1$ are guards.

Remark 2: When a colour function X_{C_i} has been denoted by another symbol, then this name must be used for the guards.

Definition 8 (General colour function): Let $\{f_i\}_{1\leq i\leq n}$ be a family of tuple colour functions, $\{g_i\}_{1\leq i\leq n}$ a family of guards, and $\{\alpha_i\}_{1\leq i\leq n}$ a family of positive integers.

A *general colour function* $f \equiv \sum_{1\leq i\leq n}[g_i]\alpha_i.f_i$ is defined by

$$f(c) \equiv \sum_{i|g_i(c)=\texttt{true}} \alpha_i.f_i(c)$$

Some abbreviations of colour expressions are useful for modelling such as the *ord* function. Let $D = C_1^{e_1} \times \ldots \times C_k^{e_k}$ and f be a tuple function from D to some $Bag(D)$. Then: $ord(X_i^j).f \equiv \sum_{1\leq q\leq s_i}[X_i^j \in C_{i,q}]q.f$. This function allows the modeller to specify a dynamic multiplicative factor corresponding to the index of the static subclass to which the colour associated with a variable by the instantiation process belongs. Its use will be illustrated in the example of figure 6.

Definition 9 (Symmetric net with Bags): A SNB is a 7-tuple $SNB = \langle P, T, Pre, Post, Cl, \mathcal{C}, \Phi \rangle$ where[2]:

- P is a finite non-empty set of places;
- T is a finite non-empty set of transitions, $T \cap P = \emptyset$;
- $Cl = \{C_1, \ldots, C_k\}$ is the set of classes, each being partitioned into s_i static sub-classes ($C_i = \uplus_{q=1..s_i} C_{i,q}$); we denote $n_i = |C_i|$ and $n_{i,q} = |C_{i,q}|$;
- $\mathcal{C}$ defines for each place and each transition its colour domain, denoted $C(s)$, which is a finite cartesian product of classes and of bags of classes;
- $Post$ (resp. Pre) is the forward (resp. backward) incidence mapping which associates with each pair $(p,t) \in P \times T$ a general colour function for bags defined from $\mathcal{C}(t)$ to $Bag(\mathcal{C}(p))$;
- Φ is a mapping that associates a guard with each transition.

Remark 3: By default, the guard $\Phi(t)$ is the constant `true`.

Definition 10 (Marking): A *marking* of a SNB is a mapping that associates with each place p a bag $m(p) \in Bag(\mathcal{C}(p))$. The initial marking of a net is denoted by m_0.

Definition 11 (Firing rule): Let m be a marking, t a transition and $c_t \in \mathcal{C}(t)$. (t, c_t) is firable at m (denoted $m[(t, c_t)\rangle$) iff:

1) the guard associated with t evaluates to `true` for c_t (i.e. $\Phi(t)(c_t) = \texttt{true}$)
2) $\forall p \in P, m(p) \geq Pre(p,t)(c_t)$.

When $m[(t, c_t)\rangle$, the firing of t *instantiated* by c_t leads to marking m' defined by: $\forall p \in P$, $m'(p) = m(p) - Pre(p,t)(c_t) + Post(p,t)(c_t)$. Given a SNB, $Reach(SNB, m_0)$ denotes the set of all reachable markings from marking m_0.

B. The multi-thread example

The growing market of multi-core processors generates an increased need for the analysis of parallel systems that are much more difficult to design than sequential ones. Since such systems are usually very regular, a formal notation capturing symmetries is of interest because it can cope with larger specifications.

So, we consider the example of a multi-core processor that is based on the following assumptions:

- a job may be multi-threaded;
- a job is assigned to a subset of cores;
- among the cores a master one is associated with the job itself while the other cores are slave ones;

[2]When bags are omitted, the definition holds for SN.

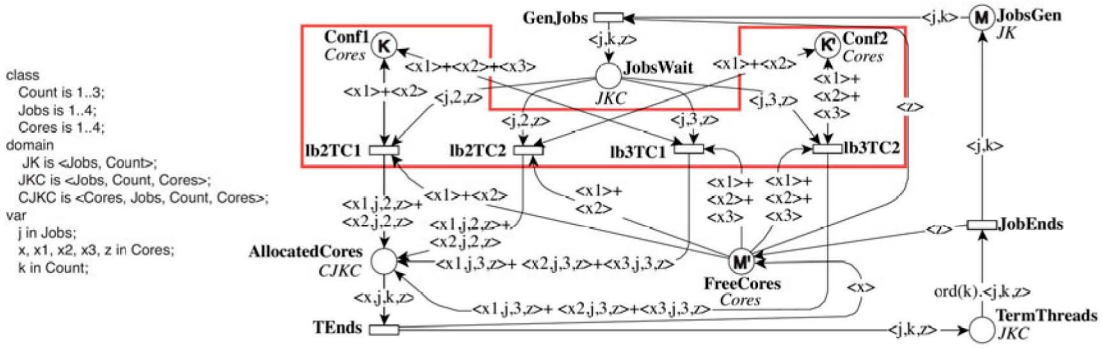

Fig. 5. The multi-thread system modelled with a SN

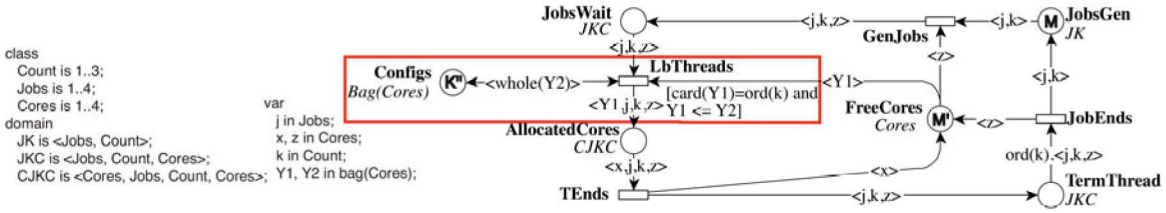

Fig. 6. The multi-thread system modelled with a SNB

- the set of cores is partitioned according to some hardware configuration;
- all threads associated with one job are assigned to slave cores simultaneously;
- every core can manage simultaneously a fixed maximal number of threads (denoted in the sequel by *MaxThreadsPerCore*).

1) The SN model: In figure 5, a job is initially generated by transition *GenJobs*. A triple containing the job number, the number of corresponding threads and the master core is put to wait for being handled in place *JobsWait*. This corresponds to the execution of an initialisation phase. Note that place *JobsGen* contains the different configurations that can occur in the system (marking $\mathbf{M}$ – for example <1,2>+<2,3> means that job 1 has two threads and job 2 has three threads).

Then, depending on the number of threads required, one of the four transitions $lb_k\,TC_i$ assigns k cores with configuration i to the k threads. These cores, having the same configuration as described in *Conf1* and *Conf2* (the initial markings of these places are respectively $\mathbf{K}$ and $\mathbf{K}'$), are removed from the set of *FreeCores* (with initial marking $\mathbf{M'}$= $MaxThreadsPerCore$ × <Cores.all>).

Markings $\mathbf{K}$ and $\mathbf{K}'$ is expressed in a symbolic way: $\mathbf{K} = Z_1 + Z_2$ and $\mathbf{K}' = Z_2 + Z_3$. Z_i are called "dynamic subclasses" (see definition in section III-B) whatever their values. Z_2 is the part that is duplicated in places *Conf1* and *Conf2*, Z_1 the part located in *Conf1* only and Z_3 the part located in *Conf2* only.

The information logged with the *AllocatedCores* is a 4-tuple with the slave core, the job identifier, the number of threads for this job, and the master core. This allows for ensuring a clean termination at a later stage. When a thread execution is finished (transition *TEnds*), the slave core becomes free again. When all threads have finished, a final phase takes place: the terminated threads (place *TermThreads*) are discarded by transition *JobEnds* and the master core is released, terminating the job execution.

One can note that there is a possible deadlock in this model when master threads consume cores that cannot then be allocated to slave threads. In the net, this case corresponds to too many occurrences of *GenJobs* preventing the firing of any $lb_k\,TC_i$ transition.

This example reveals a major drawback of Symmetric Nets, represented by the U-shape in the upper part of figure 5. We must duplicate the transitions in order to capture the consumption of a variable number of tokens. It means that, if we change the *Count* colour domain (denoting the number of threads to associate with a given job), we must adapt the series of transitions $lb_k\,TC_i$. This is not very convenient for modellers. Furthermore, the system cannot be parameterised easily, which is a problem.

2) The SNB model: A new version for the multi-threads example is shown in figure 6. The modified part is framed.

One can note that the numerous transitions $lb_k\,TC_i$ in the net of figure 5 are now expressed using a single transition: *LbThreads*. Configurations are now stored in one place: *Configs* that contains *Bag(Cores)* tokens. Let us note that the marking of *Configs*, $\mathbf{K}''$ is defined from $\mathbf{K}$ and $\mathbf{K}'$ by $\mathbf{K}'' = \{\mathbf{K}\} \cup \{\mathbf{K}'\} = \text{whole}(Z_1 \cup Z_2) \cup \text{whole}(Z_2 \cup Z_3)$.

A bag of cores `Y1` is selected among the free cores, with

the same cardinality as the number of threads to execute (this is specified in the guard of transition *LbThreads* with formula `card(Y1)=ord(k)`). For the transition to be fired, an available configuration where `Y1 <= Y2` (i.e. $\|Y1\| \subseteq \|Y2\|$) must be found (this is expressed by the second term of the guard).

The job ends when all corresponding threads are terminated, and they are removed simultaneously from place *TermThread* as in the Symmetric Net version of the example.

III. Symbolic reachability graph for SNB

The symbolic reachability graph aims at reducing the reachability graph size (thus rendering verification amenable) by regrouping some "equivalent" markings into *symbolic markings* and using a symbolic firing rule compatible w.r.t. the normal firing rule. Thus, many properties of the model, like accessibility, boundness or liveness can directly be checked on the symbolic reachability graph, allowing for the analysis of larger specifications.

A. Symbolic markings

Symbolic markings are based on the notion of *admissible permutations* in the set of colour classes. An admissible permutation is a family $\sigma = \{\sigma_i\}_{i\in I}$ such that a permutation σ_i of C_i fulfils: 1) $\forall C_{i,q},\ \sigma_i(C_{i,q}) = C_{i,q}$ i.e., any item of a static subclass $C_{i,q}$ of C_i is mapped to an item of the same subclass and 2) if C_i is an ordered class then admissible permutations are restricted to rotation (the order of an ordered class cannot be modified).

Given these restrictions, the action of a permutation σ on a colour c of a place p, $c = \bigotimes_{i\in I, j\in 1..e_i(p)} c_i^j \in \mathcal{C}(p)$, is defined by $\sigma(c) = \bigotimes_{i\in I, j\in 1..e_i(p)} \sigma_i(c_i^j)$. We can define the action of σ on a marking m by $\forall p \in P, \forall c \in \mathcal{C}(p), \sigma.m(p)(\sigma(c)) = m(p)(c)$. Note that the enabling rule for a transition is preserved when applying an admissible permutation on a marking and on a transition occurrence:

$$m[(t,c)\rangle m' \iff \sigma.m[(t,\sigma(c))\rangle \sigma.m'$$

Thus, markings obtained with the application of a permutation for a given marking m are "equivalent" in terms of behaviour. Therefore an equivalence class of markings can be defined: $m \sim m' \iff \exists \sigma | \sigma.m = m'$, yielding equivalence classes named *symbolic marking* and denoted $\mathcal{M}$.

The first problem is the representation of a symbolic marking. Describing an equivalence class of a set with its own elements is obviously very expensive in terms of storage and brings no advantage w.r.t. the explicit reachability graph. To tackle this problem, a first approach [12], [13] represents an equivalence class with one of its elements (i.e. a marking). This method reduces the storage requirement for markings but does not provide any saving w.r.t. the state transitions issued from these markings.

An alternative approach [7] consists in a symbolic representation of both the markings (inside an equivalence class) and the transitions issued from these markings. Observe that the number of transitions issued from a marking of a SN may be exponential w.r.t. the size of the SN and thus, the symbolic firing rule is mandatory in order to manage large models.

B. Symbolic marking representation

Let m be an explicit marking. Roughly speaking, we first partition every static subclass $(C_{i,q})$ such that inside the partition, two colours have the same distribution of token components corresponding to the class C_i for m. Then, forgetting the identities of colours inside any partition but memorizing the size of this partition leads to our symbolic marking representation.

More formally, we define for every class C_i a set of dynamic subclasses $\{Z_i^j\}_{1\leq j\leq m_i}$ such that every Z_i^j has two attributes: its cardinality $(card(Z_i^j))$ and the index of the static subclass it belongs to $(d(Z_i^j))$. Given these partitions, the symbolic marking $(mark)$ is represented as an ordinary marking where the dynamic subclasses are substituted to colours. The following definition formalises the characteristics of a symbolic marking representation.

Definition 12 (Symbolic marking representation): A symbolic marking representation of a SNB, $\mathcal{M} = \langle m, card, d, mark\rangle$ is defined by:

- $m : I \mapsto \mathbb{N}^*$ defines the number of dynamic subclasses for every class C_i. $m(i)$ is also denoted m_i and $\widehat{C}_i = \{Z_i^j \mid 0 < j \leq m_i\}$ denotes the set of dynamic subclasses of C_i.
- $card : \bigcup_{i\in I} \widehat{C}_i \mapsto \mathbb{N}^*$ denotes the size of every dynamic subclass.
- $d : \bigcup_{i\in I} \widehat{C}_i \mapsto \mathbb{N}^*$ denotes the index of the corresponding static subclass to which every dynamic subclass belongs. Hence d and $card$ fulfil the following constraints:
  1) $d(Z_i^j) \in \{1, \ldots, s_i\}$ i.e. $d(Z_i^j)$ is the index of a static subclass of C_i.
  2) $\sum_{d(Z_i^j)=q} card(Z_i^j) = n_{i,q}$: the size of a static subclass is the sum of the sizes of the dynamic subclasses that belong to it.
  3) $\forall i \in I, \forall 1 \leq j < j' \leq m_i, d(Z_i^j) \leq d(Z_i^{j'})$: the dynamic subclasses are ordered w.r.t. the order of static subclasses.
- $mark$ associates with every place p a symbolic content: $mark(p) \in Bag(\bigotimes_{i\in I}(\widehat{C}_i)^{e_i(p)} \bigotimes_{i\in I} Bag(\widehat{C}_i)^{e'_i(p)})$. Then, dynamic subclasses act as colours for ordinary markings.

The semantics of a symbolic marking representation is a set of equivalent ordinary markings.

Definition 13 (Symbolic representation semantics): Let $\mathcal{M}$ be a symbolic marking representation. Then the set $[\![\mathcal{M}]\!]$ of associated ordinary markings is defined by $m \in [\![\mathcal{M}]\!]$ iff:

- $\forall i \in I, \exists \alpha_i : C_i \mapsto \widehat{C}_i$; α_i distributes the colours among the dynamic subclasses. As usual, we linearly extend α_i to a mapping from $Bag(C_i)$ to $Bag(\widehat{C}_i)$.
- $\forall Z_i^j \in \widehat{C}_i, |\alpha_i^{-1}(Z_i^j)| = card(Z_i^j)$; these mappings must preserve the size constraints.
- $\forall C_i^j, \alpha_i^{-1}(Z_i^j) \subseteq C_{i,d(Z_i^j)}$; these mappings must preserve the static subclass constraints.

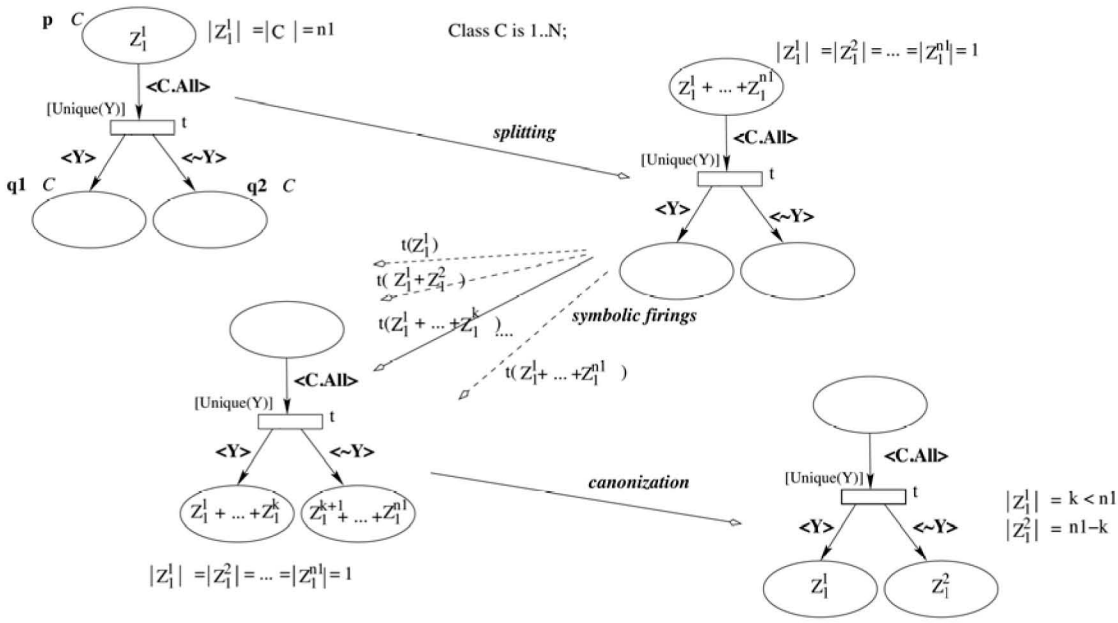

Fig. 7. Example of symbolic firing

- $\forall p \in P, \forall c \in \mathcal{C}(p)$ with $c = \bigotimes_{i \in I, j \in 1..e_i(p)} c_{i,j} \bigotimes_{i \in I, j \in 1..e'_i(p)} b_{i,j}$:

  $m(p)(c) = mark(p)(\bigotimes_{i \in I, j \in 1..e_i(p)} \alpha_i(c_{i,j}) \bigotimes_{i \in I, j \in 1..e'_i(p)} \alpha_i(b_{i,j}))$

  the marking of a place must be preserved by the symbolic transformation.
- When C_i is ordered, $\forall Z_i^j \exists c \in \alpha_i^{-1}(Z_i^j)$ such that $\alpha_i(!c) \in Z_i^{(j \mod m_i)+1}$ and $\forall c' \in \alpha_i^{-1}(Z_i^j), c' \neq c, \alpha_i(c') \in Z_i^j$; the instantiation via α_i of dynamic subclasses must preserve the order of C_i.

It must be emphasized that different representations yield the same set of explicit markings. However, it is possible to define and compute a canonical representation as developed in the appendix [18]. Roughly speaking, a symbolic representation is canonical if the number of dynamic subclasses is minimal and the numbering of dynamic subclasses ensures that the representation is minimal w.r.t. some lexicographic ordering.

C. Symbolic firing rule

The second step in the symbolic reachability graph construction is the design of a symbolic firing rule for symbolic markings. Our goal is to "produce" and "consume" dynamic subclasses instead of colours. A dynamic subclass is selected for each occurrence of a class in the colour domain. However, assume that we instantiate variable $X_{C_i}^j$ with the dynamic subclass Z_i^k. Such an instantiation is sound iff $card(Z_i^k) = 1$ (meaning that this subclass is reduced to a single colour).

Furthermore, the instantiation of $Y_{Bag(C_i)}^j$ should require to select 1) only a subset of colours in some Z_i^k and 2) to select colours from different Z_i^k. Thus in order for the symbolic rule to correspond to the explicit firing rule, we need to preprocess a symbolic representation $\mathcal{M}$. The goal of this preprocessing, called splitting, is to produce $\mathcal{M}'$ such that $[\![\mathcal{M}]\!] = [\![\mathcal{M}']\!]$ and the cardinality of every dynamic subclass of $\mathcal{M}'$ is 1.

Once this splitting has been performed, the transition is fired as in definition 11 with dynamic subclasses instead of colours; this leads to a new symbolic marking. However, this firing includes an optimisation step that reduces the number of possible instantiations.

Rather than formally define the optimisation, we illustrate it on figure 7. It first shows the symbolic marking obtained after the splitting. Applying usual instantiation, variable Y could be associated with any bag $\sum_{j \in J} Z_1^j$ for any $J \subseteq \{1, ..., n_1\}$, n_1 being the maximal cardinality of dynamic subclasses in $Y_{Bag(C_i)}^j$. This would lead to 2^{n_1} different firings.

However, we require that if a dynamic subclass Z_1^j occurs in Y then any $Z_i^{j'}$ with $j' < j$ also occurs in Y. This restriction does not eliminate any associated explicit firing due to the semantics of symbolic markings. Now, the number of different firings is only n_1. This constraint can be generalised to any number of variables occurring in a transition by an arbitrary order over these variables.

In this case, let Z_i^j, $Z_i^{j'}$ obtained by the splitting of the same dynamic subclass with $j' < j$. We require that if Z_i^j occurs in the instantiation of a variable, then $Z_i^{j'}$ occurs either in the instantiation of the same variable or of a previous variable. We emphasize that this reduction is impossible with the approach in [12], [13].

The last step is the canonisation of the representation. The whole process is formally described in [18].

Example: Our approach preserves the use of symbolic states together with bag functions. For example, let us consider the model of figure 5 with M' containing 4 available cores, and M containing 4 configurations (two requiring two additional cores and two requiring three additional cores). If there is at least 3 occurrences of *GenJobs* prior to any occurrence of lb_iTC_j, the system will inevitably become deadlocked in the following configuration: there is no token in either *AllocatedCores* or *FreeCore* — one symbolic state. In the SNB of figure 6, the same configuration is the single symbolic state representing a deadlock.

Thus, the introduction of bags can only reduce the size of the symbolic reachability graph.

Moreover, it may also reduce the number of symbolic firings (those that are expensive in terms of CPU usage). For example, the firing of both *lb2TC1* and *lb2TC2* in the model of figure 5 corresponds to only a single firing of *LbThreads* in the model of figure 6.

IV. Conclusion

In this paper, we have extended the symbolic reachability graph and its related symbolic firing to Symmetric Nets with bags in tokens (SNB) as introduced in [13].

SNB have two main advantages. First, the use of bags in Symmetric nets allows for easier and more readable modelling. The Petri net specification can thus be parameterised without changes in its structure (e.g. adding places or transitions). Hence, the specifier does not have to concentrate on choosing tricks or duplicating large parts of nets. Moreover, these could lead to bad choices that would hamper the analysis capabilities.

Second, it enables the use of the symbolic reachability graph technique, thus allowing for analysing large Petri nets. Our approach maintains a low complexity on the symbolic reachability graph constructions, contrary to previous works like [12], [13].

To achieve this goal, we provide a new consistent set of definitions for SNB. We show on an example in figure 6 the advantages of SNB for a more concise modelling : a single transition corresponds to several similar ones in the SN model of figure 5. Then, we define for SNB the symbolic reachability graph structure and the associated optimised firing rule.

We plan to soon develop within the CPN-AMI Petri net environment (`http://move.lip6.fr/software/CPNAMI/`) [19] both the extended formalism, the adaptation of the symbolic reachability graph and the game-based algorithms for control synthesis.

References

[1] K. Jensen, "Coloured Petri nets and the invariant-method," *Theor. Comput. Sci.*, vol. 14, pp. 317–336, 1981.

[2] M. Beaudouin-Lafon, W. Mackay, M. Jensen, P. Andersen, P. Janecek, H. Lassen, K. Lund, K. Mortensen, S. Munck, A. Ratzer, K. Ravn, S. Christensen, and K. Jensen, "CPN/Tools: A Tool for Editing and Simulating Coloured Petri Nets ETAPS Tool Demonstration Related to TACAS," in *Tools and Algorithms for the Construction and Analysis of Systems, 7th International Conference, TACAS 2001*, L. Springer Verlag, Ed., vol. 2031, 2001, pp. 574–577.

[3] K. Jensen, L. Kristensen, and L. Wells, "Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems," *STTT*, vol. 9, no. 3-4, pp. 213–254, 2007.

[4] W. Reisig, "Petri nets and algebraic specifications." *Theoretical Computer Science*, vol. 80, pp. 1–34, 1991, newsletterInfo: 38,39.

[5] H. Genrich and K. Lautenbach, "System modeling with high-level Petri-nets," in *Theoretical Computer Science*, no. 13, 1981, pp. 103–136.

[6] L. Hillah, F. Kordon, L. Petrucci, and N. Trèves, "PN standardisation : a survey," in *International Conference on Formal Methods for Networked and Distributed Systems (FORTE'06)*, vol. 4229. Paris, France: Springer Verlag, LNCS, September 2006, pp. 307–322.

[7] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad, "Stochastic well-formed colored nets and symmetric modeling applications," *IEEE Transactions on Computers*, vol. 42, no. 11, pp. 1343–1360, 1993. [Online]. Available: citeseer.ist.psu.edu/chiola93stochastic.html

[8] ——, "A symbolic reachability graph for coloured Petri nets," *Theoretical Computer Science*, vol. 176, no. 1–2, pp. 39–65, 1997.

[9] H. Klauck, "Algorithms for parity games," in *Automata, Logics, and Infinite Games*, ser. LNCS, vol. 2500. Springer, 2002, pp. 107–129.

[10] A. de Groot, J. Hooman, F. Kordon, E. Paviot-Adet, I. Vernier-Mounier, M. Lemoine, G. Gaudiere, V. Winter, and D. Kapur, "A survey: Applying formal methods to a software intensive system," in *6th International Symposium on High-Assurance Systems Engineering*. IEEE Computer Society, 2001, pp. 55–64.

[11] F. Bonnefoi, L. Hillah, F. Kordon, and G. Frémont, "An approach to model variations of a scenario: Application to Intelligent Transport Systems," in *Workshop on Modelling of Objects, Components, and Agents (MOCA'06)*, Turku, Finland, June 2006.

[12] P. Huber, A. M. Jensen, L. O. Jepsen, and K. Jensen, "Reachability trees for high-level petri nets." *Theoretical Computer Science, Vol 45*, no. 3, pp. 261–292, 1986, newsletterInfo: 27.

[13] T. Junttila, "On the symmetry reduction method for petri nets and similar formalisms," Ph.D. dissertation, Helsinki University of Technology, Espoo, Finland, 2003.

[14] F. A. Emerson and A. P. Sistla, "Symmetry and model checking," *Formal Methods in System Design: An International Journal*, vol. 9, no. 1/2, pp. 105–131, August 1996. [Online]. Available: citeseer.ist.psu.edu/emerson94symmetry.html

[15] K. Schmidt, "How to calculate symmetries of petri nets," *Acta Inf.*, vol. 36, no. 7, pp. 545–590, 2000.

[16] E. A. Emerson and R. J. Trefler, "From asymmetry to full symmetry: New techniques for symmetry reduction in model checking," in *CHARME*, ser. Lecture Notes in Computer Science, vol. 1703. Springer, 1999, pp. 142–156.

[17] S. Baarir, C. Dutheillet, S. Haddad, and J.-M. Ilié, "On the use of exact lumpability in partially symmetricalwell-formed nets," in *QEST*. IEEE Computer Society, 2005, pp. 23–32.

[18] S. Haddad, F. Kordon, L. Petrucci, J.-F. Pradat-Peyre, and N. Trèves, "Efficient State-Based Analysis by Introducing Bags in Petri Nets Color Domains," LSV, École Normal Supérieure de Cachan, Tech. Rep. RR LSV 09-07, March 2009.

[19] A. Hamez, L. Hillah, F. Kordon, A. Linard, E. Paviot-Adet, X. Renault, and Y. Thierry-Mieg, "New features in CPN-AMI 3 : focusing on the analysis of complex distributed systems," in 6^{th} *International Conference on Application of Concurrency to System Design (ACSD'06)*. Turku, Finland: IEEE Computer Society, June 2006, pp. 273–275.