Symbolic Graph for Petri nets

Isabelle Vernier
Laboratoire IBP-MASI
Université Paris 6
4, place Jussieu, 75252 Paris - Cedex 05, France
email : isabelle.vernier@masi.ibp.fr
tel : (33 1) 44 27 71 04, fax : (33 1) 44 27 62 86

Abstract: In this paper we consider programs with one parameter, the number of processes, represented by Petri nets. A marked Petri net models a program with a fixed number of processes, i.e., an instantiated program. We give an algorithm to build a symbolic graph that represents the reachability graphs of a Petri net with an infinite number of initial markings. There is an initial marking for each value of the parameter. We introduce symbolic markings that define sets of markings of instantiated programs. We prove that the symbolic graph represents exactly the reachable markings of all the instantiated programs. Each possible execution of an instantiated program is represented in the symbolic graph. We show how to verify properties on the symbolic graph.

I. Introduction

The study of behavioral properties of parallel programs needs, in many cases, the building of its state graph. This building is usually too complex. Even if the model of the program has a reasonable size, the size of the state graph prevents one from exploiting it.

We consider programs modeled by Petri nets [17]. To obtain a reduced reachability graph, two ways are possible. A first approach is to reduce the net [3], [11], [13] and then to build the reachability graph. A second approach is to build a reduced graph. Only the properties preserved by the reductions can be verified on the graph. Several studies have concerned the reduction of the size of the reachability graph. In [9] the modification of markings greater than one of their ancestors allows one to avoid the building of infinite graphs and to minimize the size of the obtained coverability graph. In [21] sequences of transitions that only differ in the order of the transitions are identified. In [4] and [12] the size of the graph is reduced by the computation of a unique representative for each equivalence class of markings. In [22], the building of the reachability graph is modular. To apply these methods, the value of each parameter of the program, resources and processes, must be fixed. Therefore, to study a program with different values of the parameters new methods are necessary.

Previous works study methods to verify properties of programs without fixing value of the parameters. The main result is the undecidability of the properties verification [1], [20]. Therefore, all the proposed methods have some limitations. The first one is that the programs have a unique parameter, the number of processes. All the processes have an identical behavior independent of their identity. The method we propose applies to this subset of programs.

The methods to verify properties of parallel programs are not totally automatic [5], [16], [27]. In these three methods, the verification of the properties of the instantiated programs is deduced from the verification of the properties of a representative program. The problem is to find this program. Some extensions of the methods of Kurshan and Wolper [18], [2] give algorithms to find the representative program. These algorithms are semi-decidable. The method presented in [10] allows one to verify properties that only refer to the behavior of a unique process. The solution proposed in [19] has some failure cases.

A common feature of all these methods is that the verification is achieved on a structure that is equivalent to the behavior of the instantiated programs regarding a set of properties. This set is infinite for some methods. The verification structure does not allow one the solve the reachability problem since the reachable states of the instantiated programs are not represented. It does not directly give a better understanding of the behavior of the instantiated programs since it does not represent their behavior.

The algorithm we propose builds a symbolic graph. It applies on a subclass of programs and has some failure cases. They appear when we cannot represent in a finite graph the behaviors and the reachable states of all the instantiated programs. Our symbolic graph represents the states and the behaviors of all the instantiated programs. Once this graph is build, it is possible to solve the reachability problem and to have a better understanding of the behaviors of the instantiated programs.

Section 2 presents the program we study and the restriction of the Petri nets we consider. Section 3 defines the symbolic markings that are nodes of the symbolic graph and that represent sets of states of the instantiated programs. Section 4 gives the firing rule defined from the symbolic markings. Section 5 presents the algorithm to compute the symbolic reachability tree from which the symbolic graph is deduced. We prove that the symbolic graph represents the reachable states and the behaviors of all the instantiated programs. Section 6 presents some properties whose verification is possible on the symbolic graph. In section 7 we conclude and present future extensions of our work.

II. Parallel Programs and Petri nets

We build a symbolic graph with the view to use it to verify behavioral properties of the instantiated programs. As it is a too complex problem we have developed our building algorithm on a subclass of parallel programs. Once the basis well defined, it is possible to consider extensions of the subclass. The model we have chosen to represent the parallel programs is the Petri net model.

II.1. Petri nets

A Petri net is a bipartite graph. The vertices are the places and the transitions. The places, the transitions and the tokens represent respectively the states, the actions and the processes or variables. The state of a process is given by the place that contains its token. The marking of a place is the number of tokens it contains. The initial marking of a Petri net associates with each place the number of processes or resources it initially contains. A Petri net with an initial marking is a marked Petri net.

<u>Definition 1</u> Marked Petri net

A Petri net $PN = <P, T, Pre, Post, m_0>$ is a Petri net where:

- P is a set of places, T is a set of transitions, $T \cap P = \emptyset$, $T \cup P \neq \emptyset$,
- Pre is an application from $P \times T$ to $\mathbb{N}$, that represents the weight of the arcs linking the places to the transitions,
- Post is an application from $P \times T$ to $\mathbb{N}$ that represents the weight of the arcs linking the transitions to the places,
- m_0 is an application from P to $\mathbb{N}$ that represents the initial marking of the Petri net.

Graphically, circles and rectangles represent respectively the places and the transitions. The weight of an arc is represented only if it is different from one. The arcs with a weight equal to zero are not represented.

The places p such that Pre(p,t) (resp. Post(p,t)) differs from zero are the input (resp. output) places of transition t.

A transition t is enabled from a marking if and only if the number of tokens in each place p is greater than the value Pre(p,t).

<u>Definition 2</u> Enabled transition

Let m be a marking of a Petri net and t a transition, t is enabled from m, m[t> iff:

$\forall p \in P,\ m(p) \geq Pre(p,t)$.

The firing rule defines the way to compute the markings reachable from another. The principle is that the application Pre (resp. Post) defines for each place and for each transition the number of tokens that are consumed (produced) while the firing of the transition.

Definition 3 Firing rule

Let m be a marking of a Petri net and t a transition, t is enabled from m and the reached marking m' is computed as follows, m[t>m':

$\forall p \in P$, $m'(p) = m(p) - Pre(p,t) + Post(p,t)$.

To compute the set of reachable markings of a marked Petri net the firing rule is applied to the initial marking and to the new reached markings.

A state machine is a Petri net whose transitions have exactly one input place and one output place. All the arcs are weighted by one. Therefore a transition represents an action executed by one process. The input (resp. output) place is the state of the process before (resp. after) the firing.

An extended state machine is a state machine that allows one to model rendez-vous between processes.

Definition 4 Extended state machine

An extended state machine SM is a state machine $<P, T, Pre, Post, m_0>$ where:

- $\forall t \in T, \sum_{p \in P} Pre(p,t) = \sum_{p \in P} Post(p,t)$.

Remark

Each possible marking m of an extended state machine is such that $\sum_{p \in P} m(p) = \sum_{p \in P} m_0(p)$.

The notations m[t> and m[t>m' are extended to sequences of transitions:

- m[s> means that sequence s is enabled from m,

- m[s>m' means that the firing of s from m leads to m'.

II.2. Parallel Programs

We consider the same class of programs as in the methods of Clarke, Kurshan, Wolper and German. Two parts compose the programs: *n* processes that have the same behavior and a unique resources manager. The behavior of the identical processes is independent of their identity.

The processes communicate through shared variables, resources, whose accesses are supervised by the resources manager. Another communication possibility is a rendez-vous between processes.

An extended state machine represents the *n* processes. A Petri net represents the resources manager and the resources. The transitions that model accesses to the resources are common to the two Petri nets. They are synchronization transitions.

Notations

Consider two Petri nets $PN = <P, T, Pre, Post, m_0>$ and $PN_1 = <P_1, T_1, Pre^*, Post^*, m_0^*>$ such that $P_1 \subseteq P$ and $T_1 \subseteq T$ then

- Pre^* and $Post^*$ are applications from $P_1 \times T_1$ to $\mathbb{N}$,

- $\forall p \in P_1, \forall t \in T_1, Pre^*(p,t) = Pre(p,t)$ and $Post^*(p,t) = Post(p,t)$,

- m_0^* is an application from P_1 to $\mathbb{N}$ and $\forall p \in P_1, m_0^*(p) = m_0(p)$.

Definition 5 Model of Parallel Programs, M2P

A M2P is a Petri net $P = <P_p \cup P_c, T_p \cup T_c \cup T_{synch}, Pre, Post, m_0>$ where:

- $P_p \cap P_c = \emptyset$, $T_p \cap T_c \cap T_{synch} = \emptyset$,

- $<P_p, T_p \cup T_{synch}, Pre^*, Post^*, m_0^*>$ is a marked extended state machine,

- $<P_c, T_c \cup T_{synch}, Pre^*, Post^*, m_0^*>$ is a marked Petri net.

The model with an initial marking represents an instantiated program. In the following, a marking with no other information is a marking of a marked Petri net.

Example 1

We consider the problem of n processes that perform an initialization stage and execute internal actions. The initialization needs the exclusive access to resource A. Once a process is initialized it can cooperate with other processes to avoid them the initialization step.

Figure 1 represents the Petri net $P = \langle P_p \cup P_c, T_p \cup T_c \cup T_{synch}, Pre, Post, m_0 \rangle$ that models this problem:

P_p = {Idle, Initialization, Initialized}, P_c = {A},

T_p = {Cooperation, Internal_Action}, T_{synch} = {Access_A, Free_A}, $T_c = \varnothing$,

m_0(Idle) = n, m_0(A) = 1, m_0(Initialization) = m_0(Initialized) = 0.

The white places and transitions represent the states and the actions of the n identical processes. The gray place is the unique place of the control Petri net. The black transitions are the synchronization transitions.

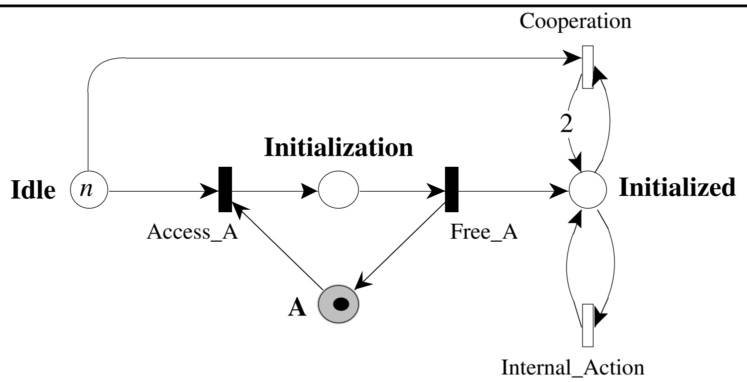

Figure 1

Initially all the processes are in place Idle. They can access to the resource A if another process does not use it. If a process is already in place Initialized they can cooperate with it to obtain the initialized status. A process that has accessed to resource A owns it until the initialization stage ends and then it releases the resource to become an initialized process. The Cooperation transition represents a rendez-vous between two processes.

III. Symbolic marking

Markings associate with each place an integer that represents the number of tokens, processes or resources, it contains. They are not suitable to consider Petri nets whose initial marking is not entirely known, the number of processes is finite but not defined. To this purpose we define symbolic markings. A symbolic marking represents a set of markings. We first define the general symbolic markings whose definition is independent of the characteristics of the model M2P.

III.1. General Symbolic marking

The key point of the representation of a general symbolic marking is to associate an integer and a relational operator "=" or "≥" with each place. The intended meaning is:

- if "= x" is associated with a place, then exactly x processes are in this place,

- if "≥ x" is associated with a place, then at least x processes are in this place.

Definition 6 General Symbolic marking

Let gsm = <OP,M> be a symbolic marking:

- OP is defined from $P_p \cup P_c$ to {"=", "≥"},

- M is defined from $P_p \cup P_c$ to $\mathbb{N}$.

If gsm is a general symbolic marking then functions gsm.OP and gsm.M define the operator and integer attributes.

A marking of an instantiated Petri net is a special general symbolic marking. The operator "=" is associated with all the places.

Notation

{p gsm.OP(p) gsm.M(p) / $p \in P_p \cup P_c$} denotes a general symbolic marking gsm where the places such that M(p) = 0 and OP(p) = "=" are omitted.

III.2. General symbolic markings and markings

A general symbolic marking represents an infinite set of markings of instantiated Petri net. We denote set[gsm] the set of markings represented by gsm.

Definition 7 Set of markings

A general symbolic marking, gsm, represents a marking if an only if the latter fulfills the relations expressed by the functions gsm.OP and gsm.M. A general symbolic marking gsm contains all the markings m, $m \in$ set[gsm], such that $\forall p \in P_p \cup P_c$, m(p) = gsm.M(p) if gsm.OP(p) = "=" otherwise m(p) $\geq$ gsm.M(p).

Example 2

Let gsm = {Idle $\geq$ 1, Initialized $\geq$ 1, A = 1} be a general symbolic marking. All the markings m such that m(Idle) $\geq$ 1, m(Initialization) = 0, m(Initialized) $\geq$ 1 and m(A) = 1 belongs to set[gsm].

A transition is enabled from a general symbolic marking if and only if it is enabled from one of the markings it represents.

Property 1

Let gsm be a general symbolic marking and t be a transition. t is enabled from gsm iff:

$\forall p \in P_p \cup P_c$, gsm.M(p) $\geq$ Pre(p,t) or gsm.OP(p) = "$\geq$".

Proof

- If $\exists p \in P_p \cup P_c$, gsm.M(p) < Pre(p,t) and gsm.OP(p) = "=" then $\forall m \in$ set[gsm], m(p) < Pre(p,t), m does not enable t.

- If $\forall p \in P_p \cup P_c$, gsm.M(p) $\geq$ Pre(p,t) or gsm.OP(p) = "$\geq$" we build a marking $m \in$ set[gsm] such that m[t>. If gsm.M(p) $\geq$ Pre(p,t) then m(p) = gsm.M(p) else m(p) = pre(p,t).

III.3. Relations and operations on general symbolic markings

To compute the symbolic reachability tree, it is essential to be able to compare two general symbolic markings. These comparisons allow us to avoid the building of an infinite tree. The two relations we define are partial relations. They both apply to the sets of markings represented by the general symbolic markings.

Definition 8 Inclusion

Let gsm and gsm' be two general symbolic markings, gsm is included in gsm', gsm $\subseteq$ gsm', if and only if set[gsm] $\subseteq$ set[gsm'].

Definition 9 Inferiority

Let gsm and gsm' be two general symbolic markings, gsm is less or equal to gsm', gsm $\leq$ gsm', if and only if for each marking represented by gsm' there is a less marking represented by gsm and for each marking represented by gsm there is a greater marking represented by gsm'.

Properties 2

We consider "$\geq$" > "=".

a) gsm $\subseteq$ gsm' iff

$\forall p \in P_p \cup P_c$, if gsm'.OP(p) ="=" then gsm.OP(p) = "=" and gsm.M(p) = gsm'.M(p)
else gsm'.M(p) $\leq$ gsm.M(p).

b) gsm $\leq$ gsm' iff

$\forall p \in P_p \cup P_c$, gsm.OP(p) $\leq$ gsm'.OP(p) and gsm.M(p) $\leq$ gsm'.M(p)

Proof

a) $H = \forall p \in P_p \cup P_c$,

if gsm'.OP(p) ="=" then gsm.OP(p) = "=" and gsm.M(p) = gsm'.M(p) else gsm'.M(p) ≤ gsm.M(p).

- H is verified.

Let $m \in$ set[gsm] then $\forall p \in P_p \cup P_c$

if gsm.OP(p) = "=" and gsm'.OP(p) = "=" then m(p) = gsm'.M(p),

if gsm.OP(p) = "=" and gsm'.OP(p) = "≥" then m(p) ≥ gsm'.M(p),

if gsm.OP(p) = "≥" then gsm'.OP(p) = "≥" and m(p) ≥ gsm'.M(p).

If H then set[gsm] $\subseteq$ set[gsm'] and gsm $\subseteq$ gsm'.

- H is not verified.

if gsm'.OP(p) = "=" and gsm.OP(p) = "≥" then $\exists m \in$ set[gsm], m(p) > gsm'.M(p), $m \notin$ set[gsm'],

if gsm'.OP(p) = "=", gsm.OP(p) = "=" and gsm.M(p) ≠ gsm'.M(p) then $\forall m \in$ set[gsm], m(p) > gsm'.M(p), $m \notin$ set[gsm'],

if gsm'.OP(p) = "≥" and gsm.M(p) < gsm'.M(p) then $\exists m \in$ set[gsm], m(p) < gsm'.M(p), $m \notin$ set[gsm'].

If not H then set[gsm] $\not\subseteq$ set[gsm'] and gsm $\not\subseteq$ gsm'.

b) $H = \forall p \in P_p \cup P_c$, gsm.OP(p) ≤ gsm'.OP(p) and gsm.M(p) ≤ gsm'.M(p)

- H is verified.

Let m_{min}, $\forall p \in P_p \cup P_c$ $m_{min}(p)$ = gsm.M(p), $m_{min} \in$ set[gsm] and $\forall m' \in$ set[gsm'], $m_{min} \le m'$,

Let $m \in$ set[gsm], we build $m' \in$ set[gsm'] such that $m' \ge m$

$\forall p \in P_p \cup P_c$ if gsm.OP(p) = "=" then m'(p) = gsm'.M(p) else m'(p) = MAX(gsm'(p), m(p)+1).

If H then gsm ≤ gsm'.

- H is not verified.

If $\exists p \in P_p \cup P_c$, gsm.M(p) > gsm'.M(p) then $\exists m' \in$ set[gsm'], m'(p) = gsm'.M(p) and $\forall m \in$ set[gsm], m > m',

if $\exists p \in P_p \cup P_c$, gsm.OP(p) > gsm'.OP(p) then gsm.OP(p) = "≥", gsm'.OP(p) = "=", $\exists m \in$ set[gsm], m(p) = MAX(gsm.M(p), gsm'.M(p)+1) and $\forall m' \in$ set[gsm'], $m' \not\ge m$.

If not H then gsm $\not\le$ gsm'.

While the verification of properties, we will have to compute the intersection of two general symbolic markings. As for the previous relations, this operation is concerned with the sets of markings represented by the general symbolic ones.

Definition 10 Intersection between general symbolic markings

Let gsm and gsm' be two general symbolic markings, gsm $\cap$ gsm' represents the set of markings that belongs to set[gsm] $\cap$ set[gsm'].

Property 3

Let gsm_1 and gsm_2 be two general symbolic markings, $gsm = gsm_1 \cap gsm_2$ with:

$\forall p \in P_p \cup P_c$,

- if gsm_1.OP(p) = gsm_2.OP(p) = "=" then

  if gsm_1.M(p) = gsm_2.M(p) then gsm.OP(p) = "=" and gsm.M(p) = gsm_1.M(p)

  else set[gsm] = Ø,

- if gsm_i.OP(p) = "=" and gsm_j.OP(p) = "≥", $i \neq j$, $i \in \{1,2\}$, $j \in \{1,2\}$, then

  if gsm_i.M(p) ≥ gsm_j.M(p) then gsm.OP(p) = "=" and gsm.M(p) = gsm_i.M(p)

  else set[gsm] = Ø,

- if gsm_1.OP(p) = gsm_2.OP(p) = "≥" then

  gsm.OP(p) = "≥" and gsm.M(p) = MAX(gsm_1.M(p), gsm_2.M(p)).

Proof

The three following properties follow from the computation of gsm:

- $set[gsm] = Ø$ iff $set[gsm_1] \cap set[gsm_2] = Ø$,
- $m \in set[gsm]$ implies $m \in set[gsm_1] \cap set[gsm_2]$,
- $m \in set[gsm_1] \cap set[gsm_2]$ implies $m \in set[gsm]$.

III.4. Symbolic Marking and Model for parallel programs

The general symbolic markings allow us to represent sets of markings but do not take into account the particularities of the model M2P. We define a restriction of the general symbolic markings. The same transitions are enabled from all the markings that a symbolic marking represents. All the properties and relations defined for the general symbolic markings are valid for the symbolic markings.

We first define a function Limit that associates with each place the maximum number of tokens necessary for the firing of all the transitions.

Definition 11 Limit function

$\forall p \in P_p \cup P_c$, $Limit(p) = MAX(Pre(p,t))$, $t \in T_p \cup T_c \cup T_{synch}$.

If we do not know the exact number of tokens in a place p the integer associated with this place must be greater of equal to Limit(p). This condition ensures that the transitions enabled from a symbolic marking are enabled from all the markings it contains.

We consider programs with a unique parameter, the number of processes. Therefore, the possible number of tokens in the places of the control net is independent of the number of processes. For each reached symbolic marking, the operator associated with the places of this Petri net is "=".

Definition 12 Symbolic marking

A symbolic marking sm is a general symbolic marking with:

- $\forall p \in P_c$, sm.OP(p) = "=".
- $\forall p \in P_p$, if $sm.M(p) < Limit(p)$ then sm.OP(p) = "=",

Property 4

Let sm be a symbolic marking and t be a transition: sm[t> iff $\forall m \in set[sm]$, m[t>.

Proof

- If $\forall m \in sm$, m[t> then sm[t> (property 1).
- If sm[t>, then $\exists m \in sm$, m[t>. $\forall p \in P_p \cup P_c$, $m(p) \geq Pre(p,t)$.

$\forall p \in P_p \cup P_c$, sm.OP(p) = "=", $\forall m' \in sm$, $m'(p) = sm.M(p) = m(p)$ then $m'(p) \geq Pre(p,t)$,

sm.OP(p) = "≥", $sm.M(p) \geq Limit(p) \geq Pre(p,t)$ then $\forall m' \in sm$, $m'(p) \geq Pre(p,t)$.

$\forall m \in sm$, m[t>.

As the operator "=" is associated with all the places of the control Petri net, a symbolic marking represents markings of an instantiated program only if the number of processes is greater than a bound. This bound is equal to the sum of the integers associated with the places of the extended state machine.

Definition 13 bound associated with a symbolic marking

Let sm be a symbolic marking then $B(sm) = \sum_{p \in P_p} sm.M(p)$

Property 5

Let P be a program instantiated with n, M the set of markings of P (not necessarily reachable) and sm a symbolic marking

$set[sm] \cap M \neq Ø$ if and only if $n \geq B(sm)$.

Proof

The proof is trivial if we consider that if $m \in$ set[sm] then $\sum_{p \in P_p} m(p) \geq \sum_{p \in P_p} sm.M(p)$.

Example 3 Symbolic markings

The initial symbolic marking of the model of Figure 1 is: {Idle ≥ 1, A = 1}. It represents the initial markings of the instantiated Petri nets with the resource A free and all the tokens that represent identical processes in place Idle. The bound associated with this symbolic marking is 1.

The symbolic marking {Idle ≥ 1, Initialized ≥ 1, A = 1} represents the markings with at least one token in state Idle, at least one token in state Initialized and the resource A free. The bound associated with this symbolic marking is 2. This symbolic marking cannot represent a marking with only one token in one of the two places Idle and Initialized.

To the purpose of the firing rule we define the splitting up of a general symbolic marking into disjoint symbolic markings.

Definition 14 Splitting up of general symbolic markings in relation to a place

Let gsm be a general symbolic marking and p a place such that gsm.M(p) < Limit(p) and gsm.OP(p) = "≥". gsm is not a symbolic marking. gsm is split up into $gsm_0, \ldots, gsm_n$ (n = Limit(p)-gsm.M(p)) where:

$\forall p' \neq p, \forall i, 0 \leq i \leq n, gsm_i.M(p') = gsm.M(p')$ and $gsm_i.OP(p') = gsm.OP(p')$

$\forall i, 0 \leq i < n$

$gsm_i.M(p) = gsm.M(p) + i$ and $gsm_i.OP(p)$ = "="

$gsm_n.M(p) = Limit(p)$, $gsm_n.OP(p)$ = "≥".

Property 6

Let gsm be a general symbolic marking and $gsm_0, \ldots, gsm_n$ the splitting up of gsm in relation to p.

a) $\forall i, j, i \neq j$, set[gsm_i] $\cap$ set[gsm_j] = Ø,

b) $\bigcup_{i=0}^{n}$ set[gsm_i] represents the same set of markings than gsm.

Proof

a) The proof follows from the fact that the conditions on the number of tokens in place p, in the gsm_i, are disjoint.

b) The proof follows from the fact that the union of the conditions on the number of tokens in place p in the gsm_i is equal to the condition in gsm. The conditions on the other places are the same in gsm and in each gsm_i.

To obtain the splitting up of a general symbolic marking into symbolic markings we apply the splitting up procedure until the attributes associated with all places p are such that if the operator is "≥" the integer is greater or equal to Limit(p). The property 6 is extended. The obtained symbolic markings are such that they are disjoint and their union represents the same set of markings as the initial general symbolic marking.

Example 4 Splitting up of a general symbolic marking

Let gsm = {p ≥ 1, q ≥ 0, r = 2} be a general symbolic marking. Limit(p) = 2 and Limit(q) = 1, then gsm is not a symbolic marking. We first split up gsm in relation to p. We obtain the two following general symbolic markings:

gsm_0 = {p = 1, q ≥ 0, r = 2}, gsm_1 = {p ≥ 2, q ≥ 0, r = 2}.

We now split up each gsm_i in relation to q. We obtain the four following symbolic markings:

sm_{00} = {p = 1, q = 0, r = 2}, sm_{01} = {p = 1, q ≥ 1, r = 2},

sm_{10} = {p ≥ 2, q = 0, r = 2}, sm_{11} = {p ≥ 2, q ≥ 1, r = 2}.

As we have obtained symbolic markings, we stop here the splitting up.

IV. Symbolic firing rule

Consider sm a symbolic marking and t a transition enabled from sm. The problem is to compute the set of markings reachable from markings represented by sm after the firing of t. The basic rule is to apply the firing rule of Petri nets to the symbolic marking.

The application of the firing rule, defined for Petri nets, to the integers associated with the places gives a general symbolic marking. This general symbolic marking represents the set of the reachable markings.

Definition 15 Set of reachable markings

Let sm be a symbolic marking and t a transition, t is enabled from sm and the reached set of markings, gsm', is computed as follows:

$$\forall p \in P_p \cup P_c,\ gsm'.M(p) = sm.M(p) - Pre(p,t) + Post(p,t),\ gsm'.OP(p) = sm.OP(p),$$

It is trivial to see that gsm' represents the set of markings reachable from sm after the firing of t. It is possible that for some place p, gsm'.M(p) < Limit(p) and gsm'.OP(p) = "≥", gsm' is not a symbolic marking. Therefore we decompose gsm' into several disjoint symbolic markings, $sm'_0 \dots sm'_n$.

Property 7

Let sm[t>gsm', gsm' is split up into $sm'_0, \dots, sm'_n$ then

$$\bigcup_{i=0}^{n} set[sm'_i]$$ represents the set of markings reachable from sm after the firing of t.

Proof

We use the fact that set[gsm'] represents the set of markings reachable from sm after the firing of t and property 6.

Each sm'_i is reachable from a subset of the set of markings represented by sm. Conditions on the places in relation to which the decomposition of gsm' has been computed define this subset. If $sm'_i.OP(p)$ = "=", the markings m that lead to a marking of $set[sm'_i]$ are such that $m(p) = sm'_i.M(p) + Pre(p,t) - Post(p,t)$. If $sm'_i.OP(p)$ = "≥", the markings m that lead to a marking of $set[sm'_i]$ are such that $m(p) \geq sm'_i.M(p) + Pre(p,t) - Post(p,t)$.

If the firing of t from sm leads to several symbolic markings, the bound associated with some of them may be greater than the bound associated with sm. Two symbolic markings may represent markings of programs instantiated with a number of processes that verify different conditions. If a symbolic marking is a marking, it represents one marking of a single instantiated program.

Example 5 symbolic firing rule

Let sm = {Idle ≥ 1, A = 1} be the initial symbolic marking of the model of Figure 1. The transition Initialization is enabled from sm. The reached general symbolic marking is gsm = {Idle ≥ 0, Initialization = 1, A = 1}. It is not a symbolic marking since Limit(Idle) = 1. The decomposition of gsm in relation to Idle gives the two following symbolic markings:

sm'_0 = {Idle = 0, Initialization = 1, A = 1} and sm'_1 = {Idle ≥ 1, initialization = 1, A = 1}.

The markings represented by sm'_0 are reachable from markings m represented by sm with m(Idle) = 1. The markings represented by sm'_1 are reachable from markings m represented by sm with m(Idle) ≥ 2.

As sm'_0 is a marking, it represents a state of the program instantiated with one process. The symbolic marking sm'_1 represents states of the programs instantiated with at least two processes.

V. Symbolic graph

To build the symbolic graph we first built a symbolic tree. Then we identify with a symbolic marking of the tree all the nodes it contains. To compute the tree, we start from the initial symbolic marking and we apply the symbolic firing rule defined in the previous section.

However the tree would generally be infinite. We have defined two new rules to avoid the possibility of infinite development of the tree.

The first rule indicates when to stop the computation of a branch of the tree. If a symbolic marking is included in one of its ancestors, the computation of its successors will give no new information. We stop the building of this branch.

The second rule allows us to avoid the computation of potentially infinite branches of the tree. The principle we use is similar to those presented in [15] and [9]. We modify a symbolic marking that is strictly greater than one of its ancestors.

The principle of the methods of Karp and Finkel is to detect, for each reachable marking, the places that can contain an infinite number of tokens. The information that is kept gives no condition on the number of tokens (even, odd, related to the number of tokens in another place, …). We are confronted with the same limits. Since we want to keep enough information to solve the reachability problem we cannot apply the two previous methods. The drawback of this requirement is that it is sometimes not possible to represent with a finite number of symbolic markings the set of the reachable markings of the instantiated programs. Our algorithm detects these cases that are the failure cases.

We apply our modification procedure only if we are sure that the new symbolic marking represents only reachable markings of the instantiated programs. To this purpose we define "state change for one process" sequences.

V.1. "State change for one process" sequences

A "state change for one process" sequence is such that after its firing two places p_{in} and p_{out} may be identified. The number of tokens in place p_{out} is decreased by one, the number of tokens in place p_{in} is increased by one and the marking of the other places is not modified. The sequence can be iterated as long as there are processes in place p_{out}.

Definition 16 "State change for one process" sequence

A sequence s is a "state change for one process" sequence iff:

- $s = t_1 \ldots t_n$, $\forall i \in \{1,\ldots,n\}$, $t_i \in T_p \cup T_c \cup T_{Synch}$,
- There is exactly one place $p_{out} \in P_p$, such that $\sum_{i=1,\ldots,n} (Post(p_{out},t_i) - Pre(p_{out},t_i)) = -1$,
- There is exactly one place $p_{in} \in P_p$, such that $\sum_{i=1,\ldots,n} (Post(p_{in},t_i) - Pre(p_{in},t_i)) = 1$,
- $\forall p'' \in P_c \cup P_p$, $p'' \notin \{p_{out}, p_{in}\}$, $\sum_{i=1,\ldots,n} (Post(p'',t_i) - Pre(p'',t_i)) = 0$.
- $\forall k \leq n$, $\sum_{i=1,\ldots,k} (Post(p_{out},t_i) - Pre(p_{out},t_i)) \geq -1$.

Property 8

Let s be a "state change for one process" sequence, sm and sm' two symbolic markings. The sequence s is enabled from sm. If sm [s> sm' and sm'.M(p_{out}) ≥ 1 then sm' enables s.

Proof

- sm[s>sm' then sm = $sm_0[t_1> \ldots [t_n>sm_n$ = sm'.
- $\forall i$, $1 \leq i \leq n$, $\forall m' \in set[sm_i]$, $\exists m \in set[sm]$, $m[t_1 \ldots t_i>m'$. If i = n, $\forall m' \in set[sm']$, $\exists m \in set[sm]$, m[s>m'.
- $\forall p \in P_p \cup P_c$, $p \neq p_{in}, p_{out}$, $m'(p) = m(p)$, $m'(p_{in}) = m(p_{in})+1$, $m'(p_{out}) = m(p_{out})-1$.

As sm'.M(pout) ≥ 1 and $\forall k \leq n$, $\sum_{i=1,\ldots,k} (Post(p_{out},t_i) - Pre(p_{out},t_i)) \geq -1$, $\forall m' \in set[sm']$, sm'[s>.

V.2. Graph

We now have to define the procedure we apply when we compute a symbolic marking, sm', strictly greater than one of its ancestors, sm, and not included in one of its ancestors. Three cases are possible, they are examined in the same order as we present them:

1) The operator "≥" is associated with more places in sm' than in sm. We do not modify sm' and we continue the building of the branch from it. As the number of places is finite, the number of "≥" associated with places cannot increase infinitely.

2) The sequence that leads from sm to sm' is a "state change for one process" sequence. If sm'.M(p_{in}) ≥ Limit(p_{in}), we modify sm' by associating the operator "≥" to place p_{in} else sm' is not modified. The values less than Limit(p_{in}) are in finite number.

3) The building algorithm fails. We cannot represent with a finite number of symbolic markings the set of reachable states.

<u>Example 6</u> Failure case

The Petri net of Figure 2 models the access to resources A and B by a finite but unknown number of processes. Resource A is free only if Resource B is not and conversely. A process with no resource is in place idle and a process with one resource is in place busy.

Initially all the processes are in place idle and one resource A is available. This global state is represented by the symbolic marking {idle ≥ 1, A = 1}.

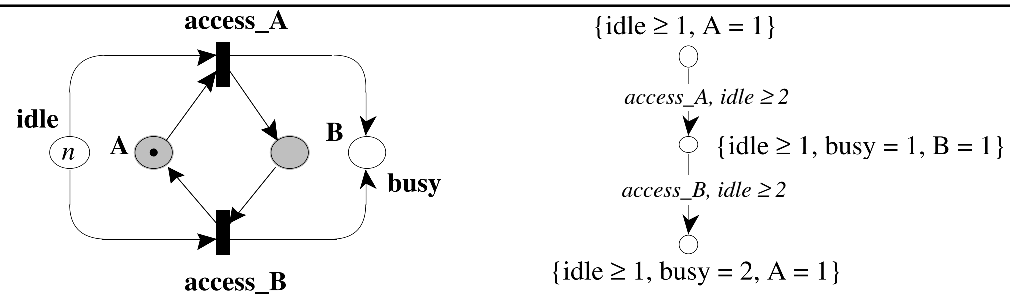

Figure 2

The firing of transition access_A leads to the symbolic marking {idle ≥ 1, busy = 1, B = 1} and the firing of transition access_B leads to the symbolic marking {idle ≥ 1, busy = 2, A = 1}. This symbolic marking is strictly greater than the initial one. However the sequence access_A, access_B is not a "state change for one process" sequence. After its execution two tokens have moved from place idle to place busy. To avoid the building of an infinite branch we should represent, with a finite number of symbolic markings, all the distributions of processes between places idle and busy with an even number of processes in place busy. It is not possible.

<u>Property 9</u>

If case 2 is applied (sm[s>sm', sm' > sm and s is a "state change for one process" sequence) the modified symbolic marking represents only markings that are reachable from markings represented by sm after iterations of sequence s.

<u>Proof</u>

We note sm' the symbolic marking before the modification and sm'_{modif} the symbolic marking after the modification.

$\forall p \in P_p \cup P_c$, $p \neq p_{in}$, $sm'_{modif}.M(p) = sm'.M(p)$, $sm'_{modif}.OP(p) = sm'.OP(p)$

$sm'_{modif}.M(p_{in}) = sm'.M(p_{in})$ and $sm'_{modif}.OP(p_{in}) = sm'.OP(p_{in})$.

As sm[s>, sm.M(p_{out}) ≥ 1, then sm'.M(p_{out}) ≥ 1. By property 8, sm'[s>, and by extension ∀m' ∈ set[sm'], m'[α.s> where $\alpha \in \mathbb{N}$, $1 \leq \alpha \leq m'(p_{out})$.

As sm' > sm, sm' is not included in sm, case 1 is not verified and the markings represented by sm' are reachable from markings represented by sm after the execution of s:

$\forall p \in P_p \cup P_c$, sm'.OP(p) = sm.OP(p),

$\exists p \in P_p \cup P_c$, sm'.M(p) > sm.M(p) and sm'.OP(p) = "=", $p = p_{in}$,

sm'.OP(p_{out}) = "≥".

If each marking m' represented by sm'_{modif} can be reached by a marking m after the firing of α' times the sequence s, α' = m'(p_{in}) - sm'.M(p_{in}), m is as follows:

$\forall p \in P_p \cup P_c$, $p \neq p_{in}, p_{out}$, m(p) = m'(p), m(p_{out}) = m'(p_{out}) + α', m(p_{in}) = m'(p_{in}) - α' = sm'.M(p_{in}).

All the possible markings m are represented by sm'. As sm'[α.s>, set[sm'_{modif}] represents markings that are reachable from markings represented by sm'. Therefore, they are reachable from sm.

Example 7 Symbolic graph

Figure 3 represents the symbolic graph of the Petri net of Figure 1. The symbolic marking associated with the gray node is the initial symbolic marking and represents markings of programs instantiated with at least one process. The markings associated with black nodes are reachable markings of the program instantiated with one process. The symbolic marking associated with the striped node represents reachable markings of programs instantiated with at least three processes. The others symbolic markings represent reachable markings of programs instantiated with at least two processes.

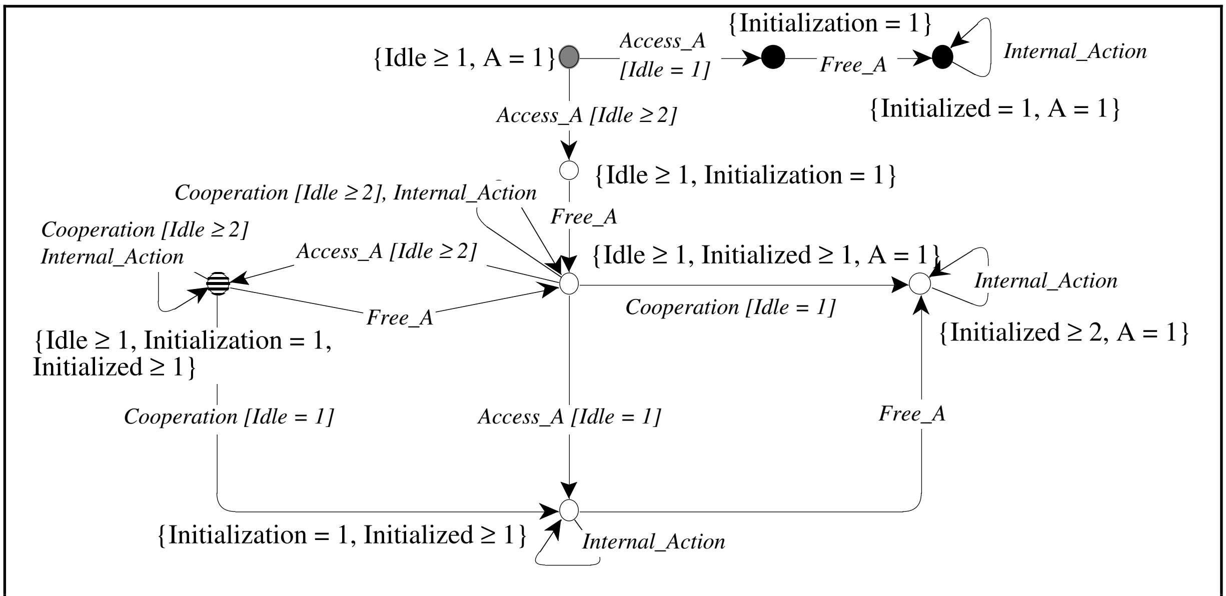

Figure 3

The following property ensures that the building algorithm ends in a finite time. The result is the detection of a failure case or the symbolic graph of the model.

Property 10

The algorithm builds a finite symbolic graph or fails.

Proof

We show that:

- if the graph is infinite, there is an infinite strictly increasing sequence of symbolic markings,
- we cannot have an infinite strictly increasing sequence of symbolic markings.

The proof follows from cases 2 and 3 of the building algorithm and from the finiteness of the number of places.

The following theorem ensures that the symbolic graph represents exactly what we expected.

Theorem 1

- The symbolic graph represents all the reachable states of each instantiated program.
- Each state represented by a symbolic marking of the symbolic graph is a reachable state of an instantiated program.

- For each sequence of actions and states of each instantiated program there is a path in the symbolic graph that represents the succession of both states and actions.

Proof

We use properties 4, 7 and 9.

VI. Symbolic graph and verification of properties

Some usual properties can efficiently be verified on the symbolic graph. This is the case for the detection of deadlocks or for the reachability problem. In [25] and [26] we show how to verify properties expressed as temporal logic formulas on the symbolic graph. Here we consider properties whose verification depends only on the reachable states. The structure of the graph is not significant.

VI. Detection of deadlocks

To detect if a Petri net is deadlock free, it is sufficient to verify that at least one transition is enabled from each reachable marking. If we consider the symbolic graph, the same verification is possible.

Theorem 2

Let $P = \langle P_p \cup P_c, T_p \cup T_c \cup T_{synch}, Pre, Post, m_0 \rangle$ be a M2P, SG the symbolic graph of P and $sm_1, \ldots, sm_n$ are the nodes of the graph.

A program instantiated with n processes is deadlock free if $\forall sm_i, n \geq B(sm_i)$, $\exists t \in T_p \cup T_c \cup T_{synch}$, $sm_i[t\rangle$.

Proof

We use the fact that the symbolic graph represents exclusively all the markings of all the instantiated programs and property 4.

- A symbolic marking enables a transition if and only if all the markings it represents enables the same transition. A program is deadlock free if at least one transition is enabled from all the symbolic markings that represent markings of the instantiated program (all the symbolic markings sm such that $n \geq B(sm)$).

- Let sm_i be a symbolic marking that enables no transition. Then no transition is enabled from all the markings represented by sm_i. All the programs instantiated with n processes, $n \geq B(sm_i)$ are not deadlock free.

The detection of symbolic markings that enable no transition gives the knowledge of the instantiated programs, may be in infinite number, that are not deadlock free. Another information is the class of markings from which no evolution is possible.

VI.2. Reachability problem

The reachability problem can easily be solved on the symbolic graph. To compute the set of markings represented by a general symbolic one that are reachable, we only have to compute the intersection between the general symbolic marking and each symbolic marking of the symbolic graph.

Theorem 3

Let $P = \langle P_p \cup P_c, T_p \cup T_c \cup T_{synch}, Pre, Post, m_0 \rangle$ be a M2P, SG the symbolic graph of P, $sm_1, \ldots, sm_n$ are the nodes of the graph and gsm a general symbolic marking.

The reachable markings represented by gsm are the markings that belong to one of the symbolic marking $sm'_i = sm_i \cap set[gsm]$.

Proof

We use the definition 10, property 3 and the fact that the intersection between a general symbolic marking and a symbolic marking is represented by a symbolic marking.

If the general symbolic marking is a marking m of an instantiated program, it is sufficient to find a symbolic marking sm of the graph such that $sm \cap m \neq \emptyset$. The intersection between a symbolic marking and a marking is empty or equals the marking.

VII. Conclusion

We have presented a method to study programs with finite but unknown number of processes. We have defined symbolic markings that represent sets of markings of instantiated programs and we have presented an algorithm to build a symbolic graph that represents the behavior of all the instantiated programs.

An important feature of the symbolic graph is that it allows us to solve the reachability problem. That is not possible with the methods already proposed to verify properties of all the instantiated programs. Another interest of the symbolic graph is that it allows us to detect deadlocks and to verify temporal logic formulas.

To characterize the class of Petri nets we can study, it is reasonable to first consider a model less expressive than Petri nets. The algorithm to build the symbolic graph is implemented in the software engineering framework AMI [7]. Therefore, it is possible to test lot of models and to identify more precisely failure cases.

Further works will deal with the possible extensions of the studied programs. The model presented in [23] allows one to distinguish dynamically the behavior of one process. The adaptation of the symbolic graph building algorithm is described in [24].

Our model is less expressive than colored Petri nets [14], but it allows one to study programs without fixing the value of the parameter. Some authors have proposed methods to build reduced reachability graphs of colored Petri nets [12], [4]. The building algorithms exploit the behavioral symmetries of processes. In [6] and [8] the author present methods, with the same principles, to reduce the size of the graph on which temporal logic formulas will be verified. To apply these two sets of methods, the parameters of the model must be fixed. It will be interesting to study how to use some of these principles to extend the subclass of Petri nets for which we can build a symbolic graph.

References

[1] Apt K.R., Kozen D.C., "*Limits for automatic verification of finite-state concurrent systems*", Information Processing Letters, vol 22, n°6, pp 307-309, 1986.

[2] "*On the Automatic Computation of Network Invariants*".

[3] Berthelot G., "*Checking properties of nets using transformations*", Advances in Petri Nets, LNCS 222, pp 19-40, 1986.

[4] Chiola G., Dutheillet C., Franceschinis G., Haddad S., "*Stochastic Well-Formed Colored Nets and Symmetric Modeling Applications*", IEEE Transactions on Computers, vol 42, n° 11, pp 1343-1360, 1993.

[5] Clarke E.M., Burch J.R., Grumberg O., Long D.E., McMillan K.L., "*Automatic verification of sequential circuit designs*", Philosophical Transactions of the Royal Society of London, Series A, vol 339, pp 105-120, 1992.

[6] Clarke E.M., Filkorn T., Jha S., "*Exploiting Symmetry in Temporal Logic Model Checking*", Proceedings of the 5^{th} International Conference, CAV'93, LNCS 697, pp 450-462, 1993.

[7] CPN-AMI, "*Reference manual, 1.2 version*", 1994.

[8] Emerson E.A., Sistla A.P., "*Symmetry and Model Checking*", Proceedings of the 5^{th} International Conference, CAV'93, LNCS 697, pp 463-478, 1993.

[9] Finkel A., "*A minimal Coverability Graph for Petri Nets*", Proceedings of the 11th International Conference on Application and Theory of Petri Nets, Paris, France, pp 233-245, 1990.

[10] German S., Sistla A.P., "*Reasoning about Systems with Many Processes*", JACM, vol 39, pp 675-735, 1992.

[11] Haddad S., "*Une catégorie régulière de réseaux de Petri de haut-niveau : définition, propriétés et réductions*", Thèse d'Université, Université Paris 6, 1987.

[12] Huber P., Jensen A.M., Jepsen L.O., Jensen K., "*Reachability trees for high-level Petri Nets*", Theoretical Computer Science, vol 45, pp 261-292, 1986.

[13] Hyung L.-K., Favrel J., "*Corrections to Generalized Petri Net Reduction Method*", IEEE Transactions on Systems, Man, and Cybernetics, vol 19, n°5, pp 1328-1329, 1989.

[14] Jensen K., Rozenberg G., (Eds), "*High-Level Petri Nets*", *Ed.* Springer-Verlag, 1991.

[15] Karp R.M., Miller R.E., "*Parallel Program Schemata*", Journal of Computer science, vol 3, n°2, May 1969.

[16] Kurshan R.P., Merritt M., Orda A., Dachs S.R., "*A Structural Linearization Principle for Processes*", Proceedings of the 5th International Conference, CAV'93, LNCS 697, Elounda, Greece, pp 491-504, 1993.

[17] Reisig W., EATCS Monographs on Theoretical Computer Science, Springer Publ. Company, vol 4, 1985.

[18] Rho J.K., Somenzi F., "*Automatic generation of networks invariants for the verification of iterative sequential systems*", Proceedings of the 5th International Conference, CAV'93, LNCS 697, Elounda, Greece, pp 123-137, 1993.

[19] Shtadler Z., Grumberg O., "*Network Grammars, Communication Behaviors and Automatic Verification*", Proceedings of the International Workshop on Automatic Verification Methods for Finite State Systems, LNCS 407, Grenoble, France, pp 152-165, 1989.

[20] Suzuki I., "*Proving properties of a ring of finite-state machines*", Information Processing Letters, vol 28, n°4, pp 213-214, 1988.

[21] Valmari A., "*Stubborn Sets for Reduced State Space Generation*", Advances in Petri Nets, LNCS 483, pp 491-515, 1991.

[22] Valmari A., "*Compositionnal State Space Generation*", Advances in Petri Nets, LNCS 674, pp 427-457, 1993.

[23] Vernier I.: "*Specification And Verification of Parametrized Parallel Programs*", Proceedings of the 8th International Symposium on Computer and Information Sciences, Istanbul, Turkey, pp 622-625, 1993.

[24] Vernier I., "*Graphe symbolique paramétré de réseaux de Petri et Logique temporelle*", Thèse d'Université, Université Paris 6, 1994.

[25] Vernier I., "*Parameterized Evaluation of CTL-X Formulae*", Proceedings of the 1st International Conference on Temporal Logic Workshop, Bonn, Germany, pp 22-31, 1994.

[26] Vernier I., "*Symbolic Verification of Parallel programs*", Institut Blaise Pascal Report, to appear, 1995.

[27] Wolper P., Lovinfosse V., "*Verifying Properties of Large Sets of Processes with Network Invariants*", Proceedings of the Proc.International Workshop on Automatic Verification Methods for Finite State Systems, LNCS 407, Grenoble, France, pp 68-80, 1989.