Checking partial-order properties of vector addition systems with states

Florent Avellaneda & Rémi Morin
Laboratoire d'Informatique Fondamentale de Marseille — CNRS, UMR 6166 — Aix-Marseille Université
163, avenue de Luminy, Case 901, F-13288 Marseille Cedex 9, France
Email: florent.avellaneda,remi.morin@lif.univ-mrs.fr

***Abstract*—Message Sequence Graphs (MSGs) form a popular model often used for the documentation of telecommunication protocols. They consist of typical scenarios of message exchanges depicted as partial-orders of events that lead from one control state to another. On the other hand Petri nets are a well-known formalism for distributed or parallel systems based on the notion of token game. Both approaches profit by a visual presentation and are the subject of numerous formal verification techniques and tools.**

In this paper we investigate a formalism which provides MSGs with the notion of token game and extends Petri nets with both control states and partial orders. Providing Petri nets with control states corresponds precisely to the model of Vector Addition Systems with States (VASSs). Thus we need to define first a partial-order semantics for VASSs which adopts the basic features of communication scenarios. To do so we extend simply the classical process semantics of Petri nets. We obtain a formal model that enjoys several interesting properties in terms of expressiveness and concision.

The addition of control states to Petri nets under the partial-order semantics leads to undecidable problems. Similarly to MSGs, one cannot decide in particular whether two given VASSs describe the same process language. However we show that basic problems about the set of markings reached along the processes of a VASS, such as boundedness, covering and reachability, can be reduced to the analogous problems for Petri nets. This relies on a new technique that simulates all prefixes of all processes. In this way Petri net tools can be used to verify the properties of a VASS under the process semantics.

We present also a technique to check effectively any MSO property of these partial orders, provided that the given system is bounded. This enables us to tackle more verification problems and subsumes known results for the model checking of MSGs. All algorithms presented in this paper have been implemented in a prototype tool available on-line.

***Keywords*-Petri nets, vector addition systems with states, non-branching processes, message sequence charts, compositional message sequence graphs, reachability, model-checking, monadic second-order logic, labeled partial orders**

INTRODUCTION

Consider a set of reactions that take place among a collection of particles such that each reaction consumes a multiset of available particles and produces a linear combination of other particle types. This kind of framework can be formalized by a vector addition system [16] or, equivalently, a (pure) Petri net [21]. Consider in addition some control state which determines whether a reaction can occur or not, and such that the occurrence of a reaction leads to a possibly distinct control state. Then the model becomes formally a vector addition system with states (a VASS), a notion introduced in [15]. It is well-known that all these models are computationally equivalent, because they can simulate each other [21].

The popular model of message sequence graphs (MSGs) can be regarded as a particular case of VASSs where the only allowed reactions are the sending and the receipt of one message from one site to another [4], [6], [11], [13], [18]. Then each sequence of reactions can be described by a partial order of events called a message sequence chart (MSC). Each MSC corresponds to several sequences of elementary actions which are equivalent up to the reordering of independent events. Similarly each sequence of MSCs is equivalent to several sequences of MSCs. Thus control states are used to focus on particular interleavings of events in order to avoid the state explosion problem due to concurrency. However there exists so far no way to regard an execution of a VASS as a partial order of events. Consequently there is no means to apply techniques or tools for Petri nets to the analysis of MSGs. In this paper we study a partial order semantics for VASSs in such a way that MSGs can effectively be regarded as a particular case of VASS. We obtain a framework that allows for counters and message losses as opposed to most works on MSCs in the literature.

We present in Section I a partial order semantics for VASSs which extends the usual process semantics of Petri nets. The approach is simple and natural. First we consider the set of firable computation sequences of a VASS and second we define the processes that represent a given sequence. Then each process describes some causal dependencies between events which are no longer linearly ordered. In this way, message sequence graphs are embedded in the framework of VASSs. However, one specific feature of the process semantics is that a computation sequence can yield several non-isomorphic processes depending on the order identical particles are consumed. Along this paper, we shall exhibit few other facts which make clear that the model of VASS is more general and more difficult to handle than MSGs.

It is easy to prove that checking the inclusion (or the equality) of two process languages given by two VASSs is undecidable by a reduction to the universality problem in Mazurkiewicz traces [22, Theorem IV.4.3]. This basic observation illustrates the computational gap between Petri

nets and VASSs under the process semantics because these two problems are decidable for Petri nets.This shows also that the analysis of the partially ordered executions of a VASS does not boil down to the verification of a Petri net in general, in spite of the well-known simulation of a VASS by a Petri net. However we present in the rest of this paper several new techniques to check properties of a VASS under the process semantics.

A key verification problem for MSGs is to detect channel divergence, i.e. to decide whether the number of pending messages along an execution is unbounded [4], [6]. This problem is NP-complete. An analogous problem in the more general setting of VASSs is the prefix-boundedness problem. It consists in checking that the set of markings reached by *prefixes* of processes is finite. We present in Section II a technique to solve this problem by means of a reduction to Petri nets. We stress that our construction differs from the usual simulation of a VASS by a Petri net because *the latter does not preserve prefix-boundedness*. We prove that prefix-boundedness is computationally equivalent to the boundedness problem for Petri nets and consequently requires exponential space [10]. This result exhibits an interesting complexity gap between MSGs and VASSs. It shows that tools used to check properties of MSGs need to be adapted in order to deal with the more expressive framework of VASSs. Other basic decision problems for the markings reached by prefixes are of course interesting. We show in particular that the reachability and the covering of a given marking can be solved using the same technique.

The model-checking problem for MSGs against monadic second-order logic (MSO) was investigated first in [17]. As opposed to earlier works [4], formulas are interpreted on the partially ordered scenarios accepted by the MSGs. This problem was proved decidable for the whole class of safe MSGs [18] (see also [11]). Each safe MSG can be regarded as a bounded VASS. However a safe MSG can describe an infinite set of markings because the reordering of events can produce an unbounded number of pending messages within channels: In other words, a safe MSG may be divergent. We present in Section III a technique to check effectively that all processes of a given bounded VASS satisfy a given MSO formula. We shall explain in details why this result subsumes, but cannot be reduced to, previous works on the model-checking of MSGs.

Due to the page limit, all proofs are omitted but they are available in the full paper [5]. The algorithms presented in this paper have been implemented in a prototype tool [3] which is built on TINA [2] for the reachability properties and MONA [1] for the MSO model-checking.

I. Model and semantics

The goal of this section is to extend the usual process semantics from Petri nets to VASSs. In order to avoid repetitive definitions we introduce the model of Petri nets

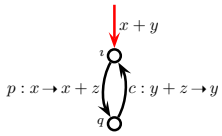

FIG. 1. A PNS with two control states

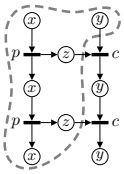

FIG. 2. A labeled causal net and a prefix

with states as a minimal framework which includes both Petri nets and VASSs. Thus *Petri nets are regarded as Petri nets with states provided with a single state whereas VASSs are simply Petri nets with states using* pure *transition rules, only*. Next we introduce the notions of firable computation sequence, reachable marking, and (non-branching) process as simple generalizations of the classical definitions in the restricted setting of Petri nets.

For simplicity's sake, for any mapping $f : A \to B$ between two finite sets A and B, we shall denote also by f the natural mapping $f : A^\star \to B^\star$ from words over A to words over B and the mapping $f : \mathbb{N}^A \to \mathbb{N}^B$ from multisets over A to multisets over B such that $f(\mu) = \sum_{a \in A} \mu(a) \cdot f(a)$ for each multiset $\mu \in \mathbb{N}^A$. Moreover we will often identify a set S with the multiset μ_S for which $\mu_S(x) = 1$ if $x \in S$ and $\mu_S(x) = 0$ otherwise.

A. Petri net with states

We borrow from the setting of Petri nets the abstract notion of *places* which can represent different kinds of components within a system: A local control state of a sequential process, a communication channel, a shared register, a particle type, a molecule in a chemical system, etc. We let P denote a finite set of places throughout this paper. As usual a multiset of places is called a *marking* and it is regarded as a distribution of *tokens* in places. Further we fix a finite set Λ of *rule names*.

A transition rule (or a reaction) is a means to produce new tokens in some places by consuming tokens in some other places. Formally a *rule* is a triple $r = (\lambda, \alpha, \beta)$ where $\lambda \in \Lambda$ is a rule name and $\alpha, \beta \in \mathbb{N}^P$ are markings called the *guard* and the *update* respectively. Such a rule is denoted by $\lambda : \alpha \to \beta$. It means intuitively that a multiset of tokens α can be consumed to produce a multiset of tokens β in an atomic way. Different rules can share the same guard α and the same update β. That is why we use here rule names to distinguish between similar but distinct rules. For each rule $r = (\lambda, \alpha, \beta)$, we put ${}^\bullet r = \alpha$ and $r^\bullet = \beta$.

Definition 1.1: A *Petri net with states* (for short: A PNS) over a set of rules R is an automaton $\mathcal{S} = (Q, \imath, \longrightarrow, \mu_{\rm in})$ where Q is a finite set of states, with a distinguished initial state $\imath \in Q$, $\longrightarrow \subseteq Q \times R \times Q$ is a finite set of arcs labeled by rules, and $\mu_{\rm in} \in \mathbb{N}^P$ is some initial marking.
Let $\mathcal{S} = (Q, \imath, \longrightarrow, \mu_{\rm in})$ be a Petri net with states. A labeled arc $(q_1, r, q_2) \in \longrightarrow$ will be denoted by $q_1 \xrightarrow{r} q_2$. A rule sequence $s = r_1 ... r_n \in R^\star$ is called a *computation sequence of* $\mathcal{S}$ if there are states $q_0, ..., q_n \in Q$ such that $\imath = q_0$ and for each $i \in [1, n]$, $q_{i-1} \xrightarrow{r_i} q_i$. These conditions will be summed-up by the notation $\imath \xrightarrow{s} q_n$. For instance, $(p : x \to x+z) \cdot (c : y+z \to y) \cdot (p : x \to x+z) \cdot (c : y+z \to y)$ is a computation sequence of the PNS with two states depicted in Fig. 1. We denote by $\mathrm{CS}(\mathcal{S})$ the set of all computation sequences of $\mathcal{S}$.

A rule sequence $s = r_1 ... r_n \in R^\star$ is *firable* from a marking μ if there are multisets of places μ_0, ..., μ_n such that $\mu_0 = \mu$ and for each $k \in [1, n]$: $\mu_{k-1} \geqslant {}^\bullet r_k$ and $\mu_k = \mu_{k-1} - {}^\bullet r_k + r_k^\bullet$. This means intuitively that each rule from s can be applied from the marking μ in the linear order specified by s: Each rule r_k consumes ${}^\bullet r_k$ tokens from μ_{k-1} and produces $r_k^\bullet$ new tokens which yields the subsequent multiset μ_k. Then we say that μ_n is reached by the rule sequence s from the marking μ. We also say that s leads to μ_n. We denote by $\mathrm{FCS}(\mathcal{S})$ the set of all *firable* computation sequences of $\mathcal{S}$. A marking is *reachable* in $\mathcal{S}$ if it is reached by a firable computation sequence of $\mathcal{S}$. A PNS is said to be *bounded* if the set of its reachable markings is finite.

B. VASS, Petri net and causal net

Originally introduced in [15], the notion of a *vector addition system with states* (for short: A VASS) can be formally defined in several slightly different ways. In this paper, a *VASS* is simply a PNS such that each rule r labeling an arc is *pure*, which means that for all places $p \in P$, ${}^\bullet r(p) \times r^\bullet(p) = 0$. This amounts to require that ${}^\bullet r(p) \geqslant 1$ implies $r^\bullet(p) = 0$ and vice versa. For this reason each rule r in a VASS can be represented by a vector $v \in \mathbb{Z}^P$ where $v(p) = r^\bullet(p) - {}^\bullet r(p)$ for all $p \in P$. We explain at present why we can identify the well-known formalism of Petri nets as particular PNSs provided with a single state.

Definition 1.2: A *Petri net* is a quadruple $\mathcal{N} = (P, T, W, \mu_{\rm in})$ where

- P is a finite set of places and T is a finite set of transitions such that $P \cap T = \emptyset$;
- W is a map from $(P \times T) \cup (T \times P)$ to $\mathbb{N}$, called the *weight function*;
- $\mu_{\rm in}$ is a map from P to $\mathbb{N}$, called the *initial marking*.

We shall depict Petri nets in the usual way as in Fig. 4: Black rectangles represent transitions whereas circles represent places; moreover tokens in places describe the initial marking. Given a Petri net $\mathcal{N} = (P, T, W, \mu_{\rm in})$ and a transition $t \in T$, ${}^\bullet t = \sum_{p \in P} W(p, t) \cdot p$ is the *pre-multiset* of t and $t^\bullet = \sum_{p \in P} W(t, p) \cdot p$ is the *post-multiset* of t. Similarly we put ${}^\bullet p = \sum_{t \in T} W(t, p) \cdot t$ and $p^\bullet = \sum_{t \in T} W(p, t) \cdot t$ for each place $p \in P$.

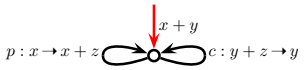

FIG. 3. A PNS with a single state

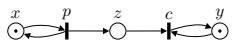

FIG. 4. and the corresponding Petri net

Let $\mathcal{N} = (P, T, W, \mu_{\rm in})$ be a Petri net. We will regard $\mathcal{N}$ as a PNS $\mathcal{S}_\mathcal{N}$ with the same set of places P and the same initial marking. Moreover $\mathcal{S}_\mathcal{N}$ is provided with a single state $\imath$ such that each transition $t \in T$ is represented by a self-loop labeled arc $\imath \xrightarrow{r} \imath$ where $r = (t, {}^\bullet t, t^\bullet)$. In this way, the class of Petri nets is faithfully embedded into the subclass of PNSs provided with a single state such that each transition carries a rule with a distinct rule name. For instance the PNS from Fig. 3 corresponds to the Petri net from Fig. 4.

If the weight function W takes only binary values then it is often described as a flow relation $F \subseteq (P \times T) \cup (T \times P)$ where $(x, y) \in F$ if $W(x, y) = 1$. Further F^+ denotes the transitive closure of F.

Definition 1.3: [9], [25] A *causal net* is a Petri net $\mathcal{K} = (B, E, F, \mu_{\rm min})$ whose places are called *conditions*, whose transitions are called *events*, and whose weight function takes values in $\{0, 1\}$ and is represented by a flow relation $F \subseteq (B \times E) \cup (E \times B)$ which satisfies the following requirements:

1) the net is acyclic, i.e. for all $x, y \in B \cup E$, $(x, y) \in F^+$ implies $(y, x) \notin F^+$.
2) the conditions do not branch, i.e. $|{}^\bullet b| \leqslant 1$ and $|b^\bullet| \leqslant 1$ for all $b \in B$.
3) the minimal conditions correspond to the initial marking: For all $b \in B$, $\mu_{\rm min}(b) = 1$ if ${}^\bullet b = \emptyset$ and $\mu_{\rm min}(b) = 0$ otherwise.

The transitive and reflexive closure F^* of the flow relation F in a causal net $\mathcal{K} = (B, E, F, \mu_{\rm min})$ yields a partial order over the set of events E. A *configuration* is a subset of events $H \subseteq E$ that is downwards closed, i.e. $e' F^* e$ and $e \in H$ imply $e' \in H$. Each configuration H defines a *prefix causal net* $\mathcal{K}_H$ whose events are precisely the events from H and whose conditions consist of the minimal conditions of $\mathcal{K}$ (with respect to the partial order relation F^*) and all places related to some event from H. For each class of labeled causal nets $\mathcal{L}$, we denote by $\mathrm{Pref}(\mathcal{L})$ the class of all prefixes of all labeled causal nets from $\mathcal{L}$.

C. Process semantics of a PNS

In this paper we are interested in a semantics of PNS based on causal nets which is a direct generalization of the

process semantics of Petri nets [9], [12], [25]. A process of a PNS $\mathcal{N}$ is a causal net $\mathcal{K}$ in which each condition of $\mathcal{K}$ is labeled by a place of $\mathcal{N}$ and each event of $\mathcal{K}$ is labeled by a transition of $\mathcal{N}$. The process semantics characterizes the labeled causal nets that describe an execution of a given Petri net. For instance the labeled causal net $\mathcal{K}$ from Fig. 2 depicts a process of the Petri net $\mathcal{N}$ from Fig. 4.

The following definition explains how processes are derived from a given rule sequence. Next the processes of a PNS will be defined as the processes of its firable computation sequences (Def. 1.5).

Definition 1.4: Let P be a set of places, Λ be a set of rule names, and R be a set of rules over P and Λ. A *process* of a rule sequence $s = r_1...r_n \in R^\star$ from a marking $\mu \in \mathbb{N}^P$ consists of a causal net $\mathcal{K} = (B, E, F, \mu_{\min})$ with n events e_1, ..., e_n provided with a labeling $\pi : B \cup E \to P \cup \Lambda$ such that the following conditions are satisfied:

1) $\pi(b) \in P$ for all $b \in B$, $\pi(e) \in \Lambda$ for all $e \in E$, and $\pi(\mu_{\min}) = \mu$;
2) $r_i = (\pi(e_i), \pi({}^\bullet e_i), \pi(e_i{}^\bullet))$ for all $i \in [1, n]$;
3) $e_i F^+ e_j$ implies $i < j$ for any two $i, j \in [1, n]$.

We denote by $[\![s]\!]_\mu$ the class of all processes of s from μ. In this definition the mapping π denotes the labeling of $\mathcal{K}$ and its natural extension to multisets. The first condition asserts that the initial marking of the causal net describes the marking μ; moreover each condition is associated with some place and each event corresponds to some rule name. The second condition requires that the label, the pre-set and the post-set of each event coincide with the name, the guard and the update of the corresponding rule. Finally the last property ensures that the total order of rules in s corresponds to an order extension of the partial order of events in $\mathcal{K}$. Consequently any subset of events $\{e_1, ..., e_k\}$ is downwards closed. Moreover the prefix causal net $\mathcal{K}'$ corresponding to the configuration $\{e_1, ..., e_{n-1}\}$ is a process of the rule sequence $r_1...r_{n-1}$ from the same marking μ.

Let H be a configuration of a process $\mathcal{K} = (B, E, F, \mu_{\min}, \pi)$ of a rule sequence s from μ. Let $B_{\max}$ be the set of maximal conditions of the prefix $\mathcal{K}_H$ w.r.t. F^*. Then the multiset of places $\pi(B_{\max})$ is called the *marking reached by* $\mathcal{K}_H$ and we say that $\mathcal{K}_H$ *leads to the marking* $\pi(B_{\max})$. Let s_H be a linear order that extends the partial order of events occurring in H. Then it is clear that the rule sequence $\pi(s_H)$ is firable from μ and leads to the marking $\pi(B_{\max})$; moreover $\mathcal{K}_H$ is a process of $\pi(s_H)$ from μ.

Definition 1.5: Let $\mathcal{S}$ be a PNS with initial marking μ_{in}. A *process of* $\mathcal{S}$ is a process of a computation sequence of $\mathcal{S}$ from μ_{in}. We let $[\![\mathcal{S}]\!]$ denote the class of all processes of $\mathcal{S}$. Thus $[\![\mathcal{S}]\!] = \bigcup_{s \in \text{CS}(\mathcal{S})} [\![s]\!]_{\mu_{\text{in}}}$. It is easy to check that the processes of a PNS provided with a single state are precisely the processes of the corresponding Petri net w.r.t. the usual process semantics [25].

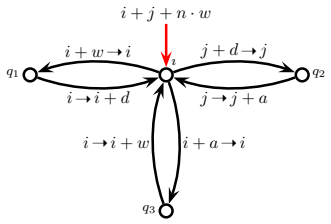

FIG. 5. Sliding window protocol

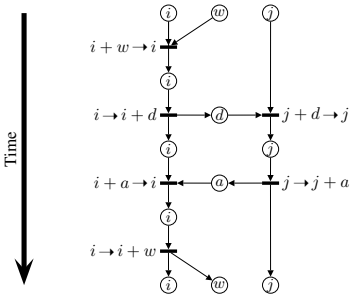

FIG. 6. A process with $n = 1$

D. From compositional MSGs to PNSs

The formalism of compositional message sequence graphs (cMSGs) was introduced in [13] in order to strengthen the expressive power of MSGs. As opposed to usual MSGs, cMSGs are built on components MSCs in which unmatched send or receive events are allowed. It was argued in [13] that simple protocols such as the alternating bit protocol can be described by cMSGs but not by MSGs. With no surprise cMSGs can be regarded as a particular case of VASS under the process semantics.

Consider a distributed system consisting of a set I of sites and a set K of communication channels between pairs of sites. The behaviour of such a system can be specified by a PNS over the set $P = I \cup K$ of places such that the sending of a message from site i to site j within the channel $k_{i,j}$ from i to j is encoded by a rule $i \to i + k_{i,j}$ and the receipt of such a message is encoded by a rule $j + k_{i,j} \to j$. Then we require that the initial marking contains a single token in each place $i \in I$. Such a PNS can actually be regarded as a compositional message sequence graph. The semantics of cMSGs consists of message sequence charts which are simply a partial order of events obtained from a process by removing all conditions.

Example 1.6: The PNS from Figure 5 describes a simplified sliding window protocol used to transmit data from a server i to a client j. The maximal number of missing acknowledgments is specified by the n initial tokens in the place w (the window). The system behaviour consists of three basic steps.

1) The server sends a new data formalized by a token d if some token w is available: It consumes first a w token:

$i + w \to i$ and next sends a new data: $i \to i + d$.

2) The client receives a data and returns an acknowledgment formalized by a token a: It consumes first a data: $j + d \to j$ and next produces the ack: $j \to j + a$.
3) The server receives an acknowledgment and increments the window size: First the ack is consumed: $i + a \to i$ and then a new token w is released: $i \to i + w$.

A typical process of this system with $n = 1$ is depicted in Figure 6. It is clear that this system is bounded.

Since counters are prohibited in MSGs, any safe cMSG equivalent to the PNS from the above example needs n distinct states. Its size is thus exponential w.r.t. the size of the PNS, provided that n is encoded in binary. In this way a bounded PNS can be exponentially more concise than an equivalent safe cMSG.

II. Checking reachability properties of prefixes

In this section we investigate three basic verification problems about the set of markings reached by *prefixes* of processes: Boundedness, covering and reachability. We show how to reduce these problems to the particular case of Petri nets in such a way that all complexity results extend from Petri nets to PNSs under the process semantics.

Definition 2.1: A marking μ is *prefix-reachable* in a PNS $\mathcal{S}$ if there exists a prefix of a process of $\mathcal{S}$ which leads to the marking μ.

Thus any reachable marking is prefix-reachable. Yet the set of prefix-reachable markings can differ from the set of reachable markings in general. For instance, each process of the PNS from Fig. 1 leads to a marking with at most 3 tokens whereas prefixes of these processes lead to infinitely many distinct markings (see in Fig. 2 a prefix of a process which leads to a marking with 4 tokens). Consequently this PNS is bounded but not prefix-bounded. In the particular case of Petri nets, however, any prefix-reachable marking is reachable, because the class of processes is prefix-closed. Thus the problems we study in this section are well-known for Petri nets but new for Petri nets with states.

The first basic problem we consider is the prefix-boundedness problem, which asks whether the set of prefix-reachable markings of a given PNS $\mathcal{S}$ is finite. We propose in this section a linear construction of a PNS $\mathcal{S}^\circ$ from $\mathcal{S}$ such that $\mathcal{S}$ is prefix-bounded if and only if $\mathcal{S}^\circ$ is bounded. Since the boundedness of $\mathcal{S}^\circ$ boils down to the boundedness of a Petri net, we get that the prefix-boundedness problem for PNSs is computationally equivalent to the boundedness problem of Petri nets. Further we show that this technique apply to other similar basic problems about prefix-reachable markings, namely covering and reachability.

A. From Petri nets with states to Petri nets

Let $\mathcal{S} = (Q, \imath, \longrightarrow, \mu_{\text{in}})$ be a fixed PNS. We build a PNS $\mathcal{S}^\circ$ that allows us to analyse the set of prefix-reachable markings of $\mathcal{S}$. The construction of $\mathcal{S}^\circ$ from $\mathcal{S}$ is illustrated

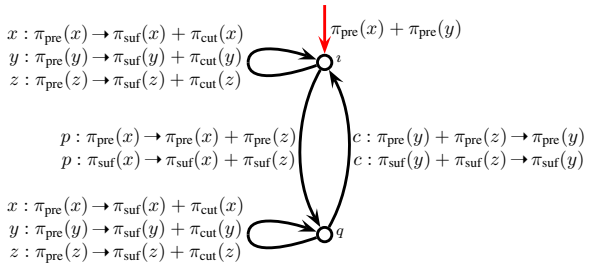

FIG. 7. Verification of prefix-reachable markings

by Fig. 7 where the PNS $\mathcal{S}^\circ$ resulting from the PNS $\mathcal{S}$ from Fig. 1 is depicted. Intuitively the PNS $\mathcal{S}^\circ$ is made of two copies of $\mathcal{S}$ that share the same set of states and that are in charge of executing on the fly events from the prefix or from the suffix respectively. Additionally some new loop labeled arcs allow tokens to move from the prefix to the suffix: This transfer is tracked by particular cut places in order to represent the marking reached by the resulting prefix.

The PNS $\mathcal{S}^\circ$ makes use of three disjoint sets of places: $P_{\text{pre}}, P_{\text{suf}}, P_{\text{cut}}$ which are copies of the set of places P of $\mathcal{S}$. We let $\pi_{\text{pre}} : P \to P_{\text{pre}}$, $\pi_{\text{suf}} : P \to P_{\text{suf}}$, and $\pi_{\text{cut}} : P \to P_{\text{cut}}$ be the bijections that map each place from P to the corresponding place in P_{pre}, P_{cut} and P_{suf} respectively. These mappings extend naturally to mappings from multisets to multisets. The initial marking μ°_{in} of $\mathcal{S}^\circ$ is the multiset $\mu^\circ_{\text{in}} = \pi_{\text{pre}}(\mu_{\text{in}})$.

The PNS $\mathcal{S}^\circ$ shares with $\mathcal{S}$ its set of states Q and its initial state $\imath$. It consists of three disjoint sets of labeled arcs: $\longrightarrow_{\text{pre}}, \longrightarrow_{\text{suf}}, \longrightarrow_{\text{cut}}$. The restriction of $\mathcal{S}^\circ$ to the labeled arcs from $\longrightarrow_{\text{pre}}$ and to the places from P_{pre} yields a PNS $\mathcal{S}^\circ_{\text{pre}}$ isomorphic to $\mathcal{S}$. Thus for each labeled arc $q_1 \xrightarrow{r} q_2$ in $\mathcal{S}$ with $r = (a, {}^\bullet r, r^\bullet)$ there exists some labeled arc $q_1 \xrightarrow{s}_{\text{pre}} q_2$ with $s = (a, \pi_{\text{pre}}({}^\bullet r), \pi_{\text{pre}}(r^\bullet))$. Similarly the restriction of $\mathcal{S}^\circ$ to the labeled arcs from $\longrightarrow_{\text{suf}}$ and to the places from P_{suf} yields a PNS $\mathcal{S}^\circ_{\text{suf}}$ isomorphic to $\mathcal{S}$, except that its initial marking is empty: For each labeled arc $q_1 \xrightarrow{r} q_2$ in $\mathcal{S}$ with $r = (a, {}^\bullet r, r^\bullet)$ there exists some labeled arc $q_1 \xrightarrow{s}_{\text{suf}} q_2$ with $s = (a, \pi_{\text{suf}}({}^\bullet r), \pi_{\text{suf}}(r^\bullet))$. The set of labeled arcs $\longrightarrow_{\text{cut}}$ consists of a self-loop $q \xrightarrow{s}_{\text{cut}} q$ for each state q and each place $p \in P$; this labeled arc allows to move a token from the place $\pi_{\text{pre}}(p)$ to the place $\pi_{\text{suf}}(p)$ and to keep track of that transfer in the place $\pi_{\text{cut}}(p)$, i.e. ${}^\bullet s = \pi_{\text{pre}}(p)$ and $s^\bullet = \pi_{\text{suf}}(p) + \pi_{\text{cut}}(p)$. Note that tokens in P_{cut} cannot be consumed.

Intuitively, for any process $\mathcal{K}$ of $\mathcal{S}$ and for any prefix $\mathcal{K}'$ of $\mathcal{K}$, the PNS $\mathcal{S}^\circ$ can simulate a computation sequence of $\mathcal{S}$ which corresponds to $\mathcal{K}$ in such a way that each event from the prefix $\mathcal{K}'$ corresponds to the occurrence of a labeled arc from $\longrightarrow_{\text{pre}}$ and each event from the suffix $\mathcal{K} \setminus \mathcal{K}'$ corresponds to the occurrence of a labeled arc from $\longrightarrow_{\text{suf}}$. Moreover the set of places P_{cut} keeps track of the tokens transferred from $\mathcal{K}$ to $\mathcal{K}'$, i.e. from $\mathcal{S}^\circ_{\text{pre}}$ to $\mathcal{S}^\circ_{\text{suf}}$, by the labeled

arcs from $\longrightarrow_{\text{cut}}$. Thus any prefix-reachable marking of $\mathcal{S}$ is represented by the restriction to $P_{\text{pre}} \cup P_{\text{cut}}$ of a reachable marking of $\mathcal{S}^\circ$. The key property of this representation, stated in Prop. 2.2 below, asserts that, conversely, each firable computation sequence of $\mathcal{S}^\circ$ corresponds to a process $\mathcal{K}$ of $\mathcal{S}$ and a prefix $\mathcal{K}'$ of $\mathcal{K}$ such that the marking of $P_{\text{pre}} \cup P_{\text{cut}}$ describes the marking reached by $\mathcal{K}'$.

In the next statement, for each marking μ and for each subset of places X, we denote by $\mu|X$ the restriction of μ to the places from X. The main results of this section rely essentially on the next observation. We claim that any prefix-reachable marking of $\mathcal{S}$ is represented by a reachable marking of μ° and vice versa. The interested reader is referred to the detailed proof given in [5, Subsection 3.2].

Proposition 2.2: A multiset of places $\mu \in \mathbb{N}^P$ is prefix-reachable in $\mathcal{S}$ if and only if there exists some reachable marking μ° of $\mathcal{S}^\circ$ such that $\mu = \pi_{\text{pre}}^{-1}(\mu^\circ|P_{\text{pre}}) + \pi_{\text{cut}}^{-1}(\mu^\circ|P_{\text{cut}})$.

B. Proof sketch of Proposition 2.2

For any rule sequence $u \in R^\star$, we call requirement of u and we denote by $\text{req}(u)$ the least marking μ such that u is firable from μ. This means that $[\![u]\!]_\mu \neq \emptyset$ if and only if $\mu \geqslant \text{req}(u)$. Let $\mathcal{S} = (Q, \imath, \longrightarrow, \mu_{\text{in}})$ be a PNS. For each rule sequence $u = r_1 ... r_n \in R^\star$ firable from μ_{in}, we let μ_u denote the marking reached by u from μ_{in}, i.e. $\mu_u = \mu_{\text{in}} + \sum_{i=1}^{n} (r_i^\bullet - {}^\bullet r_i)$. Similarly for each rule sequence s firable from the initial marking μ_{in}°, μ_s° denotes the marking reached by s in $\mathcal{S}^\circ$.

We shall use the following notion of partial computation: A *partial computation* is a triple $(u, v, w) \in R^\star \times R^\star \times R^\star$ such that $[\![v.w]\!]_{\mu_{\text{in}}} \cap [\![u]\!]_{\mu_{\text{in}}} \neq \emptyset$ and $u \in \text{CS}(\mathcal{S})$. Then $[\![v]\!]_{\mu_{\text{in}}} \neq \emptyset$ hence the rule sequence v is firable from μ_{in}. A partial computation is used as a witness for a process $\mathcal{K}_u$ of u and a prefix $\mathcal{K}_v$ of $\mathcal{K}_u$ with $\mathcal{K}_v \in [\![v]\!]_{\mu_{\text{in}}}$. Note that v need not to be a prefix of u, nor to be a computation sequence of $\mathcal{S}$. Partial computations are closely related to prefix-reachable markings, as the next basic observation shows.

Proposition 2.3: For each partial computation (u, v, w), the marking μ_v is prefix-reachable. Conversely, for any prefix-reachable marking μ, there exists some partial computation (u, v, w) such that $\mu = \mu_v$.

The proof of Prop. 2.2 relies on the two next technical lemmas which can be established by means of a bit tedious inductions. The first one asserts that for each firable computation sequence $u \in \text{FCS}(\mathcal{S})$ and each prefix $\mathcal{K}_v$ of each process $\mathcal{K}_u \in [\![u]\!]_{\mu_{\text{in}}}$, the VASS $\mathcal{S}^\circ$ can be guided in order to simulate each rule of u in its sequential order so that the marking reached by u is described by the current marking of $P_{\text{pre}} \cup P_{\text{suf}}$ while the marking reached by $\mathcal{K}_v$ is described by the current marking of $P_{\text{pre}} \cup P_{\text{cut}}$. Furthermore we have to make sure that the state $q \in Q$ reached by u is also reached by s in $\mathcal{S}^\circ$ and to check that all events from $\mathcal{K}_u$ that do not occur in $\mathcal{K}_v$ are performed by transitions from $\longrightarrow_{\text{suf}}$. To do so, we have to guide $\mathcal{S}^\circ$ to transfer *exactly* the required number of tokens from P_{pre} to P_{suf}, which corresponds to the marking of P_{cut}.

Lemma 2.4: Let (u, v, w) be a partial computation in $\mathcal{S}$ and q be some state such that $\imath \xrightarrow{u} q$ in $\mathcal{S}$. There exists some firable rule sequence s in $\mathcal{S}^\circ$ which leads to the marking μ_s° such that

(a) $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu_v$,
(b) $\pi_{\text{suf}}^{-1}(\mu_s^\circ|P_{\text{suf}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu_u$,
(c) $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) = \text{req}(w)$,
(d) $\imath \xrightarrow{s} q$ in $\mathcal{S}^\circ$.

Conversely we need to show that the marking of $P_{\text{pre}} \cup P_{\text{cut}}$ reached after any firable transition sequence s of $\mathcal{S}^\circ$ corresponds to a prefix-reachable marking of $\mathcal{S}$, i.e. to some partial computation (u, v, w). To do so, we have to build a firable rule sequence $u \in \text{FCS}(\mathcal{S})$, a process $\mathcal{K}_u \in [\![u]\!]_{\mu_{\text{in}}}$ and a prefix $\mathcal{K}_v \in [\![v]\!]_{\mu_{\text{in}}}$ inductively from s. At each step the state reached by s coincides with the state reached by u. When $\mathcal{S}^\circ$ applies an additional labeled arc a, the corresponding partial computation is either (u, v, w) if $a \in \xrightarrow{r}_{\text{cut}}$; or $(u.r, v, w.r)$ if $a \in \xrightarrow{r}_{\text{suf}}$; or $(u.r, v.r, w)$ if $a \in \xrightarrow{r}_{\text{pre}}$. In this last case, the rule r and the sequence of rules w can be performed concurrently: Formally we shall establish that ${}^\bullet r + \text{req}(w) \leqslant \mu_v$. This property follows actually from the fact that w can be fired from the marking obtained by the tokens transferred from P_{pre} to P_{suf}, i.e. $\pi_{\text{cut}}(\text{req}(w)) \leqslant \mu_s^\circ|P_{\text{cut}}$.

Lemma 2.5: Let s be a firable rule sequence in $\mathcal{S}^\circ$ leading to the state q and the marking μ_s°. There exists some partial computation (u, v, w) of $\mathcal{S}$ such that

(a) $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu_v$,
(b) $\pi_{\text{suf}}^{-1}(\mu_s^\circ|P_{\text{suf}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu_u$,
(c) $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) \geqslant \text{req}(w)$, and
(d) $\imath \xrightarrow{u} q$ in $\mathcal{S}$.

We are now ready to prove Prop. 2.2. Let μ be the marking reached by a prefix $\mathcal{K}'$ of a process $\mathcal{K} \in [\![\mathcal{S}]\!]$. According to Prop. 2.3, there exists some partial computation (u, v, w) such that $\mu_v = \mu$. By Lemma 2.4, there exists some firable rule sequence s in $\mathcal{S}^\circ$ such that $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu_v = \mu$. Conversely if $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu$ for some firable rule sequence s in $\mathcal{S}^\circ$ then Lemma 2.5 ensures that there exists some partial computation (u, v, w) such that $\pi_{\text{cut}}^{-1}(\mu_s^\circ|P_{\text{cut}}) + \pi_{\text{pre}}^{-1}(\mu_s^\circ|P_{\text{pre}}) = \mu_v$. Moreover Prop. 2.3 asserts that μ_v is the marking reached by some prefix $\mathcal{K}'$ of some process $\mathcal{K} \in [\![\mathcal{S}]\!]$.

C. Analysis of prefix-reachable markings

Proposition 2.2 enables us to derive some techniques to analyse the set of prefix-reachable markings of $\mathcal{S}$. First, the *prefix-boundedness problem* asks whether the set of prefix-reachable markings of a given PNS $\mathcal{S}$ is finite. We can easily derive from Prop. 2.2 that the PNS $\mathcal{S}$ is prefix-bounded if

and only if the PNS $\mathcal{S}^\circ$ is bounded. This latter property can be checked by means of the usual linear simulation of a VASS by a Petri net. Thus,

Theorem 2.6: The prefix-boundedness problem of PNSs is computationally equivalent to the boundedness problem of Petri nets.

Second, the *prefix-covering problem* asks whether a given multiset of places $\mu \in \mathbb{N}^P$ is covered by some prefix-reachable marking $\mu' \in \mathbb{N}^P$, i.e. $\mu(p) \leqslant \mu'(p)$ for all $p \in P$. It is easy to see that μ is prefix-covered in $\mathcal{S}$ if and only if the multiset of places $\pi_{\text{cut}}(\mu)$ is covered by some reachable marking of $\mathcal{S}^\circ$. Thus,

Theorem 2.7: The prefix-covering problem for PNSs is computationally equivalent to the covering problem in Petri nets.

Last but not least, the *prefix-reachability problem* asks whether a given multiset of places is prefix-reachable in $\mathcal{S}$. Let us consider a slight modification $\mathcal{S}'$ of $\mathcal{S}^\circ$ where for each place $p \in P_{\text{suf}}$, each state $q \in \mathcal{S}^\circ$ is provided with an additional self-loop labeled arc which carries a rule that consumes a token from p and produces nothing. Then a multiset μ of places is prefix-reachable in $\mathcal{S}$ if and only if $\pi_{\text{cut}}(\mu)$ is reachable in $\mathcal{S}'$. Thus,

Theorem 2.8: The prefix-reachability problem of PNSs is computationally equivalent to the reachability problem of Petri nets.

III. Checking MSO properties of processes

At present we aim at checking more properties about the processes of a given PNS. We show in this section how to check effectively whether all processes of a given bounded Petri net with states $\mathcal{S}$ satisfy a formula ψ expressed in monadic second-order (MSO) logic. Since we do not require the PNS to be prefix-bounded, our technique applies to infinite state systems. It relies essentially on Büchi Theorem [7] and a notion of process coloring that enables us to recover a process from one of its linearizations.

A. MSO logic

In the rest of this section, we fix a bounded PNS $\mathcal{S}$ with an initial marking μ_{in} over the finite set of places P and the finite set of rules R. In order to simplify the presentation of our result, we consider in this section that the events of a process are labeled by a rule instead of a rule name. The MSO logic we consider applies to the class of partial orders whose nodes are labeled by letters from the disjoint union $\Sigma = P \dot\cup R$, which includes in particular the processes of each rule sequence $s \in R^\star$. Thus the models we consider here are triples $(N, \preccurlyeq, \xi)$ where N is a finite set of nodes, $\preccurlyeq$ is a partial order over N, and ξ is a mapping from N to $\Sigma = P \dot\cup R$.

Formulae of the MSO logic that we consider involve first-order variables $x, y, z...$ for nodes and second-order variables $X, Y, Z...$ for sets of nodes. They are built up from the atomic formulae $P_a(x)$ for $a \in \Sigma$ (which stands for "the node x is labeled by the letter a"), $x \preccurlyeq y$, and $x \in X$ by means of the Boolean connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ and quantifiers $\exists, \forall$ (both for first order and for set variables). Formulae without free variables are called *sentences*.

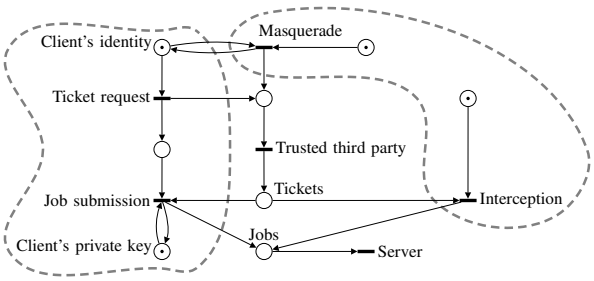

FIG. 8. A simple cryptographic protocol

The satisfaction relation $\models$ between a labeled partial order $(N, \preccurlyeq, \xi)$ and a sentence is defined canonically with the understanding that first order variables range over nodes of N and second order variables over subsets of N. The class of labeled partial orders which satisfy a sentence φ is denoted by $\text{Mod}(\varphi)$. We say that a class of labeled partial orders $\mathcal{L}$ is MSO-*definable* if there exists a sentence φ such that $\mathcal{L} = \text{Mod}(\varphi)$.

Example 3.1: The Petri net from Fig. 8 describes a simple cryptographic protocol for the submission of jobs to a server. The client is specified on the left-hand side. It can request tickets to a trusted third party by using its own identity. Then the third party produces a ticket that can be used to submit a job to the server, with the help of the client's private key. The behaviour of an intruder is depicted on the right-hand side. It can use the client's identity to produce a ticket request or intercept tickets. Consider at present the three next basic properties:

(P1) A ticket cannot be consumed without the client's private key.

(P2) The server does not consume jobs submitted by the intruder.

(P3) The client consumes only tickets that it has requested.

These properties can be easily formalized by MSO formulae over processes. For instance (P1) corresponds to the sentence $\forall x, (x : \text{Tickets}) \rightarrow (x : \text{Clients\_private\_key})$ where $x : p$ is a shorthand for the property that x is an event that consumes a token available in a condition labeled by p, i.e.

$$\exists y, P_p(y) \wedge y \prec x \wedge \forall z, (y \prec z \wedge z \preccurlyeq x) \rightarrow x \preccurlyeq z$$

The technique presented in this section can be used to check that (P1)$\rightarrow$(P2) for all processes of the above Petri net. Further it enables us to compute a counter-example (in the form of a process) for the property (P1)$\rightarrow$(P3).

B. A technique to decide $\mathcal{S} \models \psi$

Since $\mathcal{S}$ is bounded, we can compute and fix some natural number $\mathfrak{b}$ such that each reachable marking μ of $\mathcal{S}$ is $\mathfrak{b}$-bounded, that is, $\mu(p) \leqslant \mathfrak{b}$ for each $p \in P$. A rule sequence $s = r_1...r_m \in R^\star$ firable from μ_{in} is said to be $\mathfrak{b}$-*bounded* if the marking reached by each sub-sequence $r_1...r_l$ is $\mathfrak{b}$-bounded. In particular any firable computation sequence of $\mathcal{S}$ is $\mathfrak{b}$-bounded.

We fix a word $w_{\text{in}} \in P^\star$ that is a linear extension of μ_{in}, i.e. $|w_{\text{in}}|_p = \mu_{\text{in}}(p)$ for all $p \in P$. Similarly, for each rule $r \in R$, we fix a word $w_r = r.w'_r$ where $|w'_r|_p = r^\bullet(p)$ for all $p \in P$. Then for each rule sequence $s = r_1...r_m \in R^\star$, the sequence $w_s = w_{\text{in}}.w_{r_1}...w_{r_m}$ is called the *representative word of* s. We regard w_s as a linearly ordered set of nodes labeled by letters from Σ and we put $w_s = (N, \leqslant, \xi)$ where N is a set of nodes, $\leqslant$ is a total order over N, and $\xi : N \to \Sigma$ is a labeling. Nodes labeled by a place are called *place nodes* whereas nodes labeled by a rule are called *rule nodes*. Interestingly, w_s is a linear extension of any process of s, where the place nodes following a rule node labeled by r correspond to the multiset of tokens $r^\bullet$ produced by this occurrence of r.

In order to recover a process of s from the representative word w_s, we need to specify which available tokens are consumed by each occurrence of rule. To do so, we use a coloring of the place nodes of w_s so that at each step all available tokens in a given place get distinct colors. Moreover we also provide rule nodes with a series of other colors in order to specify which tokens are consumed at each step of s.

Definition 3.2: Let $w = (N, \leqslant, \xi)$ be a linear order of nodes labeled by Σ. A *process coloring* of w consists of

- a *partition* $C = \{C_1, ..., C_\mathfrak{b}\}$ of the set of place nodes; a place node $n \in N$ is said to be colored by k in place p if $\xi(n) = p$ and $n \in C_k$.
- for each place $p \in P$ and each $k \in [1..\mathfrak{b}]$, a subset of rule nodes $D_{p,k}$; we say that a rule node $n \in N$ consumes a token colored by k in place p if $n \in D_{p,k}$.

Moreover the three next conditions must be satisfied:

PC_1: For each rule node n, for each place $p \in P$, we have $\#\{k \in [1..\mathfrak{b}] \mid n \in D_{p,k}\} = ({}^\bullet\xi(n))(p)$;

PC_2: For each place $p \in P$ and each color $k \in [1..\mathfrak{b}]$, any two place nodes colored by k in place p are separated by some rule node which consumes a token colored by k in place p;

PC_3: For each rule node n which consumes a token colored by k in place p, there exists some preceding place node $n' < n$ colored by k in place p such that no rule node between n' and n consumes a token colored by k in place p.

Intuitively a place node belongs to C_k if it describes a token colored by k in place $\xi(n) \in P$. A rule node n belongs to $D_{p,k}$ if it describes an occurrence of the rule $\xi(n) \in R$ which consumes a token colored by k in place p. Thus the condition PC_1 asserts that n consumes the appropriate multiset of tokens in each place, provided that these tokens have distinct colors. Precisely PC_2 guarantees that the colors given to new tokens produced in a place by the occurrence of a rule differ from the colors used by available tokens in this place. It ensures also that the tokens produced in some place by the occurrence of a rule get distinct colors. Consequently, at each step all available tokens in a place have distinct colors. In order to recover a process of s from a process coloring of w_s, we have to make sure that there are enough available tokens when each rule is applied. The last requirement PC_3 guarantees that for each rule node which consumes a token colored by k in place p, some token of this kind occurred before the rule and has not been consumed in between.

w_s	x	y	z	p	z	x	c	y
C_1	✗		✗					✗
C_2		✗			✗	✗		
$D_{x,1}$				✗				
$D_{y,2}$							✗	
$D_{z,2}$							✗	

FIG. 9. A process coloring of $w_s = xyzpzxcy$

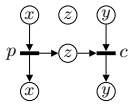

FIG. 10. Process of $s = pc$ corresponding to Fig. 9

We can show that the notion of process coloring characterizes the linear extensions of processes and allows to recover a process from a word. This property is established by the two next statements (Prop. 3.3 and 3.4). Consider for instance the rule sequence $s = pc$ from the initial marking $\mu_{\text{in}} = \{x, y, z\}$ where $p : x \to x + z$ and $c : y + z \to y$. A process coloring of $w_s = xyzpzxcy$ with $\mathfrak{b} = 2$ is given by the tabular of Fig. 9. The corresponding process is depicted in Fig. 10.

Proposition 3.3: Let $w_s = (N, \leqslant, \xi)$ be a linear order of nodes labeled by Σ which corresponds to the representative word of a rule sequence $s \in R^\star$. Let $C = (C_k)_{k \in [1..\mathfrak{b}]}$ and $D = (D_{p,k})_{p \in P, k \in [1..\mathfrak{b}]}$ form a process coloring of w_s. Let $\prec$ be the binary relation over N such that $x \prec y$ if

- either x is a rule node and y is a following place node with no rule node in between
- or y is a rule node and x is a preceding place node colored by k in place p such that y consumes a token colored with k in place p and no rule node between x and y consumes a token colored with k in place p.

Let $\preccurlyeq$ be the reflexive and transitive closure of $\prec$. Then the labeled partial order $(N, \preccurlyeq, \xi)$ is a process of s firable from μ_{in}, denoted by $\mathcal{K}_{C,D}(s)$. Moreover s is $\mathfrak{b}$-bounded.

Thus each process coloring of w_s yields a process from $[\![s]\!]_{\mu_{\text{in}}}$. Consequently s is firable from μ_{in} as soon as it admits a process coloring. With no surprise s has to be $\mathfrak{b}$-bounded, too. Conversely the next result asserts that each process of any rule sequence s firable from μ_{in} can be obtained by some process coloring of w_s, provided that s is $\mathfrak{b}$-bounded.

Proposition 3.4: Let $s = r_1...r_m$ be a $\mathfrak{b}$-bounded rule sequence firable from μ_{in} and $\mathcal{K}$ be a process of s. Then there exists a process coloring (C, D) of the representative word w_s such that $\mathcal{K}_{C,D}(s)$ is isomorphic to $\mathcal{K}$.

Thus the notion of process coloring characterizes the processes of any $\mathfrak{b}$-bounded rule sequence firable from μ_{in}.

Following the easy part of Büchi Theorem, we can design an MSO formula $\phi_{\mathcal{S}}$ which defines the words $w = (N, \leqslant, \xi)$ over Σ which are representative words of a computation sequence of $\mathcal{S}$. We can also design a formula $\phi_{pc}(C, D)$ with $\mathfrak{b} \times (|P| + 1)$ second-order free variables $C = (C_k)_{k\in[1..\mathfrak{b}]}$ and $D = (D_{k,p})_{k\in[1..\mathfrak{b}],p\in P}$ which characterizes the notion of a process coloring for a word $w = (N, \leqslant, \xi)$ over Σ. Moreover, we can build a formula $\phi_{\preccurlyeq}(x, y, C, D)$ with two first-order free variables x and y and $\mathfrak{b} \times (|P| + 1)$ second-order free variables such that for any interpretation of $C = (C_k)_{k\in[1..\mathfrak{b}]}$ and $D = (D_{k,p})_{k\in[1..\mathfrak{b}],p\in P}$ and any interpretation of x and y, $\phi_{\preccurlyeq}(x, y, C, D)$ is satisfied if and only if we have $x \preccurlyeq y$ in the process corresponding to the process coloring given by the interpretation.

Let ψ be an MSO sentence for labeled partial orders over Σ. We consider the following formula $\overline{\psi}_{\mathcal{S}}$ for words over Σ:

$$\overline{\psi}_{\mathcal{S}} = \phi_{\mathcal{S}} \wedge \exists C, \exists D, (\phi_{pc}(C, D) \wedge \neg\psi'(C, D))$$

where the formula $\psi'(C, D)$ is obtained from ψ by replacing each occurrence of $x \preccurlyeq y$ by $\phi_{\preccurlyeq}(x, y, C, D)$. Thus a word satisfies $\overline{\psi}_{\mathcal{S}}$ if (and only if) it is a representative word of a computation sequence s of $\mathcal{S}$ for which there exists a process coloring which describes a process satisfying $\neg\psi$. In this way we get the main result of this section.

Theorem 3.5: Let $\mathcal{S}$ be a bounded PNS and ψ be an MSO sentence over causal nets. All processes of $\mathcal{S}$ satisfy ψ if and only if the word sentence $\overline{\psi}_{\mathcal{S}}$ is *not* satisfiable.

We have implemented our technique on top of the tool MONA [1]. Our prototype [3] allows us in particular to design first a Petri net with TINA [2], next to use TINA to compute an upper bound for the reachable markings, and finally to apply Theorem 3.5 to check MSO formulae over processes with the help of MONA. Continuing Example 3.1, we could check that (P1)→(P2) for all processes. Further our tool was able to compute a counter-example for the property (P1)→(P3) in the form of a short process.

C. Comparisons to related works

Theorem 3.5 subsumes previous works in several extents. As opposed to [11], [18], we do not assume FIFO behaviours and consequently we cannot make use of the notion of representative linearizations. The fact is that, as already mentioned, a computation sequence can correspond to several non-isomorphic processes depending on the order identical particles are consumed. Therefore we need the notion of process coloring (Def. 3.2) and the related results to recover a process from a word. This is the main difference with the setting of MSCs because these are completely specified by any of their linearizations as long as messages are never lost and always delivered in a FIFO manner. Still, the FIFO restriction can be formalized in MSO logic and our technique applies also in this special case. Second Petri nets and VASSs abstract away from the notions of sites and channels in the setting of MSCs: A place can describe the local state of a site, a communication channel, a shared-variable, etc. In particular our approach applies to any bounded Petri net. To the best of our knowledge, the model checking problem of bounded Petri nets against MSO formulae under the process semantics has not been investigated so far in the literature.

The model-checking of graphs representing the executions of a system against MSO sentences has been studied in different settings. Provided that the class of graphs considered is definable in MSO logic and tree-width bounded, the satisfiability of an MSO formula is known to be decidable [8], [23], [19]. However the processes of a PNS need not to be MSO-definable —even in the particular case of a non-divergent MSG, because non-divergent MSGs can describe non-regular sets of MSCs— so this line of work does not apply to our setting. On the other hand, the class of processes of any bounded Petri net is MSO-definable. Consequently the partial order of events can be described by a concurrent automaton or a regular event structure [24] for which branching time model-checking is available [14]. This approach fails however for Petri nets with states which are not prefix-bounded or whose processes are not MSO-definable.

IV. Conclusion

We investigate a generalization of compositional MSGs which adopts the abstract token game of Petri nets and keeps a semantics based on partially ordered sets of events called processes. This model allows for the specification of bounded counters and appears to be exponentially more concise than MSGs. We show how to check basic properties of the markings reached along partial executions, namely boundedness, covering and reachability. Processes are a means to track the causes of events occurring in an execution. For bounded systems, we present a method to check any MSO property of processes by a reduction to the satisfiability of a word sentence. As illustrated by Example 3.1, the process semantics of Petri nets can be used to model and check systems with specific behavioural constraints, such as FIFO channels, causal communication, or private keys, as soon as these restrictions are formalized

by an MSO sentence. The techniques presented in this paper allow us to check protocol specifications that include message losses and bounded counters. They have been implemented in a prototype tool [3] built on top of TINA [2] to check the prefix-boundedness of a given PNS and MONA [1] to check MSO properties of processes of a given bounded PNS.

Previous works have proposed to mix MSCs and Petri nets. In particular, netcharts [20] form a model of distributed system where local states of components are formalized by places of a 1-safe Petri net whose transitions are labeled by an MSC. This model is expressively equivalent to communicating finite-state machines which makes it difficult to check under the FIFO semantics adopted. On the other hand Petri nets with states do not benefit so far from effective relationships to models of distributed systems similar to those available for MSGs [4], [11].

ACKNOWLEDGMENT

This work is partly supported by project ECSPER (ANR-09-JCJC-0069).

REFERENCES

[1] The MONA Project. http://www.brics.dk/mona. [Online; accessed 14-Jan-2013].

[2] TIme petri Net Analyzer (TINA). http://projects.laas.fr/tina.

[3] VASS Checker (VaChe). http://pageperso.lif.univ-mrs.fr/~florent.avellaneda/LeVaChe.

[4] R. Alur and M. Yannakakis. Model checking of message sequence charts. In Jos C. M. Baeten and Sjouke Mauw, editors, *CONCUR*, volume 1664 of *Lecture Notes in Computer Science*, pages 114–129. Springer, 1999.

[5] F. Avellaneda and R. Morin. Vector addition systems with states vs. Petri nets. Technical report, Laboratoire d'informatique Fondamentale de Marseille - LIF, October 2012. Available at http://hal.archives-ouvertes.fr/hal-00686444.

[6] H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In Ed Brinksma, editor, *TACAS*, volume 1217 of *Lecture Notes in Computer Science*, pages 259–274. Springer, 1997.

[7] J.R. Büchi. Weak second-order arithmetic and finite automata. *Z. Math. Logik Grundlagen Math.*, 6:66–92, 1960.

[8] B. Courcelle. The expression of graph properties and graph transformations in monadic second-order logic. In Grzegorz Rozenberg, editor, *Handbook of Graph Grammars*, pages 313–400. World Scientific, 1997.

[9] J. Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28:575–591, 1991.

[10] J. Esparza and M. Nielsen. Decidability issues for Petri nets - a survey. *Bulletin of the EATCS*, 52:244–262, 1994.

[11] B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.

[12] U. Goltz and W. Reisig. The non-sequential behavior of Petri nets. *Information and Control*, 57(2/3):125–147, 1983.

[13] E. L. Gunter, A. Muscholl, and D. Peled. Compositional message sequence charts. In Tiziana Margaria and Wang Yi, editors, *TACAS*, volume 2031 of *Lecture Notes in Computer Science*, pages 496–511. Springer, 2001.

[14] J. Gutierrez and J. C. Bradfield. Model-checking games for fixpoint logics with partial order models. *Inf. Comput.*, 209(5):766–781, 2011.

[15] J.E. Hopcroft and J-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.

[16] R.M. Karp and R.E. Miller. Parallel program schemata. *Journal of Computer and System Sciences*, 3(2):147–195, 1969.

[17] P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In Fernando Orejas, Paul G. Spirakis, and Jan van Leeuwen, editors, *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 809–820. Springer, 2001.

[18] P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In Ramesh Hariharan, Madhavan Mukund, and V. Vinay, editors, *FSTTCS*, volume 2245 of *LNCS*, pages 256–267. Springer, 2001.

[19] P. Madhusudan and G. Parlato. The tree width of auxiliary storage. In Thomas Ball and Mooly Sagiv, editors, *POPL*, pages 283–294. ACM, 2011.

[20] M. Mukund, K. N. Kumar, and P. S. Thiagarajan. Netcharts: Bridging the gap between hmscs and executable specifications. In Roberto M. Amadio and Denis Lugiez, editors, *CONCUR*, volume 2761 of *Lecture Notes in Computer Science*, pages 293–307. Springer, 2003.

[21] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Englewood Cliffs, New Jersey: Prentice Hall, Inc., 1981.

[22] J. Sakarovitch. *Elements of Automata Theory*. Cambridge University Press, 2009.

[23] D. Seese. The structure of models of decidable monadic theories of graphs. *Ann. Pure Appl. Logic*, 53(2):169–195, 1991.

[24] P. S. Thiagarajan. Regular event structures and finite Petri nets: A conjecture. In Wilfried Brauer, Hartmut Ehrig, Juhani Karhumäki, and Arto Salomaa, editors, *Formal and Natural Computing*, volume 2300 of *Lecture Notes in Computer Science*, pages 244–256. Springer, 2002.

[25] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets*, volume 625 of *Lecture Notes in Computer Science*. Springer, 1992.