

An Incremental Verification Technique Using Decomposition of Petri Nets

Serge Haddad¹, Jean-Michel Ilić² and Kais Klai²

(1) lab. Lamsade, Université Paris Dauphine, France, (2) lab. LIP6, Université Paris VI, France
Serge.Haddad@lamsade.dauphine.fr , Jean-Michel.Ilic@lip6.fr, Kais.klai@lip6.fr

Abstract— We propose a modular verification technique for bounded Petri nets which efficiency relies on both behavioral and structural features. By focusing on linear evenemential temporal logic formula, we demonstrate how to choose a subnet on which it is enough to perform the model checking.

Keywords— Verification and validation, Modular verification, Decomposition, Petri nets.

I. INTRODUCTION

We present in this work an efficient model checking algorithm for bounded Petri nets. Elaborating such a method is useful since it is well known that the size of the reachability space of a bounded Petri net is a non primitive function of the size of the net. In order to avoid the building of the whole reachability space, we propose in this work a modular verification technique which relies on both behavioral and structural features.

As far as the verification is concerned, taking benefit from the structural composition of Petri nets is known to be a hard problem. Mainly, the researchers have obtained by composition the preservation of basic properties. In [1], Souissi brings out a particular structure of interface, called communication medium, preserving the liveness and boundedness properties. In [2], Sibertin-Blanc shows how to preserve some services of client-server protocols.

More abstract works have highlighted general methodologies for modular verification of temporal logic properties, by regarding subsystems like transition systems to be composed. The abstraction concept exploited in [3] allows one to reduce an infinite system to a finite one, over which the model checking can be performed. The presented method needs, however, some ingenuity and cannot be fully mechanized. In [4], Valmari describes how the state space of the whole system can be reduced by first reducing state spaces of modules before combining them.

In this paper, we deal with the verification of temporal logic formula ($LTL \setminus X$) without generating the full state space. By focusing on evenemential system properties, we plan to define a subnet as small as possible on which it is enough to perform the model checking. The major problem is to assert that the remaining part of the net does not constrain the behavior of the chosen subnet. Recent works have already specified a non constraining relation however only applicable with restrictions, either on the property specification language [5],

or on the system model [6]. Here, we aim to go a step forward in order to be more general and efficient. By the way, our approach is behavioral, however improved with structural information and the analysis of the property formula. The fact that the formula is taken into account leads to a notion of *observed transitions*, corresponding to the transitions the labels of which appear in the formula. We call by T_f the set of transitions observed in formula f .

We are interested in any kind of properties based on infinite sequences of observed transitions. We first give a structural characterization of a family of subnets, from which we are sure that the *infinite observed language* of the system is covered. Hence, one can directly deduce that the system satisfies a property by means of a model checking on such a subnet. The non-constraining relation is only tested in case the property is detected false. Actually, to find directly a non-constrained subnet remains a hard task, therefore our approach is incremental, driven by means of structural information.

The paper is scheduled as follows : in part II, we introduce our modular approach based on Petri net decompositions; in part III, we give a structural definition of a family of covering subnet; in part IV we propose an efficient algorithm which can be used to test the *non-constraining relation* on the fly; in part V our first experimental results are presented. Finally, part VI contains our conclusion and research perspectives.

II. A MODULAR VERIFICATION METHODOLOGY

Let us consider a bounded Petri net Σ and an evenemential temporal logic formula f of $LTL \setminus X$. In such a formula, the atomic propositions are made from names of Σ transitions (labels) and the satisfaction of f concerns the sequences of transition firings that are maximal in Σ . In this paper, we concentrate on $L^{f\infty}(\Sigma)$ the set of infinite observed sequences of Σ with respect to f . $L^{f\infty}(\Sigma)$ is obtained by projection of the infinite sequences of Σ on the labels occurring in f . Given any subnet Σ_v of Σ , the following point holds :

$$(i) (L^{f\infty}(\Sigma_v) = L^{f\infty}(\Sigma)) \Rightarrow (\Sigma_v \models f \Leftrightarrow \Sigma \models f)$$

Our approach mainly consists in proving the former equality of languages by means of language inclusions : $L^{f\infty}(\Sigma) \subseteq L^{f\infty}(\Sigma_v)$ (covering relation) and $L^{f\infty}(\Sigma_v) \subseteq L^{f\infty}(\Sigma)$ (non constraining relation).

One key point of our method is that the covering relation is obtained by structural considerations. Actually,

we define structurally a family Cov^f of subnets satisfying this relation. An element of Cov^f is called a *covering subnet*.

The choice of an appropriate subnet Σ_v must accord with the following features :

- With regard to the verification stage, it must contain the transitions which labels are in f . Let T_f be the corresponding subset of transitions, namely *observed transitions*.
- its infinite observed language must tend towards those of Σ , in order to reach the required language equality given by (i).

Two particular elements of Cov^f can be highlighted : Σ itself and the subnet Σ_f which only contains the transitions T_f and its *immediate environment* (adjacent places $\bullet T_f \bullet$ and their adjacent transitions $\bullet(\bullet T_f \bullet)\bullet$). As far as Σ is concerned, the languages equality (i) is straightforwardly ensured, but there is no gain. Hence, Σ should be the last choice to do in Cov^f after all other possibilities. With the subnet Σ_f , one can hope to improve the model checking since its size can be considerably smaller than Σ . However, it must be ensured that the subnet does not allow extra behaviors by testing the non constraining relation : $L^{f\infty}(\Sigma) \supseteq L^{f\infty}(\Sigma_f)$.

The need to test the non constraining relation is in fact general, and should be acted whichever the subnet Σ_v chosen in Cov^f . In part IV, we bring out a sufficient condition to ensure this relation. It is based on an efficient test of language inclusion that focuses on the interface between Σ_v and its environment Σ_{env} . In our view, Σ is decomposed in a pair of subnets Σ_v and Σ_{env} , such that they only share some transitions, called their interface.

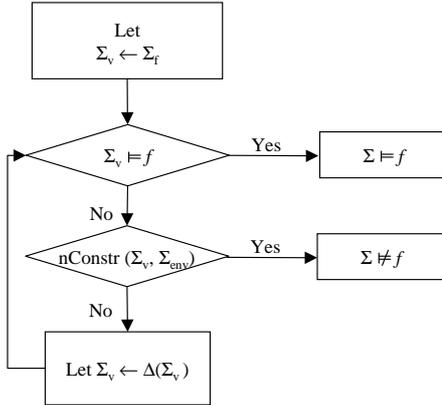


Fig. 1. The modular verification approach

We must solve an additional problem : a subnet in Cov^f is not guaranteed to be bounded, so could lead to model an infinite state transition system. Since the whole net Σ is assumed to be bounded, one can control the subnet by the knowledge of bounds for its places. A standard structural technique that can be used is the

computing of places invariants. One can also constrain the behavior of the subnet by a synchronization with its environment. In the remaining part of the paper, we denote by Σ_v a subnet of Cov^f associated with its control information.

Figure 2 schemes our iterative approach of verification, starting from Σ_f . Within each iteration, a subnet is selected in Cov^f , thus ensuring the covering relation. Firstly, the formula f is checked on the selected subnet Σ_v , so one can immediately deduce that Σ satisfies f whenever f holds on Σ_v . Otherwise, one can decide the non satisfaction of f in Σ only if Σ_{env} is proved to be non constraining w.r.t. Σ_v . If the non constraining relation does not hold, the former verification stage is processed again but on another covering subnet, the language of which tends more towards $L^{f\infty}(\Sigma)$.

In order to ease the choice of a such a subnet, we define a morphism Δ as a structural transformation over Cov^f . Intuitively, the more places, the more the language of the subnet is limited. In fact, the appropriate subnet Σ_v is built iteratively from Σ_f , by adding, at each time some places of Σ and their adjacent transitions, to an already built covering subnet. Thus, from a subnet of Cov^f , the function $\Delta : 2^\Sigma \rightarrow 2^\Sigma$ yields the next one, such that we ensure : $L^{f\infty}(\Sigma_f) \supseteq L^{f\infty}(\Sigma_v) \supseteq L^{f\infty}(\Delta(\Sigma_v)) \supseteq L^{f\infty}(\Sigma)$.

III. COVERING SUBNET

Given a Petri net Σ and its subset of observed transitions T_f , we first give a structural characterization of a family of covering subnets, namely the set Cov^f . Then, we detail how Δ is used to specify structurally the considered subset of Cov^f . For language consideration, we assume that an initial marking is defined for Σ . Moreover, observe that the initial marking of a subnet of Σ is deduced from this marking by projection on the places subnets.

A. Structural characterization

The covering subnets of Cov^f are such that they contain the transitions of T_f , moreover, each place must be associated with its input transitions. The idea is to feed up these places sufficiently. Of course, connections between places and transitions in Σ are preserved.

Definition 1 (Family of covering subnets) Let $N_\Sigma = \langle P, T, W \rangle$ be the structure of a Petri net Σ , $T_f \subseteq T$ and $Cov^f \in 2^{N_\Sigma}$, a family of subnets.

Cov^f is said to be a covering family of Σ w.r.t. T_f , iff whatever Σ_v of Cov^f , which structure is $\langle P_v, T_v, W_v \rangle$, the three following conditions hold :

- $T_f \subseteq T_v \subseteq T$
- $\forall p \in P_v, \forall t \in T, (p \in t^\bullet \implies t \in T_v)$
- $\forall p \in P_v, \forall t \in T_v,$
 $W_v^+(p, t) = W^+(p, t)$ and $W_v^-(p, t) = W^-(p, t)$

The following property states that the observable infinite language of any element of Cov^f contains the in-

finite observed language of Σ . We denote by σ^f the projection of a sequence σ on the transitions of T_f .

Property 1: $\forall \Sigma_v \in Cov^f, L^{T_f^\infty}(\Sigma) \subseteq L^{T_f^\infty}(\Sigma_v)$.

Proof: Let M be the initial marking of Σ and M_v be its projection on Σ_v . Let σ be an infinite sequence of Σ s.t. σ^{T_f} , its projection on T_f , is an infinite sequence. We show that σ^{T_v} , the projection of σ on T_v , is enabled in Σ_v , then we can directly deduce that $\sigma^{T_f} \in L^{f^\infty}(\Sigma_v)$

Assume that $\sigma^{T_v} \notin L^{T_v^\infty}(\Sigma_v)$, thus, there is an occurrence of a transition t of T_v which makes it unfirable from one intermediate marking M'_v . So, we can write σ^{T_v} like $\alpha t \beta$ where α is a finite prefix, $t \in T_v$ and β is an infinite suffix. Let M'_v be the marking reached from M_v by the firing of α .

Let p be a place of Σ_v s.t. $M'_v(p) < W^-(p, t)$. Moreover, let α' be the finite prefix of σ which projection on T_v is α , and M' be the marking of Σ reached from M by the firing of α' .

Since the sequence σ is enabled in M , the transition t is obviously enabled in M' . In contrast, t is not enabled in M'_v . Therefore, we can develop the prefix α' in $\alpha'_1 \alpha'_2$ s.t. α'_2 is a sequence wherein there is no occurrence of a transition of T_v .

The fact that t is disabled in M'_v implies that it is the firing of α'_2 which enables this transition in M' . So, after the firing of α'_2 in Σ , the place p should contain enough tokens to enable the firing of t . This situation is absurd since, by construction of Σ_v , whenever a place p is in Σ_v , all its input transitions are also in Σ_v . ■

Let us now define the particular case of the *formula subnet*, which is the first element selected in Cov^f .

B. Initial choice

The smallest subnet one can consider mainly focuses on the observed transitions, T_f . In order to model its interaction with the remaining part of the system, we propose to add the transitions which firings can directly change the marking of the places adjacent to the T_f transitions. We call this subnet the *formula subnet*, Σ_f . For construction purpose, the interface between Σ_f and its environment is highlighted by partitioning the transitions of the considered subnet in two subsets : the local transitions T_{local_0} and the interface transitions T_{int_0} .

Definition 2: Let $\langle P, T, W \rangle$ be the structure of Σ and M its initial marking. If T_f is the subset of observed transitions, then the *formula subnet* $\Sigma_f = \langle P_0, T_0 = T_{local_0} \cup T_{int_0}, W_0, M_0 \rangle$ is defined as follows :

- Let $P_0 = \bullet T_f \bullet$ be the adjacent places of the observed transitions, $T_{int_0} = \bullet P_0 \bullet \setminus T_f$ be their adjacent transitions not already in T_f .
- $\forall p \in P_0, \forall t \in T_0,$
 $W_0^+(p, t) = W_0^+(p, t)$ and $W_0^-(p, t) = W_0^-(p, t)$
- $T_{local_0} = T_f$
- $\forall p \in P_0, M_0(p) = M(p)$

C. Structural transformation

We now define the structural transformation function Δ . Applying Δ on a subnet Σ_{v_i} , ($i = 1, 2, \dots$), in Cov^f consists of adding some Σ elements to Σ_{v_i} . Informally, the new subnet $\Sigma_{v_{i+1}}$ is obtained by adding the adjacent places of T_{int_i} not already in P_i with their adjacent transitions in Σ .

Definition 3: Let Σ_{v_i} be an element of Cov^f s, $\Delta(\Sigma_{v_i})$ is the Petri net $\Sigma_{v_{i+1}}$, which structure $\langle P_{i+1}, T_{i+1} = T_{local} \cup T_{int_{i+1}} \rangle$ is defined as follows :

- $P_{i+1} = P_i \cup \bullet T_{int_i} \bullet$
- $T_{i+1} = T_i \cup \bullet (\bullet T_{int_i} \bullet) \bullet$
- $T_{int_{i+1}} = \bullet (\bullet T_{int_i} \bullet) \bullet \setminus T_{int_i}$
- $T_{local_{i+1}} = T_{i+1} \setminus T_{int_{i+1}}$

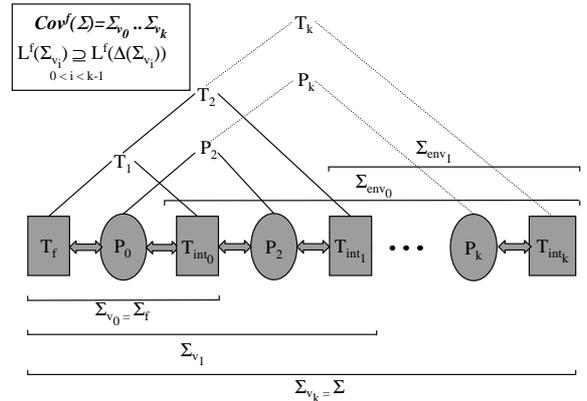


Fig. 2. A family of covering subnets

Figure 2 is an illustration of the family of the considered covering subnets. It is computed from $\Sigma_{v_0} = \Sigma_f$ by applying Δ iteratively. By construction, each of these subnets respects the structural conditions of *Definition 1*. The worst case corresponds to $\Sigma_{v_k} = \Sigma$ since only Σ elements can be added at each iteration. According to a subnet Σ_{v_i} , the subnet Σ_{env_i} denotes the environment of Σ_{v_i} in Σ , and T_{int_i} is the (shared) interface between Σ_{v_i} and Σ_{env_i} . One may note that any subnet Σ_{env_i} and the corresponding interface T_{int_i} are entirely and clearly featured right through the definition of the subnet Σ_{v_i} .

Property 2: $\forall \Sigma_v \in Cov^f$, we have :
 $L^{f^\infty}(\Sigma) \subseteq L^{f^\infty}(\Delta(\Sigma_v)) \subseteq L^{f^\infty}(\Sigma_v)$.

IV. BEHAVIORALLY COVERING OF THE SUBNET BY THE NET

The subnet Σ_v on which the verification process will be projected must not only cover the behavior of Σ according to T_f but also guarantee no extra behaviors. In other words, the language $L^{f^\infty}(\Sigma_v)$ must be a subset of $L^{f^\infty}(\Sigma)$. A naive test of this condition could be hard, therefore we propose now a sufficient behavioral condition, namely the non constraining relation, tested with the only regard to the subnet interface. Moreover, we

demonstrate how to test it efficiently.

A. Problem reduction

Let us consider a decomposition $\Sigma = \Sigma_v \parallel \Sigma_{env}$ in two subnets, such that only a set of transitions is shared, namely, T_{int} . For language consideration, let us observe that the respective initial markings of Σ_v and Σ_{env} , are directly deduced by projections of the initial marking of Σ over the subnets places.

The non-constraining property is defined as an asymmetrical relation between Σ_v against its environment Σ_{env} : for each enabled infinite sequence in Σ_v , there must be an infinite sequence enabled in Σ_{env} , which projection on T_{int} is the same. Let $L^{int\infty}(\Sigma_v)$ and $L^{int\infty}(\Sigma_{env})$, be respectively the infinite observed languages of Σ_v and Σ_{env} .

Definition 4 (Non constraining relation) :

Σ_{env} is said to be T_{int} non constraining for Σ_v iff $L^{T_{int}}(\Sigma_v) \subseteq L^{T_{int}}(\Sigma_{env})$

Property 3: : Let Σ_v be an element of Cov^f . We have $L^{f\infty}(\Sigma_v) \subseteq L^{f\infty}(\Sigma)$ if Σ_{env} is T_{int} non constraining for Σ_v .

Proof: Let M be the initial marking of Σ and let M_v and M_{env} be the ones of Σ_v and Σ_{env} . Assume that Σ_{env} is T_{int} non constraining w.r.t. Σ_v . We now consider an infinite sequence σ_v of Σ_v and show that there exists an infinite sequence σ in $L^\infty(\Sigma)$ which has the same projection on T_f than σ_v .

Let β_v be a finite prefix of σ_v , we proceed by induction on $|\beta_v|$ (number of transitions in β_v).

– $|\beta_v| = 0$ Obvious.

– Induction hypothesis: assume that there is a number n s.t. the property is true for every finite prefix β_v of σ_v with $|\beta_v| \leq n$.

Let β_v be a prefix of σ_v s.t. $|\beta_v| = n + 1$, then $\exists t \in T_v$ s.t. $\beta_v = \beta'_v t$. Let M'_v be the marking of Σ_v s.t. $M_v \xrightarrow{\beta'_v} M'_v$. Observe that t is enabled in M'_v . Since the sequence β'_v satisfies the induction hypothesis, there is a sequence $\beta' \in L(\Sigma)$ which has the same projection on T_f than β'_v . Let M' be the marking of Σ such that $M \xrightarrow{\beta'} M'$. Let β' be the shortest sequence which satisfies this condition.

Let P_v be the set of places in Σ_v .

We now demonstrate that there is a sequence η within which there is no occurrence of a transition of the interface and such that $M' \xrightarrow{\eta} t$. There are two possible cases :

first case $t \notin T_{int}$ (i.e. $t \in T_{local}$):

By construction of Σ_v , t is enabled in Σ from M' . In fact, $M_v^{P_v} = M'^{P_v}$ (marking projection on places of P_v) and the set $\bullet t$ in Σ_v is the same than in Σ . Since the in-

put places of t make it enable in M'_v we deduce that t is also enabled in M' (in this case η is the empty sequence).

second case $t \in T_{int}$:

Let β_{env} be the shortest sequence in Σ_{env} which has the same projection on T_{int} than β_v . Such a sequence exists because Σ_{env} is T_{int} non-constraining for Σ_v .

Let us develop β_{env} in $\alpha\eta t$, s.t. η is the largest subsequence within which there is no occurrence of a transition of T_{int} . Let M'_{env} be the marking of Σ_{env} s.t. $M_{env} \xrightarrow{\alpha\eta} M'_{env}$.

Observe that every transition occurring in η is local to Σ_{env} . By construction, the input places of these transitions cannot belong to Σ_v . So, η is necessary enabled from M' in Σ . The firing of η from M' leads to a marking, namely M'' , which is nothing but the composition of M'_v and M'_{env} . Because t is enabled in M'_v and M'_{env} , it is also enabled in M'' .

Consequently, the sequence $\sigma = \beta' \eta t$ is enabled in Σ from M and it has the same projection on T_f than β_v . ■

The way to verify the non constraining relation efficiently is the main difficulty in our approach. However, we do not have to check an inclusion between two Petri languages but between their projections on their transitions interface. We propose to abstract all transitions but these of T_{int} , then to check the inclusion from two deterministic automata corresponding to Σ_v and Σ_{env} , namely A_v and A_{env} . In this case, the inclusion test is reduced to a synchronized product between A_v and A_{env} , combined with an equality accordingly to the equation : $L(A_v) = L(A_v) \times L(A_{env})$. For sake of efficiency, the synchronized product is computed on the fly, during the construction of A_v and A_{env} . Also, the violation of the former equality is tested on the fly, since it only requires that every transition of A_v is synchronized with a transition of A_{env} . The worse case complexity is obtained whenever the inclusion holds (and the system property is checked false). In this case, A_v must be synchronized entirely.

The drawback effect of our techniques could be the operations of determinizations, since it is known to cause an exponential blow up of memory : each node in the deterministic structure may represent a subset of markings. However, whenever T_{int} is relatively small with respect to the considered set of transitions, we know that in practice the size of the deterministic automaton is lower than the original reachability graph [7].

The next section brings out a general algorithm used to build a deterministic automaton on the fly, from a subnet and its interface. Because its language exactly corresponds to the infinite observed language of the subnet, one can use it to abstract Σ_v and Σ_{env} in the former inclusion test. Such a structure is called a *deterministic meta graph*.

B. Equivalent deterministic meta graph

According to a Petri net Σ which its initial marking is M and a subset T_{obs} of observed transitions, a deterministic meta graph represents the observed sequences

Algorithm IV.1 Equivalent deterministic graph $(\langle \Sigma, M \rangle, T_{Obs})$

```
begin
1   $S_0 = \epsilon\text{Closure}(M)$ ;
2  insert  $S_0$  in  $G$ 
3  push  $\langle S_0, \text{FirableObs}(S_0) \rangle$  onto  $stack$ ;
4  while  $stack$  is not empty do
5       $e$  the top element of  $stack$ 
6      if  $e.\text{second}$  is not empty then
7          let  $t$  be the first transition in  $e.\text{second}$ 
8          erase  $t$  in  $e.\text{second}$ 
9           $reached := \epsilon\text{Closure}(\text{move}(e.\text{first}, t))$ 
10         if ( $reached$  is not already in  $G$ ) then
11             push  $\langle reached, \text{FirableObs}(reached) \rangle$ 
12             onto  $stack$ 
13             insert  $reached$  in  $G$ 
14         endif
15         add new successor to  $e.\text{first}$ 
16     else
17         pop the top element of  $stack$ 
18     endif
19 endwhile
end
```

w.r.t. T_{Obs} . Hence, it appears to be a deterministic graph which arcs are uniquely labelled with the occurrences of the observed transitions.

Its construction starts from a relabelling of the non observed transitions in the net, by the same symbol ϵ . In fact, these transitions are considered equivalent. Then, a determinization process is applied from firing sequences, which mainly consists of achieving the *transitive closure* of the ϵ labelled transitions that may appear within the reachability graph.

The algorithm of Figure IV.1 uses a simplified variant of the classical transitive closure algorithm for graphs. Instead of computing the transitive closure of a given graph w.r.t. the ϵ transitions, this algorithm only computes the closure from subsets of markings, resulting on larger subsets, called *meta states*. Let $\epsilon\text{Closure}$ be the function that builds, from a set of markings S its ϵ transitive closure, that means all the markings reachable from these of S by firings of ϵ transitions.

The presented *meta graph* algorithm maintains a set of *meta states* G (the reachability meta graph). It is initialized with an agenda containing a single *meta state* which is the $\epsilon\text{Closure}$ of the initial marking M of the input Petri net Σ (line 1); furthermore, the function *FirableObs* (line 3) calculates from a *meta state* S , the firable observed transitions from S . A transition t is firable from a *meta state* S if and only if, in S , there is at least one marking enabling the firing of t . Finally, function *move* (line 9) takes a *meta state* and an enabled observed transition t in order to build the set of markings reached by the firing of t from markings of S . Obviously, the concerned markings are only these which enable the firing of t . Then, function $\epsilon\text{Closure}$ is applied on the resulting set of markings leading to a reached *meta state*.

In order to test the non constraining relation, the core

Algorithm IV.2 Inclusion of projected languages $(\langle \Sigma_v, M_v \rangle, \langle \Sigma_{env}, M_{env} \rangle)$

```
begin
1   $MS_{v_0} = \epsilon\text{Closure}(M_v)$ 
2   $MS_{env_0} = \epsilon\text{Closure}(M_{env})$ 
3  if ( $\text{firableObs}(MS_{v_0}) \not\subseteq \text{firableObs}(MS_{env_0})$ ) then
4      return false
5  endif
6  push  $\langle MS_{v_0}, \text{firableObs}(MS_{v_0}) \rangle$  onto  $stack_1$ ;
7  push  $\langle MS_{env_0}, \text{firableObs}(MS_{env_0}) \rangle$  onto  $stack_2$ ;
8  save  $\langle MS_{v_0}, MS_{env_0} \rangle$  in the synchronized product.
9  while  $stack_1$  is not empty do
10      $e_1$  the top element of  $stack_1$ 
11      $e_2$  the top element of  $stack_2$ 
12     if  $stack_1.\text{second}$  is not empty then
13          $t = stack_1.\text{second}.\text{begin}()$ 
14         erase  $t$  in  $stack_1.\text{second}$ 
15          $MS_{reached_v} := \epsilon\text{Closure}(\text{move}(e_1.\text{first}, t))$ 
16          $MS_{reached_{env}} := \epsilon\text{Closure}(\text{move}(e_2.\text{first}, t))$ 
17         if  $\text{firableObs}(MS_{reached_v})$ 
18              $\not\subseteq$ 
19              $\text{firableObs}(MS_{reached_{env}})$  then
20             return false
21         endif
22         if  $\langle MS_{reached_v}, MS_{reached_{env}} \rangle$ 
23             is not already saved in the synchronized product
24         then
25             push  $MS_{reached_v}$  onto  $stack_1$ 
26             push  $MS_{reached_{env}}$  onto  $stack_2$ 
27             save  $\langle MS_{reached_v}, MS_{reached_{env}} \rangle$ 
28             in the synchronized product.
29         endif
30     else
31         pop the top element of  $stack_1$ 
32         pop the top element of  $stack_2$ 
33     endif
34 endwhile
35 return true
end
```

algorithm given in IV.2, performs the language inclusion test given in property 3. For this, a synchronized product is built on fly during the meta graphs construction of Σ_v and Σ_{env} . In this case, the observed transitions are these shared between Σ_v and Σ_{env} , since we are interested in studying behaviors of these subnets over their interface. Moreover, we test that the reached *meta states*, $MS_{reached_v}$ and $MS_{reached_{env}}$ (of Σ_v and Σ_{env} respectively) match on the firable transitions of T_{int} , the observed transitions. This test takes place in lines 17-19.

V. EXPERIMENTAL RESULTS

The efficiency of our methods relies particularly on the implementation of the non constraining test. The underlying synchronized product is not built from state spaces but *meta state* automata, inducing a lower size w.r.t. the standard one. Moreover, BDD packages are used to concisely represent the meta states.

The table of Figure 3 highlights the gain of our methodology when testing the non constraining relation, for different numbers of philosophers. Each time, the property to be proved involves the same two transitions of one philosopher. Moreover, the selected property is

Fig. 3. Experimental results

Petri net	#markings	T_{obs}	A_v		Synchronized product			
			#meta states	#nb arcs	Resp	Nb it	Cp u tim	#markings (A_v, A_{env})
Philo ₂	22	GoEat ₁ +Rel ₁	10	18	YES	1	0.00	(16,13)
Philo ₃	100	GoEat ₁ +Rel ₁	10	23	YES	1	0.01	(16, 19)
Philo ₄	466	GoEat ₁ +Rel ₁	10	23	YES	1	0.02	(16, 49)
Philo ₅	2164	GoEat ₁ +Rel ₁	10	23	YES	1	0.03	(16,187)
Philo ₆	10.054 10 ³	GoEat ₁ +Rel ₁	10	23	YES	1	0.06	(16,8.29 10 ²)
Philo ₇	4.6708 10 ⁴	GoEat ₁ +Rel ₁	10	23	YES	1	0.09	(16, 3.81 10 ³)
Philo ₈	2.16994 10 ⁵	GoEat ₁ +Rel ₁	10	23	YES	1	0.17	(16, 1.76 10 ⁴)
Philo ₉	1.0081 10 ⁶	GoEat ₁ +Rel ₁	10	23	YES	1	0.34	(16, 8.20 10 ⁴)
Philo ₁₀	4.68338 10 ⁶	GoEat ₁ +Rel ₁	10	23	YES	1	0.55	(16, 3.810 10 ⁵)
Philo ₂₀	2.19341 10 ¹³	GoEat ₁ +Rel ₁	10	23	YES	1	18.74	(16, 2.078 10 ⁹)

true so as to force our tool to build the maximum structure.

Through the table, the determining criteria is obtained by comparing the size of the synchronized product between meta states graphs (see column $\#Markings(A_v, A_{env})$) against the size of the set of markings of the corresponding whole net to be checked (see column $\#Markings$).

One can note that the non constraining subnet is found right through the first iteration ($\#\Delta$) and that A_v , on which the model checking occurs, contains only 10 meta states. Moreover, from *philo₃* to *philo₂₀*, we obtain the same A_v , although the number of markings represented by the *meta states* increases. This can be explained by the fact that, in all these cases, the structure of Σ_f is the same.

VI. CONCLUSION

The efficiency of our approach has been tested from toy case studies which correspond to standard system paradigms, however the experimental results seems very promising due the obtained reduction in size. Other benchmarks are under progress.

Our next research perspectives consist in answering to complementary problems : in particular, how to improve the memory management of A_v ? in which conditions, other models than Petri nets can benefit from our approach? Is it interesting to start the verification process from a Σ_v larger than Σ_f ?

REFERENCES

- [1] Younes Souissi, "On Liveness Preservation by Composition of Nets via a Set of Places," *Advances in Petri Nets*, vol. 524, pp. 277–295, 1991.
- [2] C. Sibertin-Blanc, "A client-server protocol for composi-

tion of Petri nets," *ICATPN*, Chicago, USA, pp.377-396, 1993.

- [3] Y. Kesten, A. Pnueli, "Modularization and Abstraction: The keys to Practical Formal Verification," *MFCS*, Brno, Czech Republic, pp.54-71, 1998.
- [4] A. Valmari, "Compositional state space generation," *International Conference on Application an Theory of Petri Nets*, Paris, France, pp.43–62, 1990
- [5] A. Santone, "Compositionality for Improving Model Checking," *Formal Methods for Distributed System Development (FORTE)*, Genoa, Italy, pp.1010–1020, 2000.
- [6] M. Lies, C. Girault, "Liveness Preservation by synchronisation," *International Conference on Application an Theory of Petri Nets*, Osaka, Japan, pp.73–92, 1996.
- [7] V. Noord, "Treatment of Epsilon Moves in Subset Construction," *ACL Computational Linguistics*, vol. 26, no.1, pp.61–76, March 2000.