A Concurrency-Preserving Translation from Time Petri Nets to Networks of Timed Automata

Sandie Balaguer, Thomas Chatain, Stefan Haar

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Concurrency-Preserving Translation from Time Petri Nets to Networks of Timed Automata

Sandie Balaguer — Thomas Chatain — Stefan Haar

N° 7338

Juillet 2010

Programs, Verification and Proofs

ISSN 0249-6399 ISRN INRIA/RR--7338--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
SACLAY - ÎLE-DE-FRANCE

A Concurrency-Preserving Translation from Time Petri Nets to Networks of Timed Automata

Sandie Balaguer , Thomas Chatain , Stefan Haar

Theme : Programs, Verification and Proofs
Algorithmics, Programming, Software and Architecture
Équipe-Projet MExICo

Rapport de recherche n° 7338 — Juillet 2010 — 22 pages

Abstract: Real-time distributed systems may be modeled in different formalisms such as time Petri nets (TPN) and networks of timed automata (NTA). This paper focuses on translating a 1-bounded TPN into an NTA and considers an equivalence which takes the distribution of actions into account. This translation is extensible to bounded TPNs. We first use S-invariants to decompose the net into components that give the structure of the automata, then we add clocks to provide the timing information. Although we have to use an extended syntax in the timed automata, this is a novel approach since the other transformations and comparisons of these models did not consider the preservation of concurrency.

Key-words: concurrency, timed traces, time Petri nets, networks of timed automata, concurrency-preserving translation

{balaguer, chatain, haar}@lsv.ens-cachan.fr

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université
4, rue Jacques Monod, 91893 ORSAY Cedex
Téléphone : +33 1 72 92 59 00

Traduction d'un réseau de Petri temporel vers un réseau d'automates temporisés, avec préservation de la concurrence*

Résumé : Les systèmes temporisés distribués peuvent être modélisés dans différents formalismes comme les réseaux de Petri temporels ou les réseaux d'automates temporisés. Cet article s'intéresse à la traduction d'un réseau de Petri temporel 1-borné vers un réseau d'automates temporisés et considère une équivalence qui tient compte de la distribution des actions. Cette traduction peut s'étendre aux réseaux de Petri temporels bornés. Nous utilisons d'abord les S-invariants pour décomposer le réseau en composants qui donnent directement la structure des automates, puis nous rajoutons des horloges pour retranscrire l'information temporelle. Bien que l'utilisation d'une syntaxe étendue dans les automates temporisés soit nécessaire, ce travail est une approche originale puisque les autres transformations et comparaisons de ces modèles ne considèrent pas la concurrence.

Mots-clés : concurrence, traces temporisées, réseaux de Petri temporels, réseaux d'automates temporisés, traduction

Contents

1 Introduction

Techniques that aim at improving reliability and safety of automated systems have dramatically improved during the last thirty years (synthesis, odel-checking, test...). Studying a complex system generally requires the use of multiple techniques and tools. Consequently the system must be translated from one formalism to another. The difficulty is to show that the different representations are equivalent. This work proposes a translation between two popular formalisms that describe timed concurrent systems: 1-bounded time Petri nets (TPN) [16] and networks of timed automata (NTA) [3]. These formalisms have different histories but they were both designed to model real-time, distributed systems. Moreover they both handle urgency, which is a key feature without which most real-time systems cannot be modeled correctly.

Because these two formalisms are used by different verification tools, transformations have been proposed, and we remark the following. (i) The transformations mainly rely on natural structural equivalences between the basic elements of the formalisms. For instance, the location of an automaton corresponds to a place of a Petri net, a transition of a Petri net corresponds to a tuple of synchronized transitions of an NTA, and the timed interval associated to a transition of a Petri net becomes a pair guard-invariant in a timed automaton. (ii) Beyond these natural equivalences, limitations for more general

models are not clear. Indeed, the natural transformations tend to preserve concurrency. But when the transformations become less immediate, one uses tricks that unfortunately destroy concurrency.

Therefore it is not surprising that the first works about formal comparisons of the expressiveness of these models do not consider preservation of concurrency. In [6], a structural transformation from TPN to NTA is defined. This transformation builds a timed automaton per transition of the TPN and preserves weak timed bisimilarity. In the other direction, [4] shows that there exist timed automata that are not weakly timed bisimilar to any TPN. In [5], the authors propose a translation from bounded timed-arc Petri nets (another variant of Petri nets extended with time) to NTA, based on the decomposition of the net in sequential components that communicate through handshake synchronizations (in the UPPAAL style). In [18], another timed extension of Petri nets with intervals on arcs is considered. For a matter of compositional properties (which is close to our problem), their Petri nets are translated to timed automata enriched with an ad-hoc mechanism of deadlines, which hides the communications between components that would be necessary to implement it.

Here we focus on the preservation of concurrency. Since both TPNs and NTA were designed to model distributed systems, we consider that not only their sequential behavior as timed transition systems is relevant, but also their distributed behavior. For that reason, we take into account the distribution of actions over a set of processes, each process representing a component which has its own alphabet of actions. When an action belongs to several processes, it represents a synchronization, otherwise it is a local action. In the untimed context, Mazurkiewicz traces [9] are defined using an independence relation that arises naturally from this distribution of actions.

However, in the presence of time such relation would have less nice properties because even actions that occur in two totally independent processes may be ordered by their occurrence time. These orders induced by causality and by the time stamping of events appear in [1], where timed MSCs (Message Sequence Charts) and MSCs with timing constraints are considered and in [2] where the authors consider distributed timed automata with independently evolving clocks. In [15, 17], an independence relation is defined among the actions of a timed automaton using a diamond property that takes time into account. This relation is used to define partial order reduction techniques that avoid the combinatorial explosion in the analysis of timed automata. Anyway the time constraints make this independence relation very restrictive. Therefore it cannot be seen as a general concurrency relation for timed systems.

In this article, we propose to study the distribution of actions as a first step towards the understanding of what concurrency means in timed systems. For this purpose, we define a notion of timed traces as a partial order representation of executions of our models for real-time distributed systems. We will use them as an alternative to timed words, to represent the executions of either an NTA or a TPN where processes have been identified.

Then we propose a structural transformation from 1-bounded TPNs to NTA which preserves the distribution of actions. That is we require that if the TPN represents the product of several components (called processes), then each process should have its counterpart as one timed automaton in the resulting NTA. To this end, we first discuss how to identify processes in a TPN; then the struc-

ture of each process gives a natural transformation into an automaton, and we show how to equip this automaton with timed constraints so that the resulting NTA preserves the timed traces. We show that this transformation is possible in general only if we allow the automata to read the states of their neighbors, which we interpret as a dependency between the processes, that was hidden in the TPN.

This paper is organized as follows. Section 2 presents centralized timed systems, and Sect. 3 presents distributed timed systems and introduces timed traces. In Sect. 4, we recall how to identify the processes in a Petri net. Lastly, in Sect. 5, we propose a translation from a TPN to a timed bisimilar NTA with the same distribution of actions.

2 Centralized timed systems

Timed automata are a popular formalism for modeling centralized timed systems. Their runs can be described by timed words, and their semantics can be expressed as a timed transition system.

2.1 Basics

Definition 1 (Timed Words)**.** A *timed word* w over a finite alphabet Σ is a finite or infinite sequence $w = (a_0, d_0)(a_1, d_1)\dots(a_n, d_n)\dots$ s.t. for each $i \geq 0, a_i \in \Sigma$, $d_i \in \mathbb{R}_{\geq 0}$ and $d_{i+1} \geq d_i$ (the d_i's are absolute dates). △

A timed language over Σ is a set of timed words over Σ.

Definition 2 (Timed Transition System)**.** A *timed transition system* (TTS) is a tuple $S = (Q, q_0, A, \to)$ where

- Q is a set of states,
- $q_0 \in Q$ is the initial state,
- A is a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$,
- $\to \subseteq Q \times (A \cup \mathbb{R}_{\geq 0}) \times Q$ is a set of edges.

△

If $(q, e, q') \in \to$, we also write $q \xrightarrow{e} q'$.

An initial path of a TTS is a possibly infinite sequence of transitions $\rho = q_0 \xrightarrow{\tau_0} q'_0 \xrightarrow{a_0} \dots q_n \xrightarrow{\tau_n} q'_n \xrightarrow{a_n} \dots$. The timed word $w = (a_0, d_0)(a_1, d_1)\dots a_n, d_n)\dots$ is said to be *accepted* by the TTS if there exists an initial path ρ such that $d_i = \sum_{j=0}^{i} \tau_j$ for every $1 \leq i \leq k$.

Definition 3 (Timed Bisimulation)**.** Let $S_1 = (Q_1, q_1^0, A, \to_1)$ and $S_2 = (Q_2, q_2^0, A, \to_2)$ be two TTS and $\approx$ be a binary relation over $Q_1 \times Q_2$. We write $q \approx q'$ for $(q, q') \in \approx$. $\approx$ is a *timed bisimulation* relation between S_1 and S_2 if:

- $q_1^0 \approx q_2^0$,
- if $q_1 \xrightarrow{a}_1 q'_1$ with $a \in A \cup \mathbb{R}_{\geq 0}$ and $q_1 \approx q_2$, then $\exists q_2 \xrightarrow{a}_2 q'_2$ such that $q'_1 \approx q'_2$; conversely if $q_2 \xrightarrow{a}_2 q'_2$ with $a \in A \cup \mathbb{R}_{\geq 0}$ and $q_1 \approx q_2$, then $\exists q_1 \xrightarrow{a}_1 q'_1$ such that $q'_1 \approx q'_2$. △

2.2 Timed automata

The set $\mathcal{B}(C)$ of clock constraints over the set of clocks C is defined by the abstract syntax $g ::= x \bowtie k \mid g \wedge g$, where $x \in C$, $k \in \mathbb{N}$ and $\bowtie \in \{<, \leq, =, \geq, >\}$. Invariants are clock constraints of the form $g ::= x \leq k \mid x < k \mid g \wedge g$.

Definition 4 (Timed automaton [3])**.** A timed automaton (TA) is a tuple $\mathcal{A} = (L, \ell_0, C, \Sigma, E, Inv)$ where

- L is a finite set of *locations*,
- $\ell_0 \in L$ is the *initial* location,
- C is a finite set of *clocks*,
- Σ is a finite set of *actions*,
- $E \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is a set of *edges*,
- $Inv : L \to \mathcal{B}(C)$ assigns *invariants* to locations.

△

If $(\ell, g, a, r, \ell') \in E$, we also write $\ell \xrightarrow{g,a,r} \ell'$. For such an edge, ℓ is called the *source* location, g the *guard*, a the *action*, r the set of clocks to be *reset* and ℓ' the *target* location.

Semantics We denote by (ℓ, v) a *state* of a TA, where $\ell \in L$ is the current location and $v : C \to \mathbb{R}_{\geq 0}$ is a *clock valuation* that maps each clock to its current value. The pair (ℓ, v) is a legal state for the timed automaton only if the valuation v satisfies the invariant of location ℓ, denoted by $v \models Inv(\ell)$. The initial state is (ℓ_0, v_0), where v_0 maps each clock to 0. For each set of clocks $r \subseteq C$, the valuation $v[r]$ is defined by $v[r](x) = 0$ if $x \in r$ and $v[r](x) = v(x)$ otherwise. For each $d \in \mathbb{R}_{\geq 0}$, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d$ for each $x \in C$.

Let $\mathcal{A} = (L, \ell_0, C, \Sigma, E, Inv)$ be a TA. We define the TTS generated by $\mathcal{A}$ as $T(\mathcal{A}) = (S, s_0, \Sigma, \to)$, where:

- $S = \{(\ell, v) \in L \times (C \to \mathbb{R}_{\geq 0}) \mid v \models Inv(\ell)\}$,
- $s_0 = (\ell_0, v_0)$,
- $\to \in S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is defined by
  - Action step: $(\ell, v) \xrightarrow{a} (\ell', v')$ iff $\exists (\ell \xrightarrow{g,a,r} \ell') \in E$, $v \models g$, $v' = v[r]$ and $v' \models Inv(\ell')$,
  - Time delay step: $\forall d \in \mathbb{R}_{\geq 0}, (\ell, v) \xrightarrow{d} (\ell, v + d)$ iff $\forall d' \in [0, d], v + d' \models Inv(\ell)$.

A run of a TA $\mathcal{A}$ is a path in $T(\mathcal{A})$ starting in s_0 where continuous and discrete transitions alternate. A timed word is accepted by $\mathcal{A}$ if it is accepted by $T(\mathcal{A})$.

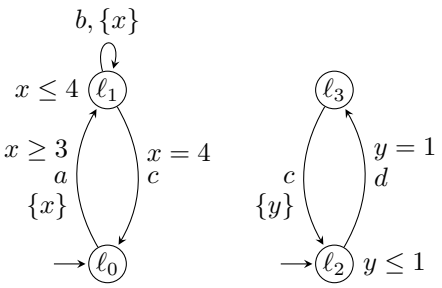

Figure 1: A network of timed automata.

3 Distributed timed systems

Distributed timed systems are systems with several components (or processes) that may perform local actions or synchronize with each other. We focus on two models for such systems: networks of timed automata and one of the variants of Petri nets extended with time, called time Petri nets, introduced in [16]. We first present the sequential semantics of these systems, as it is usually done. Then we define a partial order semantics which reflects the distribution of actions, as an alternative to timed words.

3.1 Networks of timed automata

A network of timed automata (NTA) is a parallel composition of n timed automata $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$, with $\mathcal{A}_i = (L_i, \ell_i^0, C_i, \Sigma_i, E_i, Inv_i)$ (see Fig. 1). We denote by $C = \bigcup_i C_i$ the set of clocks and $\Sigma = \bigcup_i \Sigma_i$ the set of action names. Clocks and action names may be shared.

Sequential semantics The set of synchronizations *Sync* is defined as the set of $(e_1, \ldots, e_n) \in (E_1 \cup \{\bullet\}) \times \cdots \times (E_n \cup \{\bullet\}) \setminus \{(\bullet, \ldots, \bullet)\}$ such that the same label a is attached to all the edges $e_i \neq \bullet$, and for all i such that $e_i = \bullet$, $a \notin \Sigma_i$. For any $s = (e_1, \ldots, e_n) \in Sync$, $I_s = \{i \in [1..n] \mid e_i \neq \bullet\}$ denotes the indices of the automata that are concerned by the synchronization.

We denote by $(\vec{\ell}, v)$ a state of an NTA, where $\vec{\ell} \in L_1 \times \cdots \times L_n$ is the vector of current locations and v is a clock valuation. The semantics of the NTA $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ can be described as the timed transition system $(S, s_0, \Sigma, \to)$ such that:

- $S = \{(\vec{\ell}, v) \in (L_1 \times \cdots \times L_n) \times (C \to \mathbb{R}_{\geq 0}) \mid v \models \bigwedge_i Inv_i(\ell_i)\}$,
- $s_0 = (\vec{\ell_0}, v_0)$ with $\forall x \in C, v_0(x) = 0$,
- $\to \in S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is defined by
  - Action step: $(\vec{\ell}, v) \xrightarrow{a} (\vec{\ell'}, v')$ iff
    - $\exists s = (e_1, \ldots, e_n) \in Sync$ s.t. $\forall i \leq n$, if $a \notin \Sigma_i, \ell_i' = \ell_i$ and $e_i = \bullet$, otherwise $e_i = (\ell_i, g_i, a, r_i, \ell_i')$

  - $v \models \bigwedge_{i\in I_s} g_i$, $v' = v[\bigcup_{i\in I_s} r_i]$, and $v' \models \bigwedge_i Inv_i(\ell'_i)$
- Time delay step: $\forall d \in \mathbb{R}_{\geq 0}, (\vec{\ell}, v) \xrightarrow{d} (\vec{\ell}, v+d)$ iff $\forall d' \in [0,d], v+d' \models \bigwedge_i Inv_i(\ell_i)$.

Local and extended syntaxes We call *local syntax* the common syntax in which clocks are local i.e. can be read and reset by only one automaton. Thus, invariants are of the form $g ::= x \leq k \mid x < k \mid g \wedge g$, as defined in 2.2.

We define an *extended syntax* (that will be used in Sec. 5) in which clocks can be read by any automaton, and invariants are of the form $g ::= x \leq k \mid x < k \mid g \wedge g \mid \ell \mid g \vee g$. The two last constructors are not standard. In an invariant, "ℓ" is true if ℓ is a current location, that is, invariants are evaluated according to the state of the system (current locations and valuation) and not only to the valuation. We denote by $\mathcal{B}(C, L)$ the set of such constraints over the set of clocks C and the set of locations L.

Other operators that do not extend the expressiveness of g can be used, such as the negation of a location: $\neg\ell_i \equiv \bigvee_{\ell\in L_i\setminus\{\ell_i\}} \ell$, the implication: $\ell \Rightarrow (x \leq k) \equiv \neg\ell \vee (x \leq k)$, and the minimum of a set of clocks: $min_{i\in I}(x_i) \leq k \equiv \bigvee_{i\in I}(x_i \leq k)$.

This extended syntax does not change the expressiveness w.r.t. the sequential semantics. But we will show in Sect. 5 that, if we consider the *distributed* timed language (see subsection 3.3), the extended syntax improves the expressiveness of the NTA.

3.2 Time Petri nets

Definition 5 (Petri Net)**.** A *Petri net* is a tuple (P, T, F, M_0) where P and T are two disjoint sets, called set of *places* and set of *transitions*, $F \subseteq (P\times T)\cup(T\times P)$ is the set of *arcs* connecting places and transitions such that $\forall t \in T, \exists p \in P$ s.t. $(p,t) \in F$, and $M_0 \subseteq P$ is the *initial marking*. △

Definition 6 (Time Petri Net [16])**.** A *time Petri net* (TPN) is a tuple $(P, T, F, {}_0, efd, lfd)$ where (P, T, F, M_0) is a Petri net and $efd : T \to \mathbb{R}$ and $lfd : T \to \mathbb{R}\cup\{\infty\}$ associate an *earliest firing delay* $efd(t)$ and a *latest firing delay* $lfd(t)$ with each transition t. △

For $x \in P \cup T$, we define the pre-set of x as ${}^\bullet x = \{y \mid (y,x) \in F\}$ and the post-set of x as $x^\bullet = \{y \mid (x,y) \in F\}$. Given a set $X \subseteq P \cup T$, we define ${}^\bullet X = \bigcup_{x\in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x\in X} x^\bullet$.

Sequential semantics A marking M of a TPN is a subset of P (we consider 1-bounded TPNs). A state of a TPN is given by (M, ν) where M is a marking and $\nu : T \to \mathbb{R}_{\geq 0}$ is a valuation such that each value $\nu(t)$ is the elapsed time since the last time transition t was enabled. ν_0 is the initial valuation with $\forall t \in T, \nu_0(t) = 0$. A transition t is *enabled* in a marking M iff ${}^\bullet t \subseteq M$. For 1-bounded TPNs, if a transition t is enabled in a reachable state (M, ν), then $t^\bullet \cap (M \setminus {}^\bullet t) = \emptyset$.

A transition t' is newly enabled by the firing of t from marking M if it is not enabled by $M\setminus{}^\bullet t$ (intermediate marking) and it is enabled by $M' = (M\setminus{}^\bullet t)\cup t^\bullet$ (reached marking). Formally:

$$\uparrow enabled(t', M, t) \Leftrightarrow ({}^\bullet t' \subseteq M') \wedge \left({}^\bullet t' \not\subseteq (M\setminus{}^\bullet t)\right)$$

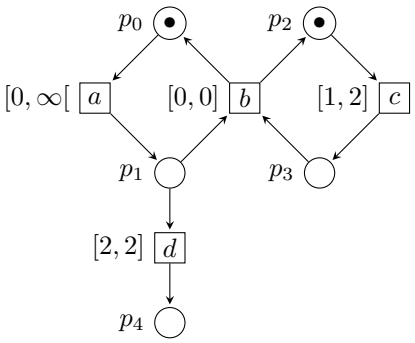

Figure 2: A time Petri net. Places are represented by circles and transitions are represented by boxes.

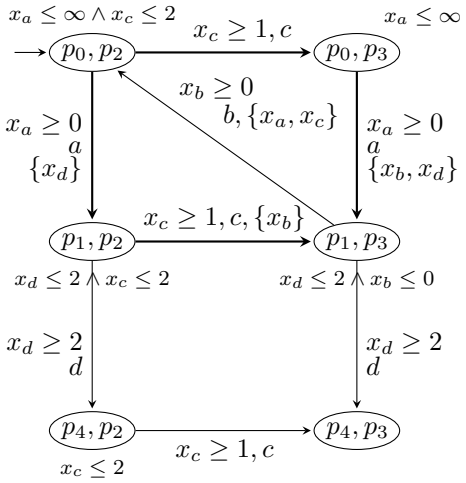

Figure 3: The semantics of the TPN of Fig. 2 as a timed automaton.

For the firing delays of a transition, we use the strong semantics: t can fire if it is enabled and $\nu(t) \geq efd(t)$ and has to fire before $\nu(t)$ overtakes $lfd(t)$.

With these rules, we are able to define the semantics of the TPN $(P, T, F, M_0, , lfd)$ as the TA $(L, \ell_0, C, \Sigma, E, Inv)$, called marking TA and introduced in [10], such that:

- $L \subseteq 2^P$ is the set of all reachable markings,
- $\ell_0 = M_0$ is the initial marking,
- C is a finite set of clocks s.t. each clock x_t is associated to one transition t,

- $\Sigma = T$ is the finite set of actions,
- $E = \big\{(M, g, t, r, M') \mid M' = (M \backslash {}^\bullet t) \cup t^\bullet, g \equiv x_t \geq efd(t), r = \{x_{t'} \mid \uparrow enabled(t', M, t)\}\big\}$ is the set of edges,
- $Inv : L \to \mathcal{B}(C)$ assigns invariants to markings s.t. $Inv(M) = \bigwedge_{\bullet t \subseteq M}(x_t \leq lfd(t))$.

A timed word is accepted by a TPN iff it is accepted by its marking TA. Figure 3 shows the marking TA of the TPN presented in Fig. 2. We note that concurrency is not explicit in this automaton, although we can observe a diamond that shows the possible interleavings between actions a and c.

This interpretation of a TPN as a TA gives naturally its sequential semantics.

A sequential semantics is not adapted to describe distributed systems because the information about the distribution of actions is lost. We aim at identifying *processes* in such systems, and defining their semantics with new notions such as timed traces and distributed timed languages that reflect the distribution of actions. In an NTA, it is clear that a process is an automaton and we can also identify processes in a TPN (see Sect. 4).

3.3 Timed traces

Once processes have been identified, we can describe the runs of distributed timed systems as *timed traces*, where each action is associated with the processes that have performed it. Actions may be local or shared (synchronizations).

Definition 7 (Timed Trace, Distributed Timed Language)**.** A *timed trace* over the alphabet Σ, and the set of processes $\Pi = (\pi_1, \ldots, \pi_n)$ is a tuple $\mathcal{W} = (E, \preccurlyeq, \lambda, t, proc)$ where:

- E is a set of events,
- $\preccurlyeq \subseteq (E \times E)$ is a partial order over E,
- $\lambda : E \to \Sigma$ is a labeling function,
- $t : E \to \mathbb{R}_{\geq 0}$ maps each event to a date and is such that, if $e_1 \preccurlyeq e_2$, then $t(e_1) \leq t(e_2)$;
- $proc : \Sigma \to 2^\Pi$ maps each action to a subset of Π,

and such that for any i in $[1..n]$, $\preccurlyeq_{|\pi_i} = \preccurlyeq \cap (E_i \times E_i)$ is a *total* order on E_i, where $E_i = \{e \in E \mid \lambda(e) \in \Sigma_i\}$, and $\Sigma_i = \{\sigma \in \Sigma \mid \pi_i \in proc(\sigma)\}$ is the alphabet of π_i.

A *distributed timed language* is a set of timed traces. △

Figure 4 gives a representation of a timed trace. An event $e \in E$ is denoted by $(\lambda(e), t(e))$ and events are ordered along one process from the top to the bottom of the line, but two events on different processes may not be ordered. For example, $(a, 4)$ and $(d, 1)$ are not ordered but $(b, 4) \preccurlyeq (d, 9)$ because $(c, 8)$ takes them apart by transitivity.

Given an accepted timed word $w = (a_1, d_1) \ldots (a_n, d_n) \ldots$ and the distribution of actions *proc* over the automata, we can build an accepted timed trace for an NTA: $E = \{e_1, \ldots, e_n, \ldots\}$, λ and t are such that, $\forall i, \lambda(e_i) = a_i$ and $t(e_i) = d_i$, and $\preccurlyeq$ is the transitive closure of the relation $\preccurlyeq'$ define as $e_i \preccurlyeq' e_j \Leftrightarrow \big(i \leq j \wedge proc(\lambda(e_j)) \cap proc(\lambda(e_j)) \neq \emptyset\big)$.

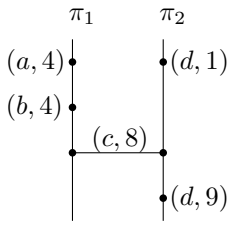

Figure 4: A timed trace representing an accepted run for the NTA of Fig. 1. One possible associated timed word is $(d,1)(a,4)(b,4)(c,8)(d,9)$.

4 S-subnets as processes for Petri nets

Identifying processes in a TPN is not as immediate as in an NTA. But, in practice, when a system is modeled as a TPN, the designer knows its physical structure and builds the TPN as a composition of components that model the subsystems. Anyway, if a TPN is given without its decomposition, these components can be identified.

We first define S-subnets as the processes of a Petri net, and the decomposition of a Petri net in S-subnets. Then we show how we can find this decomposition. We borrow the main definitions from [8], where the authors give a method (introduced in [11]) to decompose a live and bounded free-choice net in such components and we adapt this method to decompose more general nets.

4.1 Decomposition in S-subnets

Since the notion of process involves only the structure and does not depend on any time property, in this section, we consider only the structure of a Petri net: a net is denoted by (P,T,F) where P is the set of places, T is the set of transitions, and $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs.

A net (P,T,F) is an *S-net* if $\forall t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$.

An S-net is an automaton where locations are places and edges are transitions. We want to decompose a net N in S-nets that cover the net. To do so, we introduce the notion of *S-subnet*.

A net (P',T',F') is a *subnet* of the net (P,T,F) if $P' \subseteq P$, $T' \subseteq T$ and $F' = F \cap \big((P' \times T') \cup (T' \times P')\big)$.

We say that the subnet (P',T',F') of N is *P-closed* if $T' = {}^\bullet P' \cup P'^\bullet$. That is, any transition connected to a place which is in the subnet is also in the subnet. The subnet of N *generated* by a set of places P' is the P-closed subnet (P',T',F') of N.

Definition 8 (S-subnet)**.** An S-subnet of a net N is a *P-closed subnet* $N' = (P',T',F')$ of N such that N' is an *S-net*. △

The net $N = (P,T,F)$ is *decomposable* in S-subnets iff there exists a set of S-subnets $\{N_1,\ldots,N_n\}$ with $N_i = (P_i,T_i,F_i)$, such that $\bigcup_i P_i = P$. In this case, the set of S-subnets is called a *cover* of N (and $\bigcup_i T_i = T$ because the S-subnets are P-closed).

Note that the notion of S-subnet generalizes the notion of S-component presented in [8] because we do not impose that the subnet is strongly connected.

Definition 9 (Incidence matrix)**.** Let N be the net (P, T, F). The incidence matrix $\mathbf{N} : (P \times T) \to \{-1, 0, 1\}$ of N is defined by

$$\mathbf{N}(p,t) = \begin{cases} -1 & \text{if } (p,t) \in F \text{ and } (t,p) \notin F \\ 1 & \text{if } (p,t) \notin F \text{ and } (t,p) \in F \\ 0 & \text{otherwise} \end{cases}$$

△

An incidence matrix is given in Table 1. The entry $\mathbf{N}(p,t)$ corresponds to the change of the marking of the place p caused by the occurrence of transition t. Hence, if t is fired from marking M, the new marking is $M' = M + \mathbf{t}$, where $\mathbf{t}$ is the column vector of $\mathbf{N}$ associated to t.

Definition 10 (S-invariant [14])**.** An S-invariant of a net N is an integer-valued solution of the equation $X \cdot \mathbf{N} = \mathbf{0}$. △

From the definition of incidence matrix it follows that a mapping $I : P \to \mathbb{Q}$ is an S-invariant iff for every transition t holds $\sum_{p \in {}^\bullet t} I(p) = \sum_{p \in t^\bullet} I(p)$.

An S-invariant I of a net is called *semi-positive* if $I \geq \mathbf{0}$ and $I \neq \mathbf{0}$. The *support* of a semi-positive S-invariant I, denoted by $\langle I \rangle$, is the set of places p satisfying $I(p) > 0$. Every semi-positive S-invariant I satisfies ${}^\bullet\langle I \rangle = \langle I \rangle^\bullet$.

Proposition 11. *A Petri net (P, T, F) is decomposable in S-subnets iff there exists a set of S-invariants $\{X_1, \ldots X_n\}$ such that,*

- $\forall i \in [1..n], X_i : P \to \{0,1\}$ *(set of places),* *(1)*
- $\forall i \in [1..n], \forall t \in T, \sum_{p \in {}^\bullet t} X_i(p) = 1 \ \big(= \sum_{p \in t^\bullet} X_i(p)\big)$, *(2)*
- $\forall p \in P, \sum_i X_i(p) \geq 1$ *(the set covers the net).* *(3)*

To our knowledge such proof does not exist yet.

Proof. ($\Rightarrow$) Assume P is decomposable in S-subnets, then there exists a set of n S-subnets $N_i = (P_i, T_i, F_i)$, with $i \in [1..n]$, such that $\bigcup_i P_i = P$. We can choose n mappings $X_i : P \to \{0,1\}$ such that for each place p, $X_i(p) = 1$ if $p \in P_i$, and $X_i(p) = 0$ otherwise. Since N_i is an S-net, for each transition t, $|P_i \cap {}^\bullet t| = |P_i \cap t^\bullet| = 1$. Therefore, for each transition t, $\sum_{p \in {}^\bullet t} X_i(p) = 1$ and $\sum_{p \in t^\bullet} X_i(p) = 1$; that is, $\sum_{p \in {}^\bullet t} X_i(p) = \sum_{p \in t^\bullet} X_i(p)$ which defines an S-invariant. Moreover, $\forall p \in P, \sum_i X_i(p) \geq 1$ also holds because each place is in at least one subset of places.

($\Leftarrow$) Assume now that there exists a set of S-invariants $\{X_1, \ldots, X_n\}$ which satisfies the three conditions of Proposition 11. We show that the n subnets generated by each $\langle X_i \rangle$ with i in $[1..n]$, are S-subnets that cover N. We denote them by $N_i = (P_i, T_i, F_i)$, with $P_i = \langle X_i \rangle$ and $T_i = {}^\bullet\langle X_i \rangle = \langle X_i \rangle^\bullet$. By construction, N_i is a P-closed subnet of N. Moreover, since for each place p, $X_i(p) \in \{0,1\}$, $p \in \langle X_i \rangle$ implies that $X_i(p) = 1$, and $p \notin \langle X_i \rangle$ implies that $X_i(p) = 0$. That is, for each transition t, $|{}^\bullet t \cap P_i| = |{}^\bullet t \cap \langle X_i \rangle| = \sum_{p \in {}^\bullet t} X_i(p) = 1$ and, in the same way, $|t^\bullet \cap P_i| = 1$. Hence N_i is an S-net. Lastly, the n S-subnets cover the net because for each place p, $\sum_i X_i(p) \geq 1$, which implies that there exists i in $[1..n]$ such that $p \in \langle X_i \rangle$, that is $\bigcup_i \langle X_i \rangle = P$. □

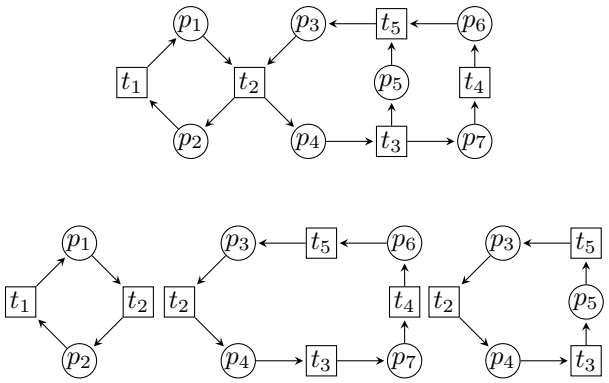

Figure 5: A net which is decomposable in S-subnets, and its decomposition.

Note that, if N is connected, every X_i is minimal (w.r.t. set inclusion). When the net is decomposable, there exists a set $\{I_1, \dots I_k\}$ of minimal S-invariants that covers the net and such that for each $1 \leq i \leq k$, if I_i is removed from the set, then the net is no longer covered (the set is minimal). This set gives a decomposition of the net in the S-subnets generated by the minimal S-invariants. Note that this decomposition is not unique and that a place may be shared by several S-subnets.

The number of tokens in an S-subnet is constant. Thus, an S-subnet initially marked with one token represents an automaton where the active location is the marked place. Such subnet is called a *process*. If the S-subnet is initially marked with m tokens, then it corresponds to m processes with the same structure but not necessarily starting in the same place, and these processes do not synchronize with each others. To simplify, we only consider 1-bounded TPNs.

4.2 An example of decomposition

We want to decompose the net shown in Fig. 5. To this purpose, we determine its S-invariants that satisfy conditions 1 and 2. If they cover the net (condition 3), then the net is decomposable.

Table 1: The incidence matrix of the net of Fig. 5

$\mathbf{N} =$

	t_1	t_2	t_3	t_4	t_5
p_1	1	−1	0	0	0
p_2	−1	1	0	0	0
p_3	0	0	0	0	1
p_4	0	0	−1	0	0
p_5	0	0	1	0	−1
p_6	0	0	0	1	−1
p_7	0	0	1	−1	0

With the incidence matrix given in Table 1, we obtain the following S-invariants: $X_1 = [1\ 1\ 0\ 0\ 0\ 0\ 0]$, $X_2 = [0\ 0\ 1\ 1\ 0\ 1\ 1]$, and $X_3 = [0\ 0\ 1\ 1\ 1\ 0\ 0]$.

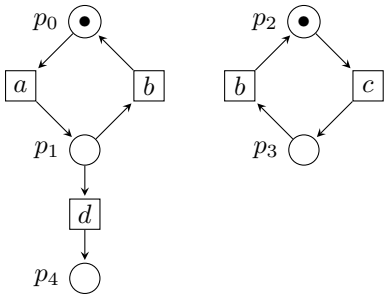

Figure 6: The processes of the TPN of Fig. 2.

These S-invariants satisfy the three conditions of Prop. 11, they are minimal and they form a minimal set. Therefore the net is decomposable in the three S-subnets generated by the sets of places $\{p_1, p_2\}$, $\{p_3, p_4, p_6, p_7\}$, and $\{p_3, p_4, p_5\}$, see Fig. 5.

4.3 Algorithms and size of the decomposition

Some algorithms for the computation of minimal S-invariants can be found in [7] where they are called p-semiflows.

The number of places in the decomposition is equal to $\sum_i |P_i|$ and is at most $|P|^2$ because a place may be shared by several components and no more than $|P|$ components are needed to cover the net. And the number of transitions is $\sum_i |T_i|$ and is at most $|T| \times |P|$ for the same reason. But these upper bounds are pessimistic since generally there are fewer components and few places and transitions are duplicated in all components.

5 Translation from time Petri net to network of timed automata

A TPN can be translated in a TA which accepts the same timed words (see Fig. 3). But we would like to translate it in an NTA which accepts the same timed traces. In this section, we propose a structural translation from a TPN to an NTA, based on the decomposition in processes.

5.1 Procedure

We first look at the untimed net to determine the processes and we check that each subnet is initially marked with one token. We get the subnets shown in Fig. 6. This translation involves three more steps:

1. Each subnet is translated in an automaton preserving its structure (places become locations and transitions become edges). Each edge is labeled with the name of the corresponding transition.

2. Time is added by providing each automaton with a clock x_i. This clock is reset on each edge. The idea is that the value of x_i gives the time elapsed

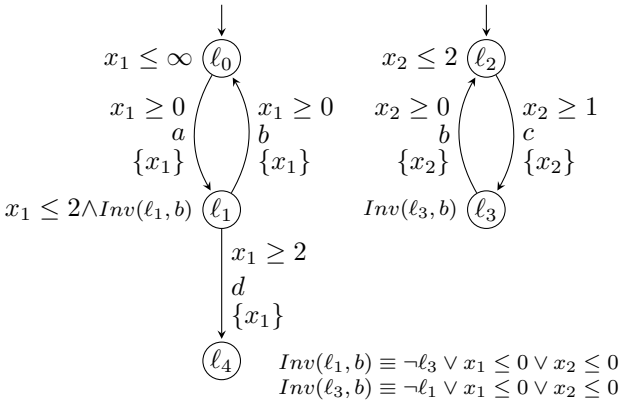

Figure 7: The resulting NTA.

in the current location. On each edge, if $[a, b]$ is the firing interval of the corresponding transition, we add a guard $x_i \geq a$, and if the transition is not shared, we add an invariant $x_i \leq b$ on the source location.

3. Then, we have to deal with the synchronizations (transitions with several input places). Such transitions have to fire if they are enabled and their latest firing delay is reached. On our example, see Fig. 7, we can stay in $\vec{\ell} = (\ell_1, \ell_3)$ as long as $min(v(x_1), v(x_2)) \leq 0$ (because $min(v(x_1), v(x_2))$ is the elapsed time since b was enabled and $lfd(b) = 0$). Thus, we add $Inv(\ell_1, b) \equiv \ell_3 \Rightarrow (x_1 \leq 0 \vee x_2 \leq 0) \equiv \neg\ell_3 \vee (x_1 \leq 0 \vee x_2 \leq 0)$ and $Inv(\ell_3, b) \equiv \ell_1 \Rightarrow (x_1 \leq 0 \vee x_2 \leq 0) \equiv \neg\ell_1 \vee (x_1 \leq 0 \vee x_2 \leq 0)$ in the invariants of ℓ_1 and ℓ_3 (actually we only need to add this "global" invariant to the invariant of one of the source locations concerned by the synchronization).

Formally, a TPN $\mathcal{N} = (P, T, F, M_0, efd, lfd)$ with n processes can be translated in the NTA $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$ with $\forall i \in [1..n], \mathcal{A}_i = (P_i, \ell_i^0, C, \Sigma_i, E_i, Inv_i)$ such that:

- P_i (places of the i^{th} subnet) is the set of locations,
- ℓ_i^0 s.t. $\{\ell_i^0\} = P_i \cap M_0$ is the initial location,
- $C = \{x_1, \ldots, x_n\}$ is the set of clocks,
- $\Sigma_i = T_i$ (transitions of the i^{th} subnet) is the set of actions,
- $E_i \subset (P_i \times \mathcal{B}(C) \times T_i \times 2^C \times P_i)$ such that $E_i = \left\{(p, g, t, r, p') \mid p \in {}^\bullet t \wedge p' \in t^\bullet, g \equiv x_i \geq efd(t), r = \{x_i\}\right\}$ is the set of edges,
- $Inv_i : P_i \to \mathcal{B}(C, P)$ assigns invariants to locations s.t. $\forall p \in P_i, Inv_i(p) \equiv \bigwedge_{t \in p^\bullet} Inv(t)$,
  where $Inv(t) \equiv (\bigwedge_{p' \in {}^\bullet t} p') \Rightarrow \min_{k \in I_t}(x_k) \leq lfd(t)$ with $I_t = \{i \in [1..n] \mid t \in T_i\}$ the set of indices of the subnets that contain t.

Here, we use the extended syntax (see subsection 3.1): automaton $\mathcal{A}_i$ can read the clocks of the other automata, but does not reset them and it can also read the current location of the other automata in its invariants. $Inv_i(p)$ makes sure that we cannot overtake the latest firing delay of an enabled transition which is in the post-set of p.

5.2 Size of the network of timed automata

Once the decomposition is computed, we directly have the structure of the timed automata. Thus the NTA has at most $|P|^2$ locations and $|T| \times |P|$ edges (see Subsection 4.3). The number of edges is exactly $\sum_{t\in T} |I_t|$.

Then, the timing information is provided by as many clocks as processes, that is at most $|P|$ clocks. There is one clock comparison on each edge, because the guards are of the form $x_i \geq lfd(t)$. Moreover, each $Inv(t)$ contains $|I_t|$ clock comparisons (because the min ranges over $|I_t|$ clocks). $Inv(t)$ can be attached only to one of the input places of t because a state is legal as long as the valuation satisfies all the invariants of the current locations, thus, if t is enabled and one of its input places carries $Inv(t)$, $lfd(t)$ cannot be overtaken. Therefore, if we attach each $Inv(t)$ to only one of the input places of t, we have $\sum_{t\in T} |I_t|$ clock comparisons in the invariants. To conclude, the size of the timing information given by the clock comparisons is proportional to the number of edges.

Proposition 12. *The initial 1-bounded time Petri net, $\mathcal{N}$ and the network of timed automata $\mathcal{S}$ which results from the translation have the same distributed timed language (are timed bisimilar with the same distributions of actions).*

Proof. For any i in $[1..n]$, we note $p_i = M \cap P_i$ the current location of automaton $\mathcal{A}_i$. We first show that,

$$v \models \bigwedge_{1\leq i\leq n} Inv_i(p_i) \Leftrightarrow \forall t \in T \text{ s.t. } {}^\bullet t \subseteq M, \nu(t) \leq lfd(t) \tag{1}$$

Indeed, by construction, $Inv_i(p_i) \equiv \bigwedge_{t\in p_i{}^\bullet} \big((\bigwedge_{p\in {}^\bullet t} p) \Rightarrow \min_{k\in I_t}(x_k) \leq lfd(t)\big)$. Thus, $v \models \bigwedge_{1\leq i\leq n} Inv_i(p_i)$ is equivalent to $\forall t \in T$ s.t. $({}^\bullet t \cap M \neq \emptyset) \wedge ({}^\bullet t \subseteq M)$, $\min_{k\in I_t}\big(v(x_k)\big) \leq lfd(t)$, (${}^\bullet t \cap M \neq \emptyset$ can be removed).

Moreover, by construction, for each enabled transition t, $\nu(t) = \min_{i\in I_t}\big(v(x_i)\big)$, that is

$$\forall t \text{ s.t. } {}^\bullet t \subseteq M, v \models \bigwedge_{i\in I_t} g_i(t) \Leftrightarrow \nu(t) \leq efd(t) \tag{2}$$

where $g_i(t)$ is the guard associated to the edge labeled by t in automaton $\mathcal{A}_i$.

Then we define a relation $\mathcal{R}$ between states of $\mathcal{S}$ and states of $\mathcal{N}$ as follows:

$$(M, v)\mathcal{R}(M, \nu) \Leftrightarrow \forall t \in T \text{ s.t. } {}^\bullet t \subseteq M, \nu(t) = \min_{i\in I_t}\big(v(x_i)\big)$$

Note that $\mathcal{R}$ is not a bijection because the clocks of the automata do not correspond to the clocks of the transitions, and a state of $\mathcal{N}$ may correspond to several states of $\mathcal{S}$. We want to show that $\mathcal{R}$ is a timed bisimulation.

We first observe that $(M_0, v_0)\mathcal{R}(M_0, \nu_0)$ and we show that, from any correspondent states, $(M, v)\mathcal{R}(M, \nu)$, the same executions are possible.

Delay step Assume that there exists $d \in R_{\geq 0}$ such that $(M, v) \xrightarrow{d} (M, v+d)$. Then, $\forall d' \in [0, d], v + d' \models \bigwedge_{1 \leq i \leq n} Inv_i(p_i)$. Equation (1) implies that $\nu + d'$ is an admissible valuation for marking M, and $(M, v+d)\mathcal{R}(M, \nu+d)$.

Similarly, if there exists $d \in R_{\geq 0}$ such that $(M, \nu) \xrightarrow{d} (M, \nu + d)$, then, $(M, v + d)$ is also an admissible state for $\mathcal{S}$ and $(M, v+d)\mathcal{R}(M, \nu+d)$.

Action step Assume now that there exists an action t such that $(M, v) \xrightarrow{t} (M', v')$, and I_t is the set of indices of the processes that perform t. Then, there exists $e = (e_1, \ldots, e_n) \in (E_1 \cup \{\bullet\}) \times \cdots \times (E_n \cup \{\bullet\})$ s.t. $\forall i \in [1..n]$,

$$\begin{cases} \text{if } i \notin I_t, \text{ then } e_i = \bullet \text{ and } p_i = p'_i \\ \text{otherwise, } e_i = (p_i, g_i, t, r_i, p'_i) \text{ s.t. } \begin{cases} p_i \in {}^\bullet t \wedge p'_i \in t^\bullet, \\ g_i \equiv x_i \geq efd(t), \\ r_i = \{x_i\} \end{cases} \end{cases}$$

and $v \models \bigwedge_{i \in I_t} g_i$, $v' = v[\bigcup_{i \in I_t} r_i]$, and $v' \models \bigwedge_i Inv_i(p'_i)$.

$(M, v)\mathcal{R}(M, \nu)$ implies that transition t is firable from (M, ν), because it is enabled (${}^\bullet t = \{p_i \mid i \in I_t\}$) and its firing delays are respected (because of (1) and (2)). This transition leads to state (M'', ν') s.t. $M'' = (M \backslash {}^\bullet t) \cup t^\bullet = M'$, and $\forall t' \in T, \nu'(t') = \begin{cases} 0 & \text{if } \uparrow enabled(t', M, t), \\ \nu(t') & \text{otherwise.} \end{cases}$

By construction, $\forall i \in [1..n], v'(x_i) = 0$ if $i \in I_t$, and $v'(x_i) = v(x_i)$ otherwise. That is, for each transition t', $\min_{i \in I_{t'}}\big(v'(x_i)\big) = 0$ if $I_{t'} \cap I_t \neq \emptyset$ and $\min_{i \in I_{t'}}\big(v'(x_i)\big) = \min_{i \in I_{t'}}\big(v(x_i)\big)$ otherwise.

Then, for each *enabled* transition t', we distinguish two cases:

1. t' is newly enabled by the firing of t from marking M ($\uparrow enabled(t', M, t)$ holds). That means that the last token to enable t' has been created by t, that is, $I_{t'} \cap I_t \neq \emptyset$. Therefore, $\nu'(t') = 0 = \min_{i \in I_{t'}}\big(v'(x_i)\big)$.

2. t' was enabled before the firing of t. That implies $I_{t'} \cap I_t \neq \emptyset$ (because there is one token by process and the tokens in ${}^\bullet t'$ have not been moved by t). Therefore, $\nu'(t') = \nu(t') = \min_{i \in I'_t}\big(v(x_i)\big) = \min_{i \in I_{t'}}\big(v'(x_i)\big)$.

Therefore, ν' is an admissible valuation for M' and $(M', v')\mathcal{R}(M', \nu')$.

Similarly, if there exists $t \in T$ such that $(M, \nu) \xrightarrow{t} (M', \nu')$ then, we can take synchronization t: $(M, v) \xrightarrow{t} (M', v')$, such that this synchronization is shared by the automata whose indices are in I_t and, for any i, $v'(x_i) = 0$ if $i \in I_t$ and $v'(x_i) = v(x_i)$ otherwise. That is, for any transition t', $\min_{i \in I_{t'}}\big(v'(x_i)\big) = 0$ if $I_t \cap I_{t'} \neq \emptyset$, and $\min_{i \in I_{t'}}\big(v'(x_i)\big) = \min_{i \in I_{t'}}\big(v(x_i)\big)$ otherwise. Therefore, if t' is enabled, $\min_{i \in I_{t'}}\big(v'(x_i)\big) = \nu'(t')$, and $(M', v')\mathcal{R}(M', \nu')$.

We have shown that $\mathcal{R}$ is a timed bisimulation between the TTS of $\mathcal{N}$ and $\mathcal{S}$. Moreover, there is a bijection between the processes of $\mathcal{N}$ and those of $\mathcal{S}$ and we have the same distribution of actions between the processes. Therefore, $\mathcal{N}$ and $\mathcal{S}$ accept the same distributed timed language. □

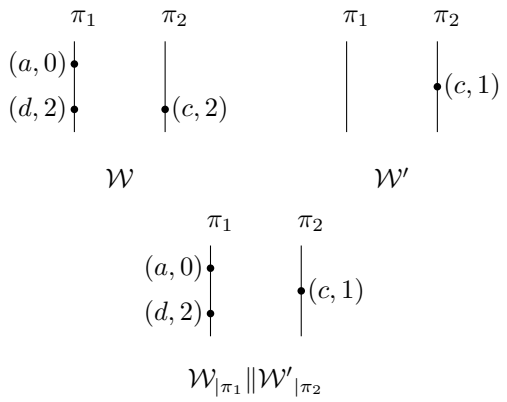

Figure 8: Two accepted timed traces and one non accepted timed trace for the TPN of Fig. 2.

5.3 Know thy neighbor!

Our translation produces a network of timed automata which accepts the same distributed timed language (and which is timed bisimilar). But we use an extended syntax (see subsection 3.1) in which each automaton can read the state (location and clock) of the other automata. We show that the use of this extended syntax is necessary.

Proposition 13. *Given a TPN $\mathcal{N}$ with its processes, in general, there does not exist any NTA $\mathcal{S}$ using the local syntax such that $\mathcal{N}$ and $\mathcal{S}$ have the same distributed timed language.*

For example, Fig. 8 shows two timed traces $\mathcal{W}$ and $\mathcal{W}'$ representing the beginning of two possible runs, without synchronization, for the TPN $\mathcal{N}$ of Fig. 2. Any NTA $\mathcal{S}$ using the local syntax and accepting $\mathcal{W}$ and $\mathcal{W}'$ would also accept the timed trace built by composing the projection of $\mathcal{W}$ onto π_1 and the projection of $\mathcal{W}'$ onto π_2 (see Fig. 8). But this timed trace is not accepted by $\mathcal{N}$.

To prove Prop. 13, we first give some definitions about timed traces, and a lemma that will be used in the proof.

Timed linearization and projection A *timed linearization* of a timed trace is a possible execution expressed as a timed word which respects both the causal order and the order imposed by the time stamping.

A timed trace $\mathcal{W}$ can be defined as a couple $(w, proc)$ where w is a timed linearization of $\mathcal{W}$, see Fig. 4.

The *projection* of a timed trace $\mathcal{W}$ onto process π_i, denoted by $\mathcal{W}_{|\pi_i}$ is defined as the projection of a linearization of $\mathcal{W}$, w, onto Σ_i, denoted by $w_{|\Sigma_i}$:

- if $w = \varepsilon$, then $w_{|\Sigma_i} = \varepsilon$

- if $w = (a, \theta) \cdot w'$,
  then $w_{|\Sigma_i} = \begin{cases} (a, \theta) \cdot w'_{|\Sigma_i} & \text{if } a \in \Sigma_i \\ w'_{|\Sigma_i} & \text{otherwise} \end{cases}$

Juxtaposition of timed words The *juxtaposition* of n timed words, $w_1 \parallel w_2 \parallel \cdots \parallel w_n$ is the timed trace over n processes, $\mathcal{W}$ such that for each i in $[1..n]$, if Σ_i denotes the set of actions that appear in w_i, then $\mathcal{W}_{|\Sigma_i} = w_i$.

We denote by $\mathcal{S}$ a network of n timed automata $(\mathcal{A}_1, \ldots, \mathcal{A}_n)$, and by $R_\theta(\mathcal{S})$ the set of all timed traces representing admissible runs of $\mathcal{S}$, without synchronization, and stopping at date θ.

Lemma 14. *Let $\mathcal{S}$ be a network of n timed automata that do not read the state of the other automata, then, for any timed traces $\mathcal{W}_1, \ldots, \mathcal{W}_n \in R_\theta(\mathcal{S})$ (not necessarily different), $\mathcal{W}_{1|\pi_1} \parallel \cdots \parallel \mathcal{W}_{n|\pi_n} \in R_\theta(\mathcal{S})$.*

Proof of Lemma 14. In θ, the automata have not yet synchronized, that is their runs stopping at date θ are independent, and they could have performed any other admissible sequence of actions, stopping at date θ, without synchronization. □

Proof of Prop. 13. Assume that the two automata corresponding to the two processes of the TPN $\mathcal{N}$ of Fig. 2 are not able to read the current location and the clock of the other automaton. Then, for any two timed traces $\mathcal{W}$ and $\mathcal{W}'$, representing two admissible runs without synchronization, stopping at date θ, the timed trace $\mathcal{W}_{|\pi_1} \parallel \mathcal{W}'_{|\pi_2}$ represents also an admissible run.

If we choose, as in Fig. 8, $\mathcal{W} = (w, proc)$ and $\mathcal{W}' = (w', proc)$, with $w = (a,0)(d,2)(c,2)$, $w' = (c,1)$ and $proc = \{(a,\pi_1), (b,\{\pi_1,\pi_2\}), (c,\pi_2), (d,\pi_1)\}$ (with $\theta = 2$), then $\mathcal{W}_{|\pi_1} \parallel \mathcal{W}'_{|\pi_2} = \big((a,0)(c,1)(d,2), proc\big)$ (see Fig. 8) should represent an admissible run for $\mathcal{S}$ and $\mathcal{N}$. Which is false because as soon as c has been performed, b must be performed immediately. Therefore, the local syntax must be extended. □

6 Loose ends and future works

6.1 TPNs with good decompositional properties

There are some simple cases when it is possible to translate a TPN in an NTA which uses the local syntax. For example, assume that for any transition t, there exists a place p in ${}^\bullet t$ which is always the last place to be marked among ${}^\bullet t$. Then, we chose to add $Inv(t)$ only in $Inv_i(p)$ (this can be done, as explained in the third step of the translation). By construction, $Inv(t) \equiv \big((\bigwedge_{p' \in {}^\bullet t} p') \Rightarrow \min_{k \in I_t}(x_k) \leq lfd(t)\big)$. In this case, $(\bigwedge_{p' \in {}^\bullet t} p')$ is always true in $Inv_i(p)$ – because if p is marked, then all places in ${}^\bullet t$ are marked – and $\min_{k \in I_t}(v(x_k)) = v(x_i) = \nu(t)$. Therefore, for any i in $[1..n]$ and for any place p in P_i, $Inv_i(p)$ can be expressed with the local syntax.

But these cases are restrictive, and it would be interesting to give a general characterization of these nets.

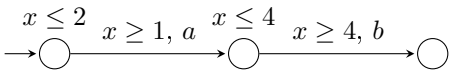

Figure 9: A TA that cannot be translated in a time S-net with one token.

6.2 Reverse translation

We can consider a reverse translation, from an NTA to a TPN. There exist translations, for example in [4] from a TA into a weak timed bisimilar TPN, but we want to preserve the distributed timed language, that is, when we translate an NTA into a TPN, we want to preserve the processes. This implies that we should be able to translate each automaton in a TPN which is an S-net with one token and then compose the obtained nets.

A time S-net with one token is less expressive than a TA with one clock because it can be translated in a TA with one clock which accepts the same timed language. Thus, it is less expressive than a TA with two clocks, according to [12]. We can even strengthen this by proving that some TA with one clock cannot be translated in finite time S-net with one token (see Prop. 15). Therefore, only a very small class of TA can be translated.

Proposition 15. *Time S-nets with one token are strictly less expressive than TA with one clock.*

Proof. Assume that the TA $\mathcal{A}$ of Fig. 9 can be translated in a finite time S-net with one token which accepts the same timed language, called $\mathcal{N}$. Then, in $\mathcal{N}$, finitely many states can be reached after having fired an a. We denote these states by $s_i = (\{p_i\}, \mathbf{0})$ with $i \in [1..n]$. The clocks of the enabled transitions have been reset.

Now, assume that we can reach s_i by firing a at some date θ_1. Then, the only possible continuation from s_i is to delay during $\delta_1 = 4 - \theta_1$ and fire b. That is, (a, θ_1) is the only possible way to reach s_i (otherwise, we would have another possible continuation from s_i).

Therefore, each state s_i can only be reached by executing a at one date θ_i, and from each s_i only one continuation is possible. This implies that $\mathcal{N}$ has a *finite number* of admissible runs whereas $\mathcal{A}$ has *infinitely* many. Thus, $\mathcal{A}$ cannot be translated in a time S-net with one token. □

6.3 Usability in practice

We have translated some example time Petri nets with the translation proposed in [6] and with our translation, and we have used Uppaal (see [13]) to check a reachability property on the resulting networks of timed automata.

Although our translation only works for bounded TPNs and does not always give a model in the Uppaal style (with handshake synchronizations), it generally produces networks with fewer automata, because their translation produces $n + 1$ automata for an initial net with n transitions. And we think that our translation gives an NTA which is more readable.

Regarding the number of clocks, we also generally have fewer clocks because we have one clock by process instead of one clock by transition. But as mentioned in [6], UPPAAL only considers the active clocks during the verification. In our case, in a given state, all clocks are active and with the translation of [6], the number of active clocks is equal to the number of enabled transitions in the corresponding marking (Theorem 3 in [6]). Therefore, we can have fewer active clocks if there are some conflicts.

6.4 Towards identification of concurrency in timed systems

This work is a starting point for a more advanced study of concurrency in timed systems. Indeed, concurrency in timed systems involves both causality and the time stamping of events. Transitions that appear as concurrent in an untimed model may not remain independent when time constraints are added. First, time constraints may easily force a temporal ordering between them. But, even worse, the occurrence of a transition may have consequences on apparently concurrent transitions due to time constraints: this is what happens in our TPN of Fig. 2 where firing c after delay 1 from marking $\{p_1, p_2\}$ prevents d from firing (because it forces b to fire earlier). In our translation, the necessity to allow the automata to read the states of their neighbors highlights these complex dependences between different processes.

References

[1] S. Akshay, B. Bollig, and P. Gastin, "Automata and logics for timed message sequence charts," in *FSTTCS'07*, ser. LNCS, vol. 4855. New Delhi, India: Springer, 2007, pp. 290–302.

[2] S. Akshay, B. Bollig, P. Gastin, M. Mukund, and K. Narayan Kumar, "Distributed timed automata with independently evolving clocks," in *CONCUR'08*, ser. LNCS, vol. 5201. Toronto, Canada: Springer, 2008, pp. 82–97.

[3] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183–235, 1994.

[4] B. Bérard, F. Cassez, S. Haddad, D. Lime, and O. H. Roux, "When are timed automata weakly timed bisimilar to time Petri nets?" *Theoretical Computer Science*, vol. 403, no. 2-3, pp. 202–220, 2008.

[5] J. Byg, K. Joergensen, and J. Srba, "An efficient translation of timed-arc Petri nets to networks of timed automata," in *ICFEM'09*, ser. LNCS, vol. 5885. Springer-Verlag, 2009, pp. 698–716.

[6] F. Cassez and O. H. Roux, "Structural translation from time Petri nets to timed automata," *Journal of Systems and Software*, 2006.

[7] J. M. Colom and M. Silva, "Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal p-semiflows," in *Proceedings of the 10th International Conference on Applications and Theory of Petri Nets*. London, UK: Springer-Verlag, 1991, pp. 79–112.

[8] J. Desel and J. Esparza, *Free choice Petri nets.* New York, USA: Cambridge University Press, 1995.

[9] V. Diekert, *The Book of Traces*, G. Rozenberg, Ed. River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1995.

[10] G. Gardey, O. H. Roux, and O. F. Roux, "State space computation and analysis of time Petri nets," *Theory Pract. Log. Program.*, vol. 6, no. 3, pp. 301–320, 2006.

[11] M. Hack, "Analysis of production schemata by Petri nets," Cambridge, USA, 1972.

[12] T. A. Henzinger, P. W. Kopke, and H. Wong-Toi, "The expressive power of clocks," in *ICALP*, 1995, pp. 417–428.

[13] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *STTT*, vol. 1, no. 1-2, pp. 134–152, 1997.

[14] K. Lautenbach, "Liveness in Petri nets," G.M.D., Bonn, Germany, Tech. Rep., July 1975.

[15] D. Lugiez, P. Niebert, and S. Zennou, "A partial order semantics approach to the clock explosion problem of timed automata," *Theor. Comput. Sci.*, vol. 345, no. 1, pp. 27–59, 2005.

[16] P. M. Merlin, "A study of the recoverability of computing systems," Ph.D. dissertation, University of California, Irvine, 1974.

[17] P. Niebert and H. Qu, "Adding invariants to event zone automata," in *FORMATS*, ser. LNCS, vol. 4202. Springer, 2006, pp. 290–305.

[18] J. Sifakis and S. Yovine, "Compositional specification of timed systems (extended abstract)," in *STACS'96*. London, UK: Springer-Verlag, 1996, pp. 347–359.

Centre de recherche INRIA Saclay – Île-de-France
Parc Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 Orsay Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399