An Incremental and Modular Technique for Checking LTL\X Properties of Petri nets

Kais Klai[1], Laure Petrucci[1], and Michel Reniers[2]

[1] LIPN, CNRS UMR 7030
Université Paris 13
99 avenue Jean-Baptiste Clément
F-93430 Villetaneuse, France
{kais.klai,laure.petrucci}@lipn.univ-paris13.fr
[2] Design and Analysis of Systems (OAS)
Department of Mathematics and Computer Science
Technical University Eindhoven (TU/e)
P.O. Box 513, NL-5600 MB Eindhoven, The Netherlands
M.A.Reniers@tue.nl

Abstract. Model-checking is a powerful and widespread technique for the verification of finite state concurrent systems. However, the main hindrance for wider application of this technique is the well-known state explosion problem. Modular verification is a promising natural approach to tackle this problem. It is based on the "divide and conquer" principle and aims at deducing the properties of the system from those of its components analysed in isolation. Unfortunately, several issues make the use of modular verification techniques difficult in practice. First, deciding how to partition the system into components is not trivial and can have a significant impact on the resources needed for verification. Second, when model-checking a component in isolation, how should the environment of this component be described? In this paper, we address these problems in the framework of model-checking LTL\X action-based properties on Petri nets. We propose an incremental and modular verification approach where the system model is partitioned according to the actions occurring in the property to be verified and where the environment of a component is taken into account using the linear place invariants of the system.

1 Introduction

Model-checking is a powerful and widespread technique for the verification of finite state concurrent systems. Given a property and a model of the system, the model-checker performs an exhaustive exploration of the state space of the system to check the validity of the property. When the property is proved unsatisfied by the system, the model-checker supplies a *counterexample*, i.e., an execution scenario illustrating the violation of the property. However, the main hindrance for wider application of the model-checking approach to verify concurrent and distributed systems is the well-known state explosion problem. In fact, the size of the state space of systems grows exponentially with the number of their components. Numerous techniques have been proposed to tackle the state explosion problem in order to get a manageable state space. Among them, *on-the-fly* model-checking (e.g., [10, 5]) allows for generating only the "interesting" part of the model; *partial order reduction* (e.g., [1, 19]) is a reduction technique exploiting independence of some transitions in the system to discard unnecessary parts; *symbolic model-checking* (e.g., [7, 9, 6]) aims at checking the property on a compact representation of the system by using BDD (Binary Decision Diagram) techniques [2]. More related to this paper, *modular verification* (e.g., [20, 3, 13, 12]) is a promising natural approach which takes advantage of the modular design of concurrent and distributed systems. Using the "divide and conquer" principle, the system is broken down into components and each of these is analysed separately. Thus, the verification of the global system is downsized to the analysis of its individual components. This could reduce dramatically the complexity of the analysis. However, several issues make using modular verification difficult. First, deciding how to partition the system into components is not trivial and can have a significant impact on the resources needed for verification [4]. Second, when model-checking a component in isolation, a model of the environment interacting with the component often has to be introduced, so that the component is not completely free in its interaction with the environment. In [15], Mc Millan calls this problem *the environment problem.* Finally, once each component is specified with the abstraction of its environment, it is of utmost importance to prove that the decomposition characterises completely the properties of the whole system.

In this paper, we address these problems by supplying some heuristical but formal solutions. First, the global net N is viewed as the composition of n components $\langle N_1, \dots, N_n\rangle$ where N_1 is a subnet containing all the actions occurring in the property to be checked, and $\forall i, j = 1, \dots, n$, $i \neq j$, N_i and N_j are two subnets with disjoint sets of places. The choice of places and transitions within the components follows a particular scheme (see Section 3). Then, each subnet N_i is augmented with some additional places in order to abstract the environment i.e. $\{N_j \mid j = 1, \dots, n, j \neq i\}$ (see Section 4). The set of the abstraction places is formally determined by using the linear invariants of the global system. Based on these decomposition and abstraction steps, our modular verification approach of a LTL\X formula φ on the global net N can be summarised as follows: We first prove that once φ holds in all components (completed by their environment abstraction) analysed separately, one can check it on a reduced synchronised product of the components built in an incremental way. Then, if φ is unsatisfied by one component, a *non-constraining relation* is defined as a property allowing, when satisfied, to deduce that it is unsatisfied by the global net as well. The non-constraining relation is asymmetric and should be checked between two components. Its satisfaction makes the analysis of modules separately equivalent to a global analysis. In this paper, we present a modular algorithm to check this relation. When the non-constraining relation is unsatisfied, the partition $\langle N_1, \dots, N_n\rangle$ is refined by composing its first elements (N_1 and N_2) leading to a smaller partition which will be processed in the same way. In the worst case the property will be checked on a partition of size 1 (i.e. the whole net).

After recalling basic notions and notations related to Petri nets and the LTL logic in Section 2, Section 3 presents our decomposition scheme. Based on this decomposition, Section 4 shows how linear invariants can be used to complete a module with an abstraction of its environment. Section 5 discusses how a local counterexample is allowed by the environment of the corresponding component. This is achieved using the *non-constraining* relation which is checked in a modular way. Section 6 is dedicated to our incremental and modular verification approach. Section 7 is devoted to final discussion and comparison with related works on modular verification. Finally, concluding remarks and perspectives are presented in Section 8.

2 Preliminaries and notations

In this section, we recall some basic notions of Petri net theory and introduce some notations. We also recall the syntax and semantics of the LTL logic.

Vectors and matrices Let v be a vector or a matrix, then v^T denotes its transpose. So, if v, v' are two vectors then $v^T.v'$ corresponds to their scalar product. Let v be a vector of $\mathbb{N}^P$. The support of v is $||v|| = \{p \in P | \, v(p) > 0\}$.

Petri nets A Petri net is a tuple $N = \langle P, T, Pre, Post\rangle$ with disjoint sets P and T of places and transitions, and the backward and forward incidence matrices $Pre : P \times T \longrightarrow \mathbb{N}$ and $Post : P \times T \longrightarrow \mathbb{N}$. Given a transition t, $Pre(t)$ and $Post(t)$ denote the t-column of Pre and $Post$ respectively. The preset of a place p (resp. a transition t) is defined as ${}^\bullet p = \{t \in T \,|\, Post(p, t) > 0\}$ (resp. ${}^\bullet t = \{p \in P \,|\, Pre(p, t) > 0\}$), and its postset as $p^\bullet = \{t \in T \,|\, Pre(p, t) > 0\}$ (resp. $t^\bullet = \{p \in P \,|\, Post(p, t) > 0\}$). The preset (resp. postset) of a set X of nodes is given by the union of the presets (resp. postsets) of all nodes in X. ${}^\bullet X^\bullet$ denotes the union of the preset and the postset of X. Given a place p, 1_p denotes the vector of $\mathbb{N}^P$ where each element is zero except the element indexed by place p, which has value 1.

A marking of a net is a mapping $m : P \longrightarrow \mathbb{N}$. We call $\langle N, m_0\rangle$ a net with initial marking m_0. A marking m enables the transition t ($m\xrightarrow{t}$) iff $m(p) \geq Pre(p, t), \forall p \in P$. In this case the transition can occur, leading to the new marking m', given by: $m'(p) = m(p) - Pre(p, t) + Post(p, t), \forall p \in P$. This occurrence is denoted by $m\xrightarrow{t}m'$. If there exists a chain ($m_0\xrightarrow{t_1}m_1 \ldots \xrightarrow{t_n}m_n$), denoted by $m_0\xrightarrow{\sigma}m_n$, the sequence $\sigma = t_1 \ldots t_n$ is also called a computation or a firing sequence. We denote by T^* (resp. T^∞) the set of finite (resp. infinite) sequences of T. $T^\omega = T^* \cup T^\infty$ denotes the set of all sequences of T. The finite (resp. infinite) language of (N, m_0) is the set $L^*(\langle N, m_0\rangle) = \{\sigma \in T^* \,|\, m_0\xrightarrow{\sigma}\}$ (resp. $L^\infty(\langle N, m_0\rangle) = \{\sigma \in T^\infty \,|\, m_0\xrightarrow{\sigma}\}$) and $L^\omega(\langle N, m_0\rangle) = L^*(\langle N, m_0\rangle) \cup L^\infty(\langle N, m_0\rangle)$.

Subnets Let $N = \langle P, T, Pre, Post\rangle$ be a Petri net. $N' = \langle P', T', Pre', Post'\rangle$ is a subnet of N induced by (P', T'), $P' \subseteq P$ and $T' \subseteq T$, iff $\forall (p, t) \in P' \times T'$,

$Pre'(p,t) = Pre(p,t)$ and $Post'(p,t) = Post(p,t)$. If m is a marking of N then its projection on the places of N', denoted by $m_{\lfloor P'}$, is defined by $m'(p) = m(p), \forall p \in P'$. If σ is a computation of N, the projection of σ on a set of transitions $X \subseteq T$, denoted by $\sigma_{\lfloor X}$, is the sequence obtained by removing from σ all transitions not in X. The projection function is extended to sets of sequences (i.e., languages) as follows: $\forall \Gamma \subseteq T^\omega$, $\Gamma_{\lfloor X} = \{\sigma_{\lfloor X} — \sigma \in \Gamma\}$. Given a subnet N' of N, we also use the projection notations to denote by $m_{\lfloor N'}$ (resp $\sigma_{\lfloor N'}$) the restriction of marking m (resp. sequence σ) to places (resp. transitions) of N'.

Linear invariants Let v be a vector of $\mathbb{N}^P$, v is a positive linear invariant iff $v.W = 0$, where $W = Post - Pre$. If v is a positive linear invariant and $m \xrightarrow{\sigma} m'$ is a firing sequence, then $v^T.m' = v^T.m$.

Linear-time Temporal Logic (LTL) LTL formulae are defined by: $\varphi ::= \mathrm{a} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathbf{G}\ \varphi \mid \mathbf{F}\ \varphi \mid \varphi\ \mathbf{U}\ \varphi \mid \mathbf{X}\ \varphi$, where a is an action label, **G**, **F**, **U** and **X** denote the *always*, *eventually*, *until* and *next* operators respectively. A LTL formula is generally interpreted over labelled transition systems (e.g. a Petri net reachability graph). For a detailed description of LTL, refer to [16]. In this paper we deal with LTL\X (LTL minus the next operator) properties.

3 Decomposition scheme

In this section, we present a decomposition of a Petri net N according to some LTL\X formula φ and discuss its properties. Before giving a formal definition of the retained decomposition, we first define a more general decomposition. Here, we require that the composition of the components using transition fusion results in the original net, and that the components have no place in common.

Definition 1 (Decomposition). *Let $N = \langle P, T, Pre, Post\rangle$ and, for $1 \leq i \leq n$, $N_i = \langle P_i, T_i, Pre_i, Post_i\rangle$ be nets. Then $\langle N_1, \ldots, N_n\rangle$ is a decomposition of N iff the following criteria hold:*

- $P = \bigcup_{i=1}^n P_i$ *and* $P_i \cap P_j = \emptyset$, *for* $1 \leq i < j \leq n$;
- $T = \bigcup_{i=1}^n T_i$;
- $\forall i \in [1..n], \forall p \in P_i, \forall t \in T_i : Pre(p,t) = Pre_i(p,t)$;
- $\forall i \in [1..n], \forall p \in P_i, \forall t \in T_i : Post(p,t) = Post_i(p,t)$;
- $\forall i \in [1..n], \forall p \in P_i, {}^\bullet p(N_i) = {}^\bullet p(N)$.

Given a partition $\langle N_1, \ldots, N_n\rangle$ of a given Petri net N and two elements N_k and N_l of this partition, with $k \leq l$, we denote by $N_{(k,l)}$ the subnet obtained by the composition of all subnets $N_k, N_{k+1}, \ldots, N_l$. Such a composition leads to a new partition $\langle N_1, \ldots, N_{(k,l)}, \ldots, N_n\rangle$ which involves $n - l + k$ components. The following definition introduces the structure of the subnet $N_{(k,l)}$ for $1 \leq k \leq l$.

Definition 2. *Let $\langle N_1, \ldots, N_n\rangle$ be a decomposition of a net $N = \langle P, T, Pre, Post\rangle$. $\forall 1 \leq k \leq l \leq n$, $N_{(k,l)} = \langle P_{(k,l)}, T_{(k,l)}, Pre_{(k,l)}, Post_{(k,l)}\rangle$ is defined as follows:*

– $P_{(k,l)} = \bigcup_{i=k}^{l} P_i$;
– $T_{(k,l)} = \bigcup_{i=k}^{l} T_i$;
– $Pre_{(k,l)}(p,t) = Pre(p,t)$, *for all* $p \in P_{(k,l)}$ *and* $t \in T_{(k,l)}$;
– $Post_{(k,l)}(p,t) = Post(p,t)$, *for all* $p \in P_{(k,l)}$ *and* $t \in T_{(k,l)}$.

From now on, we denote by $\langle N_1, I, N_2\rangle$ the decomposition of a Petri net N into two subnets N_1 and N_2 with disjoint sets of places and that share the set of interface transitions I. For any subnet $N_{(k,l)}$ of N, the language of $N_{(k,l)}$ contains the language of N restricted to the transitions of $N_{(k,l)}$:

Proposition 1. *Let $\langle N_1, \ldots, N_n\rangle$ be a decomposition of a net N. $\forall 1 \leq k \leq l \leq n$ and for all markings m of N:*

$$L^{\omega}(\langle N_{(k,l)}, m_{\lfloor N_{(k,l)}}\rangle) \supseteq L^{\omega}(\langle N, m\rangle)_{\lfloor N_{(k,l)}}$$

Definition 1 allows for many different decompositions of a Petri net into n components. In the following, we define a specific decomposition which is guided by the knowledge of the transitions occurring in the formula φ to be checked on N.

Definition 3 (Decomposition according to a formula φ).
Let $N = \langle P, T, Pre, Post\rangle$ be a Petri net and let φ be a LTL\X formula involving a non-empty subset of transitions T_φ. Then $N_{T_\varphi} = \langle N_1, \ldots, N_n\rangle$ is the decomposition of N according to φ iff N_{T_φ} is a decomposition of N such that for all $1 \leq i < n$, the nets $N_i = \langle P_i, T_i, Pre_i, Post_i\rangle$ satisfy the following criteria:

– $P_1 = {}^\bullet T_\varphi{}^\bullet$ *and* $P_{i+1} = {}^\bullet T_i{}^\bullet \setminus \bigcup_{j=1}^{i} P_j$;
– $T_i = {}^\bullet P_i{}^\bullet$.

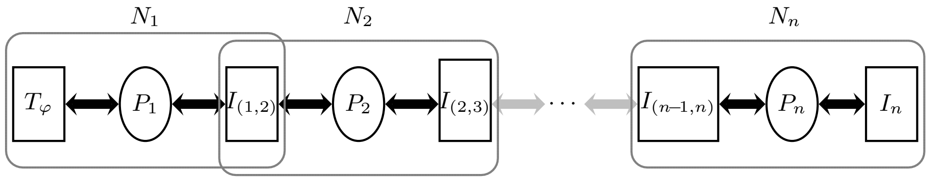

Fig. 1. Iterative decomposition scheme

Figure 1 illustrates the decomposition scheme of Definition 3. Note that this decomposition is such that each component interacts with at most two other ones (i.e. they are positioned linearly) and that the leftmost component in this scheme contains the transitions T_φ that occur in the formula φ to be checked. One could consider the subnet containing T_φ only as the first leftmost subnet. However, such a choice would allow all possible sequences on T_φ (i.e., T_φ^ω) and hence needs to be restricted further. This is ensured by completing the subnet with the places connected to T_φ in the original net. Then, the transitions connected to these places are also added so that the subnet obtained still satisfies Definition 1. Subnets N_i and N_{i+1} share a subset of transitions which we call $I_{(i,i+1)}$.

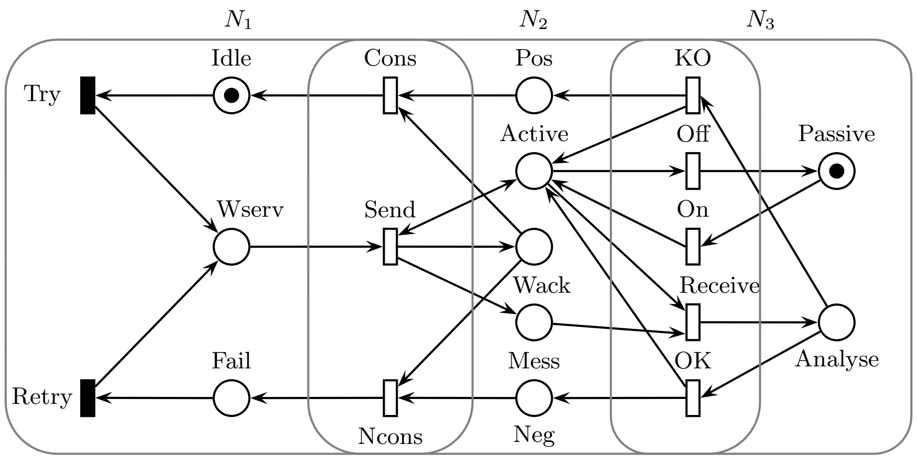

Fig. 2. A decomposable Petri net

Example 1. Figure 2 illustrates an example of a decomposable Petri net model of a simplified *client-server* system. The server switches between states Passive and Active on reception of On and Off signals respectively. The client is initially Idle. When it wants to send a message, it waits for the server to be active (place Wserv). Then, it sends its message and waits for an acknowledgement (place Wack). In case of a positive acknowledgement, it becomes Idle again. Otherwise, it tries to retransmit the message (place Fail). On reception of a message, the server analyses it and sends an acknowledgement (place Analyse).

The considered set of transitions T_φ is $\{\mathrm{Try}, \mathrm{Retry}\}$ (the black transitions in Figure 2). Using these two transitions, one can express several properties characterising the communication between the client and the server. For instance the formula $\mathbf{G}(Retry \Rightarrow ((\neg Retry)\mathbf{U}Try))$ states that each message sent can be retransmitted at most once. In this decomposition of the client-server model, subnet N_1 is unbounded since transition Cons can be executed infinitely often, thus flooding place Idle with tokens. A correct modular approach should analyse a component of the system completed by an abstraction of its environment. In the next subsection, we show how to exploit the system invariants in order to automatically construct such an abstraction.

4 Abstraction of the environment using linear invariants

Linear invariants of a Petri net correspond to a safety property of the system (see e.g. [8, 11]). They are computed by finding a generative family of positive solutions of a linear equation system. Even though the worst case time complexity of this computation is not polynomial, in practice the algorithm behaves efficiently w.r.t. the reachability graph construction.

Here, we propose to use linear invariants as a witness of the synchronisation between the two subnets of the net N with decomposition $\langle N_1, I, N_2\rangle$. Let V_N be the set of positive linear invariants of net N; these are called the global invariants. With an invariant $v \in V_N$, we associate two places $a_1^{(v)}, a_2^{(v)}$ which are added to N_1 and N_2 respectively. The current marking of the added places summarises the information given by the corresponding positive linear invariant v. The net obtained by adding an abstraction place for each invariant from a set $V \subseteq V_N$ is called the *component subnet* for V and denoted from now on by $\widehat{N}_j$.

Definition 4 (*Component subnet*). *Let $\langle N_1, I, N_2\rangle$ be a decomposition of a net N and let $V \subseteq V_N$. The component subnet related to $N_i = \langle P_i, T_i, Pre_i, Post_i\rangle$ generated from the set of invariants V is $\widehat{N}_i = \langle \widehat{P}_i, \widehat{T}_i, \widehat{Pre}_i, \widehat{Post}_i\rangle$ such that:*

- *$\widehat{T}_i = T_i$;*
- *$\widehat{P}_i = P_i \cup A_j$ (where $i \in \{1,2\}$ and $j \neq i$), with $A_j = \{a_j^{(v)} | v \in V\}$ the set of abstraction places;*
- *for all $p \in \widehat{P}_i$ and $t \in \widehat{T}_i$, $\widehat{Pre}_i(p,t) = Pre(t)^T.\Phi(p)$ and $\widehat{Post}_i(p,t) = Post(t)^T.\Phi(p)$, where the mapping Φ from $P \cup A_1 \cup A_2$ to $\mathbb{N}^{P \cup A_1 \cup A_2}$ is defined by $\Phi(p) = 1_p$, for $p \in P$, and $\Phi(a_j^{(v)}) = \sum_{p \in P_j} v(p).1_p$ for $a_j^{(v)} \in A_j$.*

The mapping from a *global* marking to markings of the component subnets is now defined to determine the initial marking of places representing the invariants.

Definition 5. *Let $\langle N_1, I, N_2\rangle$ be the decomposition of a net N and let $\widehat{N}_i$ ($i = 1, 2$) be the induced component subnets. For each marking m of N, Φ_i the projection mapping on $\widehat{N}_i$ is defined by: $\Phi_i(m)(p) = m^T.\Phi(p)$ for all $p \in \widehat{P}_i$.*

For transitions of the component subnet, all computations of the original marked net are also computations of the component subnet.

Proposition 2. *Let $\langle N_1, T_I, N_2\rangle$ be a decomposition of a net N. Then, for all markings m of N, $L^\omega(\langle \widehat{N}_i, \Phi_i(m)\rangle) \supseteq L^\omega(\langle N, m\rangle)_{\lfloor \widehat{N}_i}$.*

Note that it is also the case that $L^\omega(\langle N_i, m_{\lfloor N_i}\rangle) \supseteq L^\omega(\langle \widehat{N}_i, \Phi_i(m)\rangle)$: the addition of the places representing the invariant(s) is restricting the computations. The more invariants, the more precise the approximation of the global net behaviour.

As a consequence of proposition 2, we can use the invariants of the original net to obtain component subnets that hopefully disallow all counterexamples. Which set of invariants should be used for constructing the component subnets is a difficult question which will not be answered in this paper. However, we note that considering invariants that only have a support in one of the components is useless since they typically lead to a disconnected place in the other component. As a heuristic we propose to use all invariants that have support in both components, thus providing most information about the environment. Hence, we compute the component subnets for the decomposition $\langle N_{(1,i)}, I_{(i,i+1)}, N_{(i+1,n)}\rangle$ using *all* invariants that have support in both $N_{(1,i)}$ and $N_{(i+1,n)}$.

Example 2. The generative family of invariants of the model of Figure 2 is:

1. $v = \mathrm{Idle} + \mathrm{Fail} + \mathrm{Wserv} + \mathrm{Wack}$
2. $v' = \mathrm{Idle} + \mathrm{Fail} + \mathrm{Wserv} + \mathrm{Mess} + \mathrm{Analyse} + \mathrm{Pos} + \mathrm{Neg}$
3. $v'' = \mathrm{Active} + \mathrm{Passive} + \mathrm{Analyse}$

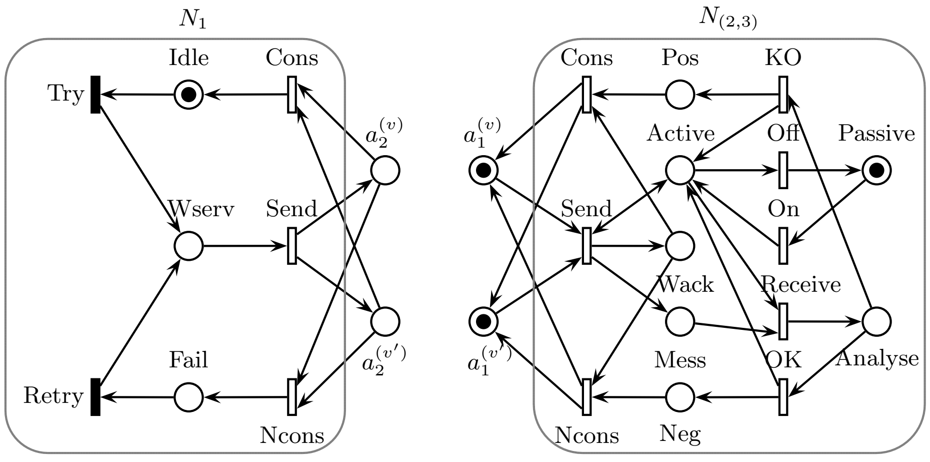

Fig. 3. The component subnets $\widehat{N}_1$ and $\widehat{N}_{(2,3)}$

The first two invariants cover both subnets while the third one is local to subnet $N_{(2,3)}$. Figure 3 illustrates the client *component subnet* obtained by using the first two invariants. The *component subnet* corresponding to the server can be obtained in a similar way. Note that the original client subnet has been enlarged with abstraction places $a_2^{(v)}$ and $a_2^{(v')}$. Let us explain for instance the underlying meaning of the abstraction place $a_2^{(v')}$. Since $\Phi(a_2^{(v')}) = 1_{\text{Mess}} + 1_{\text{Analyse}} + 1_{\text{Pos}} + 1_{\text{Neg}}$, this place contains the sum of tokens of these four places (i.e., 0). As Mess is an output place of the transition Send and the three other ones are not, $Post(a_2^{(v')}, \text{Send}) = 1$. The other arcs are obtained in a similar way.

Up to now, we have proposed a decomposition scheme based on the LTL\X formula to be checked and exploited the place invariants in order to abstract the environment of a given component while keeping some information about the interaction around the interface between two parts of the system. This scheme will be used in the following to deal with model-checking. Given a component of the system completed with an abstraction of its environment (a component subnet) how useful is a separate analysis of this component w.r.t. a global analysis of the whole net? The next section is devoted to characterising and checking whether a counterexample found locally in a component is allowed by its environment.

5 Checking the validity of local counterexamples

In order to check the validity of a counterexample found locally for a component, we introduce a sufficient condition, namely the *non-constraining relation.*

5.1 The non-constraining relation

The *non-constraining relation* is an asymmetric property to be checked between two given marked *component subnets* obtained from a net decomposition: $\langle N_2, m_2\rangle$ does not constrain $\langle N_1, m_1\rangle$ if for any firing sequence enabled from

$\langle N_1, m_1\rangle$, there exists a firing sequence enabled from $\langle N_2, m_2\rangle$, both having the same projection on the shared transitions. Then, we prove that the firing sequences enabled in the non-constrained component exactly represent the firing sequences of the global net, up to the projection on the component transitions.

Definition 6 (Non-constraining relation). *Let $\langle N_1, m_1\rangle$ and $\langle N_2, m_2\rangle$ be two marked nets with disjoint sets of places. Then, $\langle N_2, m_2\rangle$ does not constrain $\langle N_1, m_1\rangle$ iff $L^\omega(\langle N_1, m_1\rangle)_{\lfloor N_2} \subseteq L^\omega(\langle N_2, m_2\rangle)_{\lfloor N_1}$.*

Expressed as an inclusion between two projected languages, the non-constraining relation can be considered as a strong condition characterising a complete freeness of the involved component w.r.t. its interface with the environment. A naive partition of the global net into components makes this relation quite often unsatisfied. However, using abstraction places, the freeness of a given component on the interface transitions is reduced and its communication behaviour is finely approximated. For instance, in Figure 3 both component subnets have the same projected language on the interface transitions, i.e. $Send.(Cons + Ncons)^\omega$.

Proposition 3. *Let $\langle N_1, I, N_2\rangle$ be a decomposition of a net N and let m be a marking of N. If $\langle \widehat{N}_2, \Phi_2(m)\rangle$ does not constrain $\langle \widehat{N}_1, \Phi_1(m)\rangle$ then the following assertion holds: $L^\omega(\langle \widehat{N}_1, \Phi_1(m)\rangle) \subseteq L^\omega(\langle N, m\rangle)_{\lfloor \widehat{N}_1}$.*

Note that the non-constraining relation is a sufficient but not necessary condition for deducing the validity of a counterexample. It ensures that all possible local counterexamples are valid. This approach could be refined so that each representative of a set of counterexamples is checked separately.

A direct consequence of Proposition 3 is that, if $\langle \widehat{N}_2, \Phi_2(m)\rangle$ does not constrain $\langle \widehat{N}_1, \Phi_1(m)\rangle$, one can deduce that a given LTL\X formula φ does not hold in the global net N as soon as it is proved unsatisfied by $\langle \widehat{N}_1, \Phi_1(m)\rangle$.

Proposition 4. *Let $\langle N_1, I, N_2\rangle$ be a decomposition of a net N and m a marking of N. Let φ be an LTL\X formula such that the involved actions belong to N_1. If $\langle \widehat{N}_2, \Phi_2(m)\rangle$ does not constrain $\langle \widehat{N}_1, \Phi_1(m)\rangle$ then the following assertion holds:*

$$\langle \widehat{N}_1, \Phi_1(m)\rangle \not\models \varphi \implies \langle N, m\rangle \not\models \varphi$$

The non-constraining relation is defined as an inclusion between languages. Checking such a property represents the main difficulty of our approach. A naive test of this relation would result in building the synchronised product of the reachability graphs of the component subnets, which could drastically limit the applicability of our method. Thus, the remaining part of this section will be devoted to reducing the complexity of checking the non-constraining relation.

5.2 Reduction of the non-constraining relation test

Given two nets $\langle N_1, m_1\rangle$ and $\langle N_2, m_2\rangle$ with disjoint sets of places, the idea is to insert a new net $\langle N_3, m_3\rangle$ in the non-constraining relation checking process so that the following implication holds: if $\langle N_2, m_2\rangle$ does not constrain $\langle N_3, m_3\rangle$, then $\langle N_2, m_2\rangle$ does not constrain $\langle N_1, m_1\rangle$.

Obviously, from the point of view of efficiency, this would reduce the complexity of the non-constraining relation check if and only if checking whether $\langle N_2, m_2\rangle$ does not constrain $\langle N_3, m_3\rangle$ is less expensive than checking whether it does not constrain $\langle N_1, m_1\rangle$. In the following proposition, we first present the general context of this reduction by giving the minimal conditions for N_3 so that the above implication holds. Then, based on our decomposition scheme, we define the component subnet that will play the role of N_3 and which guarantees the reduction of the complexity of the non-constraining relation check.

Proposition 5. *Let N_1, N_2 and N_3 be nets with sets of transitions T_1, T_2 and T_3 respectively, such that $T_1 \cap T_2 \subseteq T_3$. Let m_1, m_2 and m_3 be markings for N_1, N_2 and N_3, respectively. If $L^\omega(\langle N_3, m_3\rangle)_{\lfloor N_1} \supseteq L^\omega(\langle N_1, m_1\rangle)_{\lfloor N_3}$, then $\langle N_2, m_2\rangle$ is non-constraining for $\langle N_3, m_3\rangle \Rightarrow \langle N_2, m_2\rangle$ is non-constraining for $\langle N_1, m_1\rangle$.*

Now, using the same abstraction principle, one can abstract both component subnets. This leads to what we call the *interface component subnet*, which allows for representing the language of the global net compactly, up to a projection on the interface. It is obtained by connecting the interface transitions to the abstraction places of both components. This structure is used in the next section in order to check efficiently the non-constraining relation.

Definition 7 (*Interface component subnet*). *Let $\langle N_1, I, N_2\rangle$ be the decomposition of a net N and let $\widehat{N}_i$ $(i = 1, 2)$ be the induced component subnets for a set of invariants $V \subseteq V_N$. The* interface component subnet *related to this decomposition is $\widehat{I} = \langle \widehat{P}, \widehat{T}, \widehat{Pre}, \widehat{Post}\rangle$ such that, for $i, j \in \{1, 2\}$, $i \neq j$:*

- *$\widehat{T} = I$;*
- *$\widehat{P} = A_1 \cup A_2$, with $A_i = \{a_i^{(v)} | v \in V\}$ the set of abstraction places of $\widehat{N}_i$;*
- *for all $a \in A_i$ and $t \in \widehat{T}$, $\widehat{Pre}(a, t) = \widehat{Pre}_j(a, t)$ and $\widehat{Post}(a, t) = \widehat{Post}_j(a, t)$.*

For a marking m of N, $\widehat{m}(p) = \Phi_1(m)(p) + \Phi_2(m)(p)$ for all $p \in \widehat{P}$.

Example 3. Figure 4 represents the interface component subnets involved in the decomposition of the net in Figure 2 associated with their initial markings.

Proposition 6. *Let $\langle N_1, I, N_2\rangle$ be a decomposition of a net N. Let $\widehat{N}_i$ $(i = 1, 2)$ and $\widehat{I}$ be the induced (interface) component subnets for a set of invariants $V \subseteq V_N$. Let m be a marking of N. Then: $\langle \widehat{N}_2, \Phi_2(m)\rangle$ is non-constraining for $\langle \widehat{I}, \widehat{m}\rangle$ $\Rightarrow \langle \widehat{N}_2, \Phi_2(m)\rangle$ is non-constraining for $\langle \widehat{N}_1, \Phi_1(m)\rangle$.*

This proposition is exploited in order to restrain the test of the non-constraining relation of $\widehat{N}_{(i+1,n)}$ w.r.t. $\widehat{N}_{(1,i)}$, to a lighter relation between $\widehat{N}_{(i+1,n)}$ and the interface component subnet $\widehat{I}_{(i,i+1)}$. One can apply the same principle in an iterative way to deduce the following implication: if $\widehat{N}_{(j,n)}$ is non-constraining for $\widehat{I}_{(j-1,j)}$ for all $i + 1 \leq j \leq n$, then $\widehat{N}_{(i+1,n)}$ is non-constraining for $\widehat{N}_{(1,i)}$.

This can drastically reduce the complexity of checking the non-constraining relation, since it is modularly checked on very small components, one at a time.

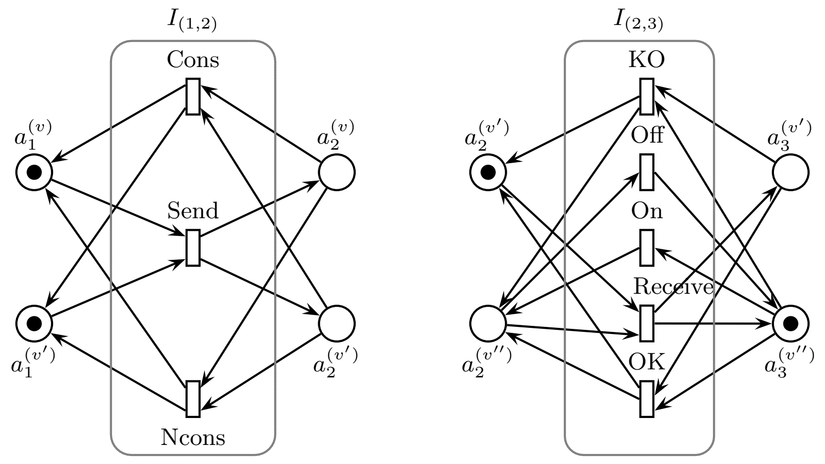

Fig. 4. The Client-Server interface component subnets

6 An incremental and modular model-checker

In this section, we use the decomposition and abstraction techniques presented in the previous sections to give an incremental modular technique for establishing the validity of a formula φ on a net N. Algorithm 1 illustrates the technique. The net is supposed to be decomposed as described in Definition 3: $N = \langle N_1, \ldots, N_n \rangle$. The algorithm verifies the validity of formula φ on the component subnet $\widehat{N}_{(1,j)}$ starting from $j = 1$ (first loop). To establish $\widehat{N}_{(1,j)} \models \varphi$, one can use any standard model checker for LTL\X formulas on Petri nets. In case the formula φ holds in this component subnet (line 4), the validity of φ w.r.t. the other component subnet $N_{(j+1,n)}$ is checked. This is done modularly on $\widehat{N}_i$ for all $i, j < i \leq n$ (lines 6–8). Since the component subnets $\widehat{N}_i$ do not contain any transition of T_φ, $\widehat{N}_i \models \varphi$ can be established only using reachability, deadlock and divergence information; for example using symbolic observation graphs as in [9]. When the property φ is proved satisfied by $\widehat{N}_{(1,j)}$ and all $\widehat{N}_i$ (for $i > j$), analysed separately, the property is checked on a reduced synchronised product (lines 9–10) using algorithm 2. This task is discussed and detailed in section 6.1.

If one of the verifications of φ fails, the next phase of the algorithm consists in checking whether the component subnets not involved in the previous verification process allow the counterexample to occur. This step is ensured by checking the non-constraining relation between the component in which φ is unsatisfied and these components considered as its environment. Algorithm 3 performs this task in a modular way, as described in section 6.2.

At this point of the algorithm, we know that the property does not hold either in $\widehat{N}_{(1,j)}$ nor in $\widehat{N}_k$ for some $j < k \leq n$. In the first case, the environment is the right-hand side partition $\langle \widehat{N}_{j+1}, \ldots, \widehat{N}_n \rangle$ and the non-constraining relation step is invoked once (line 19). While, in the second case, the environment is the left-

Algorithm 1: Checking φ on the components of a decomposition

```
Require: $\varphi, \langle N_1, \ldots, N_n \rangle$
Ensure: Check $\varphi$ on $\langle N_1, \ldots, N_n \rangle$
 1: int i,j;
 2: j=1;
 3: while j¡n do
 4:   if $\widehat{N}_{1,j}$ satisfies $\varphi$ then
 5:     i=j+1;
 6:     while $\widehat{N}_i$ satisfies $\varphi$ do
 7:       i=i+1;
 8:     end while
 9:     if i ¿ n then
10:       Check property $\varphi$ on a
          reduced synchronised product
11:     else
12:       if $\neg(\langle \widehat{N}_{i-1}, \ldots, \widehat{N}_1 \rangle$ constrains
          $\widehat{I}_{(i-1,i)})$ and $\neg(\langle \widehat{N}_{i+1}, \ldots, \widehat{N}_n \rangle$
          constrains $\widehat{I}_{(i,i+1)})$ then
13:         return false; // $N \not\models \varphi$
14:       else
15:         j=j+1;
16:       end if
17:     end if
18:   else
19:     if $\neg(\langle \widehat{N}_{j+1}, \ldots, \widehat{N}_n \rangle$ constrains
        $\widehat{I}_{(j,j+1)})$ then
20:       return false; // $N \not\models \varphi$
21:     else
22:       j=j+1;
23:     end if
24:   end if
25: end while
26: return true; // $N \models \varphi$
```

hand side partition $\langle \widehat{N}_1, \ldots, \widehat{N}_{i-1} \rangle$ and the right-hand side one $\langle \widehat{N}_{i+1}, \ldots, \widehat{N}_n \rangle$, and both have to be non-constraining for $\widehat{N}_i$. Thus the non-constraining relation is checked at most twice, once for each part of the environment (line 12). If the non-constraining relation step is successful, the counterexample is allowed by the corresponding environment and it is also allowed by the net as a whole. Thus, the invalidity of φ can be deduced: $N \not\models \varphi$. If the counterexample turns out not to be allowed by these component subnets, the verification process is started again, but a larger component subnet $\widehat{N}_{(1,j+1)}$ is then used (lines 15 and 22).

Two parts of algorithm 1 are not described yet: how to check the property on a reduced synchronised product (line 10) and how to check the non-constraining relation (lines 12 and 19). This is the issue of the following subsection.

6.1 Checking a property on a reduced synchronised product

Given a Petri net N and its decomposition in n component subnets $\langle \widehat{N}_1, \ldots, \widehat{N}_n \rangle$, the fact that an LTL\X property φ holds in each component $\widehat{N}_i$ (for $i = 1 \ldots n$) is not sufficient to deduce that φ holds in N as well. However, it helps to deduce that the possible invalidity of φ in N comes necessarily from the interaction between the different components. Hence we need to focus on the behaviour of these components around the interface and abstract the local behaviours since they have been proved satisfying the property. The symbolic observation graph (SOG) technique [9] is particularly well-suited for that purpose. Indeed, a SOG is a graph, built according to a subset of observed actions *Obs* where nodes are sets of states connected to one another by unobserved actions and arcs are exclusively labeled with action from *Obs*. Checking a LTL\X property on this graph is equivalent to checking the property over the original reachability graph. The size of the SOG is as small as the number of actions involved in the formula to be checked. In general its size is negligible w.r.t. the size of the original graph.

Algorithm 2: Checking φ on a reduced synchronised product

```
Require: $\langle N_1, \ldots, N_n \rangle, \varphi$
Ensure: check $\varphi$ over $N = \langle N_1, \ldots, N_n \rangle$
 1: int k;
 2: j=2
 3: while j¡n do
 4:     Build $SOG$, the symbolic observation graph of $\widehat{N}_{(1,j)}$
 5:     if $SOG \not\models \varphi$ then
 6:         if $\neg(\langle \widehat{N}_{j+1}, \ldots, \widehat{N}_n \rangle$ constrains $\widehat{I}_{(j,j+1)})$ then
 7:             return false; // $N \not\models \varphi$
 8:         end if
 9:     end if
10:     j=j+1
11: end while
12: return true; // $N \models \varphi$
```

Algorithm 2 uses this technique in an incremental way to check the property by exploiting our decomposition and abstraction schemes. Starting from the first component $\widehat{N}_{(1,1)}$, the property is checked iteratively on the SOG of $\widehat{N}_{(1,j)}$, for $j = 2, \ldots, n$, obtained by composing the component subnets $\widehat{N}_{(1,j-1)}$ and $\widehat{N}_j$ (lines 4–5). The model checking is performed on an incremented component subnet (line 10) and the associated SOG in two cases: the property holds in all the previous iterations, the property does not hold and the non-constraining property is unsatisfied. In the worst case, the property will be checked on the whole net. As soon as the non-constraining relation is proved satisfied, the algorithm returns *false* (the property does not hold in the global net).

6.2 The non-constraining checking algorithm

In [9], the authors propose an algorithm for checking the non-constraining relation between two subnets based on the synchronisation of their symbolic observation graphs. Here, we follow the same principle and propose a modular way for checking such a relation so that the global result is deduced from several tests performed on reduced subnets. Checking the non-constraining relation between two decomposed Petri nets is, in turn, done iteratively as described in Algorithm 3. The parameters of this step are a partition $\langle N_1, \ldots, N_n \rangle$ and a subnet I which is supposed to be adjacent to N_1. This hypothesis explains the fact that the partition $\langle \widehat{N}_{i-1}, \ldots, \widehat{N}_1 \rangle$ at line 12 of Algorithm 1 is used instead of $\langle \widehat{N}_1, \ldots, \widehat{N}_{i-1} \rangle$. Hence, the goal is to check whether the subnet induced by the partition constrains I. This task is done iteratively (lines 4–9) starting from the right-hand side of the partition and going left building towards the non-constraining of I. The correctness of this algorithm follows from Proposition 6.

Algorithm 3: Checking the non-constraining relation

```
Require: $\langle N_1, \ldots, N_n \rangle, I$
Ensure: check whether$\langle N_1, \ldots, N_n \rangle$ is nont constraining for $I$
 1: int k;
 2: $I_{(0,1)} = I$;
 3: k=n;
 4: while k¿1 do
 5:   if $N_k$ is constraining for $\widehat{I}_{(k-1,k)}$ then
 6:     return false; //the non-constraining relation is unsatisfied
 7:   end if
 8:   k=k-1;
 9: end while
10: return true; //the non-constraining relation holds
```

7 Discussion and related work

Several techniques have been proposed to push further the use of modularity in model-checking concurrent systems. As far as the verification is concerned, taking benefit from the structural composition of Petri nets is known to be a hard problem. Mainly, structural approaches aim at preserving some basic properties, such as liveness and boundedness, by composition of Petri nets (e.g. [12, 18, 17]). More general or behavioural approaches like [3, 13] deal with the minimisation of the reachable state space of each module by hiding the internal moves, before the synchronisation of modules. Reachability analysis has been proved to be effective on the resulting structure in [14] and the method has been extended to operate the model-checking of LTL\X formulae. However, experimental results show that this technique is efficient for some models, but for others the combinatorial explosion still occurs.
A common limit of existing modular approaches is that the components of the system are supposed to be known *a priori*. Even though the decomposition presented here is rather simple, having an adequate structuration into components is essential for the applicability of modular verification techniques.

In structural approaches, rather restrictive conditions are forced, thus reducing drastically the applicability to concrete systems while the synchronised product between components state graphs is quite often unavoidable in behavioral modular approaches.

Regarding the existing modular verification approaches, the contribution of this paper can be summarised in three points. First, our decomposition scheme is general and no restriction on the structure of the model is imposed. Second, we present an original formal way to combine decomposition and invariant-based abstraction of the system. Finally, we propose a modular algorithm for checking the equivalence between local and global verification. The approach we present here improves and generalises the work presented in [12]. The main improvements can be summarised in the three following points: First, contrary to [12], the decomposition scheme is not supposed to be known *a priori*. Indeed, even if the system can be decomposed intuitively into components, the decomposition obtained is not guaranteed to be suitable for the model-checking process. In this paper, we proposed to take advantage of the knowledge of the actions involved in the property to be verified in order to define a set of possible decompositions and combine such decompositions with the invariant-based abstraction. Second, in [12] the class of properties that can be handled by the method is not clearly identified. The authors speak about infinite observed sequences but it is not easy to say whether a given property depends on this kind of sequences only. Here, we extend the approach to LTL\X properties. Finally, we proposed a complete and self-contained modular verification approach for the LTL\X logic.

8 Conclusion

In this paper we addressed the modular verification of LTL\X properties over finite systems described as Petri nets. Our algorithm for such a modular verification aims at verifying a formula on a part of the system only. To achieve this, a subnet containing actions occurring in the property is completed by an abstraction of its environment and incrementally refined by including more and more details from the original system until the property can be proved either true or false. We exploit the structure of the system to both decompose the model w.r.t. the property to be checked and to compute the abstraction of the obtained components. The *non-constraining* relation is used to establish whether or not the counterexamples that might result from the local verification are globally allowed by the system. Some of the non-constraining relations that need to be established can themselves be checked iteratively.A tool implementing these algorithms is currently under development. It will provide experimental data fro testing on the efficiency of our algorithm and on some of the heuristics incorporated in it such as the decision to use all place invariants of the system. An interesting perspective of this work would be the refinement of the non-constraining relation. In fact, instead of checking the inclusion of two components projected languages, one could check whether the projection of a specific counterexample on the interface transitions is allowed by the environment component.

References

1. Girish Bhat and Doron Peled. Adding partial orders to linear temporal logic. In *Int. Conf. on Concurrency Theory (CONCUR)*, volume 1243 of *Lecture Notes in Computer Science*, pages 119–134. Springer, 1997.
2. Randal E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
3. Søren Christensen and Laure Petrucci. Modular analysis of Petri nets. *Computer Journal*, 43(3):224–242, 2000.
4. Jamieson Cobleigh, Dimitra Giannakopoulou, and Corina Pasareanu. Learning assumptions for compositional verification. In *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 2619 of *Lecture Notes in Computer Science*. Springer, 2003.

5. Jean-Michel Couvreur. On-the-fly verification of linear temporal logic. In *World Congress on Formal Methods (FM)*, volume 1709 of *Lecture Notes in Computer Science*, pages 253–271. Springer, 1999.
6. Jean-Michel Couvreur. A bdd-like implementation of an automata package. In *Int. Conf. on Implementation and Application of Automata (CIAA)*, volume 3317 of *Lecture Notes in Computer Science*, pages 310–311. Springer, 2004.
7. Jaco Geldenhuys and Antti Valmari. Techniques for smaller intermediary bdds. In *Int. Conf. on Concurrency Theory (CONCUR)*, volume 2154 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2001.
8. Claude Girault and Rdiger Valk. *Petri Nets for Systems Engineering — A Guide to Modeling, Verification, and Applications*. Springer, 2003.
9. Serge Haddad, Jean-Michel Ilié, and Kaïs Klai. Design and evaluation of a symbolic and abstraction-based model checker. In *Int. Conf. on Automated Technology for Verification and Analysis (ATVA)*, volume 3299 of *Lecture Notes in Computer Science*. Springer, 2004.
10. Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. A space-efficient on-the-fly algorithm for real-time model checking. In *Int. Conf. on Concurrency Theory (CONCUR)*, volume 1119 of *Lecture Notes in Computer Science*, pages 514–529. Springer, 1996.
11. Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Springer, 1997. Three Volumes.
12. Kaïs Klai, Serge Haddad, and Jean-Michel Ilié. Modular verification of Petri nets properties: A structure-based approach. In *Int. Conf. on Formal Techniques for Networked and Distributed Systems (FORTE)*, volume 3731 of *Lecture Notes in Computer Science*, pages 189–203. Springer, 2005.
13. Charles Lakos and Laure Petrucci. Modular analysis of systems composed of semi-autonomous subsystems. In *Int. Conf. on Application of Concurrency to System Design (ACSD)*, pages 185–194. IEEE Comp. Soc. Press, 2004.
14. Timo Latvala and Marko Mkel. LTL model-checking for modular Petri nets. In *Int. Conf. on Application and Theory of Petri Nets (ATPN)*, volume 3099 of *Lecture Notes in Computer Science*, pages 298–311. Springer, 2004.
15. Kenneth L. McMillan, Shaz Qadeer, and James B. Saxe. Induction in compositional model checking. In *Int. Conf. on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 312–327. Springer, 2000.
16. Amir Pnueli. Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends. In *Current Trends in Concurrency*, volume 224 of *Lecture Notes in Computer Science*, pages 510–584. Springer, 1986.
17. Christophe Sibertin-Blanc. A client-server protocol for composition of Petri nets. In *Int. Conf. on Application and Theory of Petri Nets (ATPN)*, volume 691 of *Lecture Notes in Computer Science*. Springer, June 1993.
18. Younès Souissi and Gérard Memmi. Compositions of nets via a communication medium. In G. Rozenberg, editor, *Advances in Petri Nets*, volume 483 of *Lecture Notes in Computer Science*, pages 457–470. Springer, 1991.
19. Antti Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1992.
20. Antti Valmari. Composition and abstraction. In *MOVEP*, volume 2067 of *Lecture Notes in Computer Science*, pages 58–98. Springer, 2000.