Concurrent semantics for timed distributed systems
Livrable du projet ANR ImpRo (ANR-2010-BLAN-0317)

Sandie Balaguer, Thomas Chatain, and Stefan Haar

INRIA & LSV (CNRS & ENS Cachan)

This document defines a formalism called *timed traces* that aims at describing the concurrent semantics of various models for real-time distributed systems. This formalism, based on partial orders, provides an alternative to timed words and takes the distribution of actions into account.

To demonstrate the interest of timed traces, we equip two popular formalisms, 1-bounded time Petri nets (TPN) and networks of timed automata (NTA), with a concurrent sematics in terms of timed traces and we propose a translation from TPN to NTA. As opposed to previous approaches, our translation preserves timed traces rather than only timed words.

This work shows that, even in a real-time setting where the events look more or less totally ordered by time, partial orders can be relevant and represent the structural dependencies between events that come from the synchronizations and the locality of actions.

The document is a preliminary version of

> Sandie Balaguer, Thomas Chatain, and Stefan Haar. A concurrency-preserving translation from time Petri nets to networks of timed automata. *Formal Methods in System Design*, 40(3):330–355, 2012.

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

Sandie Balaguer · **Thomas Chatain** · **Stefan Haar**

Abstract Several formalisms to model distributed real-time systems coexist in the literature. This naturally induces a need to compare their expressiveness and to translate models from one formalism to another when possible. The first formal comparisons of the expressiveness of these models focused on the preservation of the sequential behavior of the models, using notions like timed language equivalence or timed bisimilarity. They do not consider preservation of concurrency. In this paper we define timed traces as a partial order representation of executions of our models for real-time distributed systems. Timed traces provide an alternative to timed words, and take the distribution of actions into account. We propose a translation between two popular formalisms that describe timed concurrent systems: 1-bounded time Petri nets (TPN) and networks of timed automata (NTA). Our translation 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.

Keywords concurrency · timed traces · time Petri nets · networks of timed automata · concurrency-preserving translation

S. Balaguer
LSV, ENS Cachan & CNRS, 61 avenue du Président Wilson, 94230 Cachan, France
and INRIA Saclay – Île-de-France, Orsay, France
E-mail: balaguer@lsv.ens-cachan.fr

T. Chatain
LSV, ENS Cachan & CNRS, 61 avenue du Président Wilson, 94230 Cachan, France
E-mail: chatain@lsv.ens-cachan.fr

S. Haar
LSV, ENS Cachan & CNRS, 61 avenue du Président Wilson, 94230 Cachan, France
and INRIA Saclay – Île-de-France, Orsay, France
E-mail: haar@lsv.ens-cachan.fr

1 Introduction

Techniques that aim at improving reliability and safety of automated systems have dramatically improved during the last thirty years (synthesis, model-checking, test, etc.). 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) [25] and networks of timed automata (NTA) [3]. These formalisms have different histories but 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.

Both formalisms are supported by a variety of simulation and verification tools, like UPPAAL [21], EPSILON [11] and KRONOS [8] for (networks of) timed automata, and ROMEO [15], TINA [7] and CPN TOOLS [19] for time Petri nets.

Because these tools have their specificities, several tools are often used for the design, analysis or verification of a single system. This usually requires to model the system in several formalisms, typically TPN and NTA. Therefore several transformations have been proposed; we observe 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 with 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 [10], 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, [5] shows that there exist timed automata that are not weakly timed bisimilar to any TPN. In [9], 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 [27], another timed extension of Petri nets with intervals on arcs is considered. In order to guarantee compositional properties, 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. This implies that, if a model represents a system that involves several components, then the model should be structured so that it is easy to identify each component, and a transformation should preserve this structure.

Our motivation for this is twofold: first, a transformation is much more readable if it preserves the components and yields a model that is closer to the real system; second, preserving the components avoids combinatorial explosion of the size of the model and makes it possible to use modular analysis based on the components or partial order techniques, which are crucial when one analyzes large distributed systems.

In order to formalize preservation of concurrency in the context of real-time models, 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 [14] 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 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 [24,26], 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. However, 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 define a notion of timed traces as a partial order representation of executions of our models for real-time distributed systems. They generalize timed words and represent the executions of either an NTA or a TPN on which processes have been identified. Then we define a structural transformation from 1-bounded TPNs to NTA which preserves timed traces. That is we require that if the TPN represents the product of several components (called processes), then each process has its counterpart as one timed automaton in the resulting NTA and the distribution of actions among the components is preserved.

To this end, we first discuss how to identify processes in a TPN. The structure of each process gives a natural transformation into an automaton. Then we focus on the timed constraints and show how to equip the automata with clocks, guards and invariants 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. Notice also that the decomposition of a PN into components is not always possible. However, we believe that most PNs that model real systems are decomposable. It is also known (see [13]) that well-formed free-choice nets are decomposable in strongly connected components.

This paper is organized as follows. Section 2 presents centralized timed systems, and Section 3 presents distributed timed systems and introduces timed traces. In Section 4, we recall how to identify the processes in a Petri net. Lastly, in Section 5, we propose a translation from a 1-bounded TPN to a timed bisimilar NTA with the same distribution of actions. Finally we discuss extensions and limitations of our translation, in particular we define conditions under which our translation can be adapted to avoid using shared clocks. This article is a revision and extension of the conference version of this work [4].

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 $(S, s_0, \Sigma, \rightarrow)$ where

- S is a set of states,
- $s_0 \in S$ is the initial state,
- Σ is a finite set of actions disjoint from $\mathbb{R}_{\geq 0}$,
- $\rightarrow \subseteq S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is a set of edges. △

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

An initial *path* of a TTS is a possibly infinite sequence of transitions $\rho = s_0 \xrightarrow{\tau_0} s'_0 \xrightarrow{a_0} \cdots s_n \xrightarrow{\tau_n} s'_n \xrightarrow{a_n} \cdots$. The timed word $w = (a_0, d_0)\dots(a_n, d_n)\dots$ 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 $i \geq 0$.

Definition 3 (Timed Bisimulation) Let $T_1 = (S_1, s_1^0, \Sigma, \rightarrow_1)$ and $T_2 = (S_2, s_2^0, \Sigma, \rightarrow_2)$ be two TTS and $\approx$ be a binary relation over $S_1 \times S_2$. We write $s_1 \approx s_2$ for $(s_1, s_2) \in \approx$. $\approx$ is a *timed bisimulation* relation between T_1 and T_2 if:

- $s_1^0 \approx s_2^0$,
- if $s_1 \xrightarrow{a}_1 s'_1$ with $a \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $s_1 \approx s_2$, then $\exists s_2 \xrightarrow{a}_2 s'_2$ such that $s'_1 \approx s'_2$; conversely if $s_2 \xrightarrow{a}_2 s'_2$ with $a \in \Sigma \cup \mathbb{R}_{\geq 0}$ and $s_1 \approx s_2$, then $\exists s_1 \xrightarrow{a}_1 s'_1$ such that $s'_1 \approx s'_2$. △

2.2 Timed automata

The set $\mathscr{B}(C)$ 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 $A = (L, \ell_0, C, \Sigma, E, Inv)$ 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*,
- $E \subseteq L \times \mathscr{B}(C) \times \Sigma \times 2^C \times L$ is a set of *edges*,
- $Inv : L \rightarrow \mathscr{B}(C)$ 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.

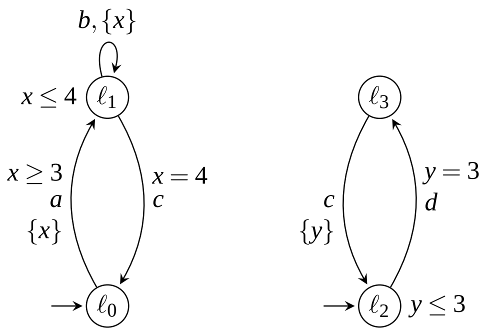


Fig. 1 A network of timed automata (initial locations are indicated by an arrow that is not rooted in any location)

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 $A = (L, \ell_0, C, \Sigma, E, Inv)$ be a TA. We define $T(A)$, the TTS generated by A as $T(A) = (S, s_0, \Sigma, \to)$, such that

- $S = \{(\ell, v) \in L \times (C \to \mathbb{R}_{\geq 0}) \mid v \models Inv(\ell)\}$,
- $s_0 = (\ell_0, v_0)$,
- $\to \in S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ 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 A is a path in $T(A)$ starting in s_0 where time delay steps and action steps alternate. A timed word is *accepted by A* if it is accepted by $T(A)$.

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 [25]. 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 over the processes, 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 $(A_1, \ldots, A_n)$, with $A_i = (L_i, \ell_i^0, C_i, \Sigma_i, E_i, Inv_i)$ (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 $(e_1,\dots,e_n) \in (E_1 \cup \{\bullet\}) \times \dots \times (E_n \cup \{\bullet\}) \setminus \{(\bullet,\dots,\bullet)\}$ 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 $s = (e_1,\dots,e_n) \in Sync$, $I_s = \{i \in [1..n] \mid e_i \neq \bullet\}$ denotes the indices of the automata that are concerned by the synchronization.

We denote by $(\vec{\ell},v)$ a state of an NTA, where $\vec{\ell} \in L_1 \times \dots \times L_n$ is the vector of current locations and v is a clock valuation. The semantics of the NTA $(A_1,\dots,A_n)$ can be described as the timed transition system $(S, s_0, \Sigma, \rightarrow)$ such that

- $S = \{(\vec{\ell},v) \in (L_1 \times \dots \times L_n) \times (C \rightarrow \mathbb{R}_{\geq 0}) \mid v \models \bigwedge_i Inv_i(\ell_i)\}$,
- $s_0 = (\vec{\ell}_0, v_0)$ with $\forall x \in C, v_0(x) = 0$,
- $\rightarrow \in S \times (\Sigma \cup \mathbb{R}_{\geq 0}) \times S$ is defined by
  - Action step: $(\vec{\ell},v) \xrightarrow{a} (\vec{\ell}',v')$ iff
    - $\exists s = (e_1,\dots,e_n) \in Sync$ s.t. $\forall i \in [1..n]$, 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 $v' \models \bigwedge_{i \in [1..n]} Inv_i(\ell'_i)$
  - Time delay step: $\forall d \in \mathbb{R}_{\geq 0}, (\ell,v) \xrightarrow{d} (\ell, v+d)$ iff $\forall d' \in [0,d], v+d' \models \bigwedge_i Inv_i(\ell_i)$.

Local vs extended syntax. We call *local syntax* the common syntax in which clocks are local, i.e. every clock 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 Subsection 2.2.

We define an *extended syntax* (that will be used in Sect. 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 $\mathscr{B}(C,L)$ 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 enhances the expressiveness of the NTA.

Although it is not generally allowed to share active locations in timed automata, there are several variants of timed automata that can handle such a feature. For example, timed automata can be extended with shared variables as in UPPAAL [21] and a boolean variable can be associated with each location and used to denote whether the location is enabled. In [20], the authors propose another variant, Timed Cooperating Automata, a parallel composition of sequential automata where the edges can be guarded with timing constraints of the form $q = \tau$ (location q is enabled for τ time units), $q[\tau]$ (location q is enabled for at least τ time units), $q\{\tau\}$ (location q is disabled for at most τ time units) or boolean combinations of these terms.

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*

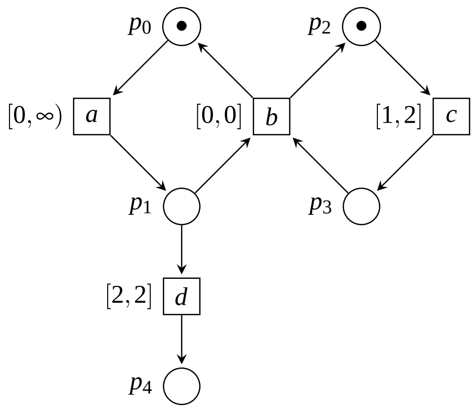


Fig. 2 A time Petri net (places are represented by circles and transitions are represented by boxes)

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*. $\triangle$

Definition 6 (Time Petri Net [25]) A *time Petri net* (TPN) is a tuple (P,T,F,M_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. $\triangle$

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 the pre-set and the post-set of X as ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ 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 $t^\bullet \cap (M \setminus {}^\bullet t) = \emptyset$.

When defining newly enabled transitions, we use the most common semantics, called intermediate semantics [6]: 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, we define the predicate $\uparrow\!enabled(t',M,t)$ as follows:

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

Lastly, 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 t *has to fire* before $\nu(t)$ overtakes $lfd(t)$.

With these rules, we are able to define the semantics of a TPN as a TA called marking TA and introduced in [16]. Indeed, the marking TA of the TPN (P,T,F,M_0,efd,lfd) is the TA $(L,\ell_0,C,\Sigma,E,Inv)$ such that

- $L \subseteq 2^P$ is the set of reachable markings,
- $\ell_0 = M_0$,
- each clock $x_t \in C$ is associated with one transition t,
- $\Sigma = T$,
- $E = \left\{(M,g,t,r,M') \mid M' = (M \setminus {}^\bullet t) \cup t^\bullet, g \equiv x_t \geq efd(t), r = \{x_{t'} \mid \uparrow\!enabled(t',M,t)\}\right\}$,
- for each reachable marking $M \in L$, $Inv(M) \equiv \bigwedge_{{}^\bullet t \subseteq M} \left(x_t \leq lfd(t)\right)$.

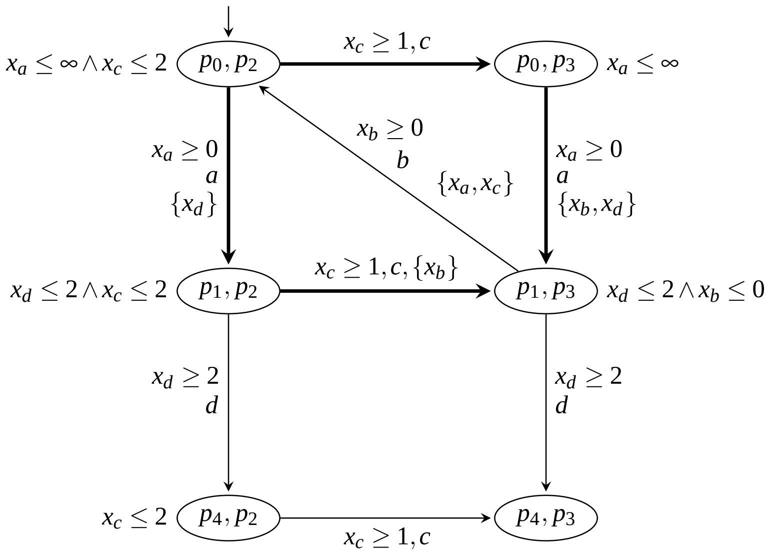

$x_a \leq \infty \wedge x_c \leq 2$
p_0, p_2
$x_c \geq 1, c$
p_0, p_3
$x_a \leq \infty$
$x_a \geq 0$ a $\{x_d\}$
$x_b \geq 0$ b $\{x_a, x_c\}$
$x_a \geq 0$ a $\{x_b, x_d\}$
$x_d \leq 2 \wedge x_c \leq 2$
p_1, p_2
$x_c \geq 1, c, \{x_b\}$
p_1, p_3
$x_d \leq 2 \wedge x_b \leq 0$
$x_d \geq 2$ d
$x_d \geq 2$ d
$x_c \leq 2$
p_4, p_2
$x_c \geq 1, c$
p_4, p_3

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

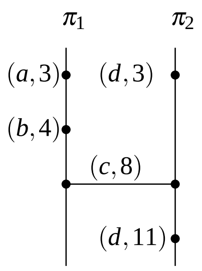


Fig. 4 A timed trace representing a run of the NTA of Fig. 1 (one possible associated timed word is $(d,3)(a,3)(b,4)(c,8)(d,11)$)

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, as it naturally gives the sequential semantics of the TPN, even though we can observe a diamond (bold edges) that shows the possible interleavings between actions a and c.

A sequential semantics is not adapted to describe distributed systems because the information about the distribution of actions over the different components is lost. We aim at identifying the components, that we call *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 each automaton is a process, and we will see in Sect. 4 that it is also possible to identify processes in a TPN.

3.3 Timed traces

Once processes have been identified, we can describe the runs of distributed timed systems as *timed traces*. With this definition, each action is associated with a set of processes that always perform it together and simultaneously, therefore it may be local or shared (synchronizations). *Events* (action occurrences) are partially ordered since two events on disjoint sets of processes may not be causally ordered.

Definition 7 (Timed Trace, Distributed Timed Language) A *timed trace* over the alphabet Σ and the finite set of processes $\Pi = (\pi_1, \dots, \pi_n)$ is a tuple $W = (E, \preccurlyeq, \lambda, \delta, proc)$ where

- E is a countable set of *events*,
- $\preccurlyeq \subseteq (E \times E)$ is a *partial order* over E such that, for any event e, the set $\{e' \in E \mid e' \preccurlyeq e\}$ is finite,
- $\lambda : E \to \Sigma$ is a labeling function,
- $\delta : E \to \mathbb{R}_{\geq 0}$ assigns a date to every event such that, if $e_1 \preccurlyeq e_2$, then $\delta(e_1) \leq \delta(e_2)$;
- $proc : \Sigma \to 2^{\Pi}$ is the *distribution of actions* that maps each action to a subset of Π,

and such that, for any i in $[1..n]$, $\preccurlyeq_{|\pi_i}$ is a *total* order on E_i, with the following definitions:

- $\Sigma_i = \{a \in \Sigma \mid \pi_i \in proc(a)\}$ denotes the alphabet of process π_i,
- $E_i = \{e \in E \mid \lambda(e) \in \Sigma_i\}$ denotes the set of events that occur on process π_i,
- $\preccurlyeq_{|\pi_i} = \preccurlyeq \cap (E_i \times E_i)$.

A *distributed timed language* is a set of timed traces. $\triangle$

Figure 4 gives a representation of a timed trace. Each process is represented by a vertical line, and each event is represented by a dot or dots connected by a horizontal line, depending on whether it occurs on one process or on several processes. Each event $e \in E$ is also labeled by the pair $(\lambda(e), \delta(e))$. Moreover, events are ordered along each process from the top to the bottom of the line, and we can see that events on different processes are not always ordered. For example, $(a,3) \preccurlyeq (b,4)$, $(b,4)$ and $(d,3)$ are not ordered, and $(b,4) \preccurlyeq (d,11)$ because $(c,8)$ takes them apart by transitivity.

Given an accepted timed word $w = (a_0,d_0)\dots(a_n,d_n)\dots$ and the distribution of actions $proc$ over the automata, we can build an accepted timed trace for an NTA. Namely, $E = \{e_0, \dots, e_n, \dots\}$, λ and δ are such that, for each $i \geq 0$, $\lambda(e_i) = a_i$ and $\delta(e_i) = d_i$, and $\preccurlyeq$ is the transitive closure of the relation $\preccurlyeq'$ defined as: for any events e_i and e_j, $e_i \preccurlyeq' e_j \iff \big(i \leq j \wedge proc(\lambda(e_j)) \cap proc(\lambda(e_j)) \neq \emptyset\big)$.

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 into S-subnets. Then we show how we can find this decomposition. We borrow the main definitions from [13], where the authors give a method (introduced in [17]) to decompose a live and bounded free-choice net into such components and we adapt this method to decompose more general nets.

4.1 Decomposition into 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 $\forall t \in T$, $|{}^\bullet t| = |t^\bullet| = 1$. Thus, an S-net can be seen as an automaton (places are locations and transitions are edges). 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 a net $N = (P,T,F)$ 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*.

A 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 $\bigcup_{i\in[1..n]} P_i = P$. In this case, the set of S-subnets is called a *cover* of N (and $\bigcup_{i\in[1..n]} T_i = T$ because the S-subnets are P-closed). We are looking for *minimal* S-subnets w.r.t. the set inclusion of their generating places, and we notice that connected S-subnets are always minimal. We are also looking for *minimal covers*, i.e. covers such that if one S-subnet is removed, then the net is no longer covered.

Note that the notion of S-subnet generalizes the notion of S-component presented in [13] 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 Fig. 5(b). The entry $\mathbf{N}(p,t)$ corresponds to the change of the marking of 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 with t.

Definition 10 (S-invariant [22]) 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 $I : P \to \mathbb{N}$ 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 ${}^\bullet\langle I\rangle = \langle I\rangle^\bullet$.

In the sequel, we consider S-invariants I such that $I : P \to \{0,1\}$ (set of places). Notice that the set of places of a minimal S-subnet is a minimal S-invariant, and conversely.

Proposition 1 *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*

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

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, $|P_i \cap {}^\bullet t| = |P_i \cap t^\bullet| = 1$ if $t \in T_i$ and 0 otherwise. Therefore, for each transition t, $\sum_{p \in {}^\bullet t} X_i(p) = \sum_{p \in t^\bullet} X_i(p)$, which characterizes an S-invariant. Moreover, this sum equals 0 or 1. Lastly, since each place is in at least one subset of places, for each place p, $\sum_{i \in [1..n]} X_i(p) \geq 1$.

($\Leftarrow$) Assume now that there exists a set of S-invariants $\{X_1, \ldots, X_n\}$ which satisfies the three conditions of Prop. 1. 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, $|{}^\bullet t \cap P_i| = |{}^\bullet t \cap \langle X_i \rangle| = \sum_{p \in {}^\bullet t} X_i(p) = 1$ or 0, from (2). If $t \in T_i = \langle X_i \rangle^\bullet$, then ${}^\bullet t \cap \langle X_i \rangle \neq \emptyset$ and we must have $|{}^\bullet t \cap \langle X_i \rangle| = 1$. In the same way, if $t \in T_i$, $|t^\bullet \cap P_i| = 1$. Hence N_i is an S-net. Lastly, the n S-subnets cover the net because for each place p, $\sum_{i \in [1..n]} X_i(p) \geq 1$, which implies that there exists i in $[1..n]$ such that $p \in \langle X_i \rangle$, that is $\bigcup_{i \in [1..n]} \langle X_i \rangle = P$. $\square$

When the net is decomposable, there exists a set $\{I_1, \ldots I_k\}$ of minimal S-invariants that is a minimal cover of the net. Such a 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, as shown by the examples in Paragraph. 4.1 below.

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 PNs, but we explain how the procedure can be extended to k-bounded PNs in Subsection 6.1. Lastly, notice that the conservation of the number of tokens in each S-subnet implies that unbounded PNs are not decomposable.

Decomposition algorithm. Some algorithms for the computation of minimal S-invariants can be found in [12] where they are called p-semiflows. Therefore, it is possible to compute the set $\mathbf{X}$ of minimal S-invariants with values in $\{0,1\}$ from a given incidence matrix $\mathbf{N}$. Hence, Algorithm 1 below describes how a net can be decomposed.

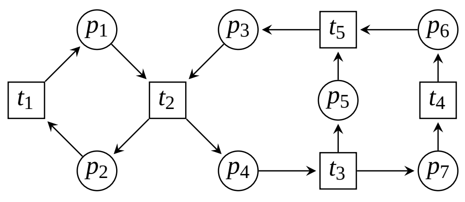
Decomposition examples. Below are some examples of decomposition. In Example 1, the net is decomposable, the decomposition is unique and some places belong to several components. In Example 2, the net is decomposable, the decomposition is not unique, and places belong to only one component. Lastly, in Example 3, the net is not decomposable.

Example 1 We want to decompose the net shown in Fig. 5(a). To this purpose, we determine its minimal S-invariants with values in $\{0,1\}$.

With the incidence matrix given in Fig. 5(b), we obtain the following non-zero minimal S-invariants: $X_1 = [1\ 1\ 0\ 0\ 0\ 0\ 0]$, $X_2 = [0\ 0\ 1\ 1\ 0\ 1\ 1]$, and $X_3 = [0\ 0\ 1\ 1\ 1\ 0\ 0]$. These S-invariants cover the net, therefore the net is decomposable. They also form a minimal cover (if one S-invariant is removed, the net is no longer covered), therefore they give a decomposition of the net. Hence the net is decomposable in the three S-subnets generated by the sets of places $\{p_1, p_2\}$ (X_1), $\{p_3, p_4, p_6, p_7\}$ (X_2), and $\{p_3, p_4, p_5\}$ (X_3), see Fig. 5(c).

```
Data: incidence matrix N
Result: minimal set S of minimal S-subnets that covers the net if the net is decomposable,
        empty set otherwise
begin
    S ← ∅;
    X ← set of minimal S-invariants with values in {0,1}, computed from N;
    if X does not cover the net then
        return S;
    end
    while X is not a minimal cover do
        foreach X in X do
            if X \ {X} covers the net then
                X ← X \ {X};
                break;
            end
        end
    end
    foreach X in X do
        S ← subnet generated by X;
        S ← S ∪ {S};
    end
    return S;
end
```

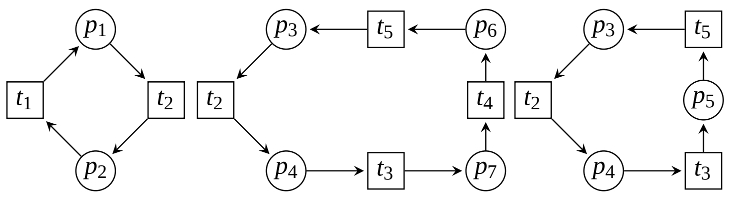
Algorithm 1: Decomposition algorithm



(a) A decomposable net

$$\begin{array}{c|ccccc}
 & t_1 & t_2 & t_3 & t_4 & t_5 \\
\hline
p_1 & 1 & -1 & 0 & 0 & 0 \\
p_2 & -1 & 1 & 0 & 0 & 0 \\
p_3 & 0 & -1 & 0 & 0 & 1 \\
p_4 & 0 & 1 & -1 & 0 & 0 \\
p_5 & 0 & 0 & 1 & 0 & -1 \\
p_6 & 0 & 0 & 0 & 1 & -1 \\
p_7 & 0 & 0 & 1 & -1 & 0
\end{array}$$

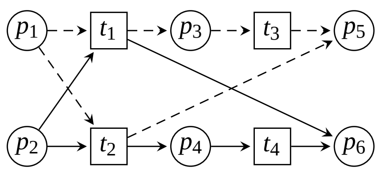
(b) Its incidence matrix



(c) Its decomposition

Fig. 5 A net which is decomposable in S-subnets, its incidence matrix, and its decomposition

Example 2 We want to decompose the net shown in Fig. 6(a). With the incidence matrix given in Fig. 6(b), we obtain the following non-zero minimal S-invariants: $X_1 = [1\ 0\ 1\ 0\ 1\ 0]$, $X_2 = [1\ 0\ 0\ 1\ 0\ 1]$, $X_3 = [0\ 1\ 1\ 0\ 1\ 0]$ and $X_4 = [0\ 1\ 0\ 1\ 0\ 1]$. The net is covered, therefore decomposable, and there are two minimal covers $\{X_1, X_4\}$ and $\{X_2, X_3\}$, therefore two decompositions. The two components of the decomposition given by $\{X_1, X_4\}$ are denoted in Fig. 6(a) by different line types: the arcs of the S-subnet generated by $\{p_1, p_3, p_5\}$ (X_1) are represented by dashed lines, and those of the one generated by $\{p_2, p_4, p_6\}$ (X_4) are represented by plain lines. In the second possible decomposition, p_1 and p_2 are switched.



(a) A decomposable net (the different line types denote the arcs of the two different components of one decomposition)

$$\begin{array}{c|cccc} & t_1 & t_2 & t_3 & t_4 \\ \hline p_1 & -1 & -1 & 0 & 0 \\ p_2 & -1 & -1 & 0 & 0 \\ p_3 & 1 & 0 & -1 & 0 \\ p_4 & 0 & 1 & 0 & -1 \\ p_5 & 0 & 1 & 1 & 0 \\ p_6 & 1 & 0 & 0 & 1 \end{array}$$

(b) Its incidence matrix

Fig. 6 A net which is decomposable in S-subnets and its incidence matrix

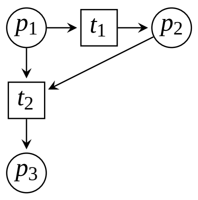


Fig. 7 A non decomposable net

Example 3 Consider the net of Fig. 7. Any S-subnet N' containing p_2 must also contain its input and output transitions t_1 and t_2. Then it must contain an input place for t_1 and an output place for t_2, which are necessarily p_1 and p_3. This means that the only candidate for being a S-subnet containing p_2 is the entire net, but it is not an S-net since t_2 has two input places. This can also be seen by computing the S-invariants from the incidence matrix: there is no non-zero solution with values in $\{0,1\}$ (but there are some with values in $\mathbb{N}$, for example $[1\ 1\ 2]$). Therefore, this net is not decomposable.

4.2 Size of the decomposition.

Assume net $N = (P,T,F)$ is decomposable in n S-subnets $N_1,\ldots,N_n$, such that $N_i = (P_i,T_i,F_i)$ is the subnet generated by P_i. The number of places in the decomposition is equal to $\sum_{i\in[1..n]} |P_i|$ 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 $\sum_{i\in[1..n]} |T_i|$ 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. Therefore, this translation deals with TPNs whose untimed support is *decomposable*. Moreover, in this section, we consider only TPNs whose untimed support is 1-bounded, in order to simplify the explanation, but the procedure can easily be extended to TPNs whose untimed support is k-bounded and still decomposable, as explained in Subsection 6.1. In

Subsection 6.2, we will discuss an extension to deal with bounded TPNs whose untimed support is unbounded and therefore not decomposable.

5.1 Procedure

Our procedure translates a time Petri net $\mathcal{N}$ into a network of timed automata and relies on a decomposition of the untimed support of $\mathcal{N}$ into S-subnets (that may be obtained using Algorithm 1). Therefore, our procedure is not (at least directly) applicable if the net is not decomposable. We also require that each S-subnet is initially marked with one token (we discuss the case when S-subnets are not marked, or marked with more than one token in Subsection 6.1). For our example of Fig. 2, we get the subnets shown in Fig. 8(a).

Each S-subnet determines a process in the time Petri net and will be translated into a timed automaton. We focus now on the treatment of time constraints in order to get a network of timed automata which has the same distributed timed language as $\mathcal{N}$.

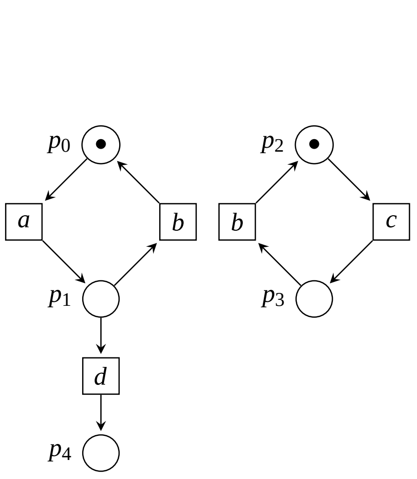
This involves three steps:

1. Each S-subnet is translated into 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 single clock x_i. This clock is reset on each edge. The idea is that the value of x_i gives the time elapsed 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. 8(b), we can stay in (ℓ_1, ℓ_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).

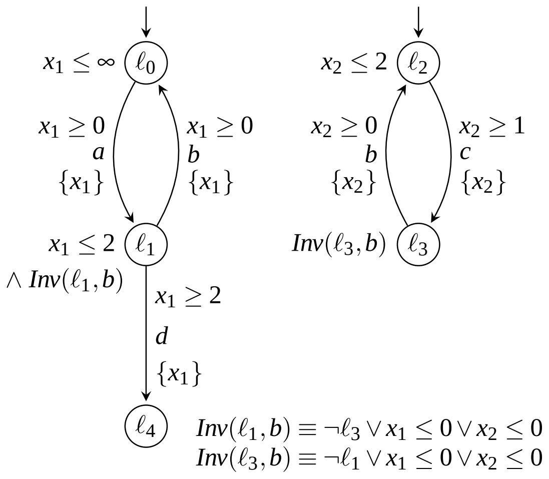
Formally, a TPN $\mathcal{N} = (P, T, F, M_0, efd, lfd)$ with n processes can be translated in the NTA $(A_1, \ldots, A_n)$ with, for all i in $[1..n]$, $A_i = (L_i, \ell_i^0, C, \Sigma_i, E_i, Inv_i)$ where

- $L_i = P_i$ (places of the i^{th} subnet),
- ℓ_i^0 is s.t. $\{\ell_i^0\} = P_i \cap M_0$,
- $C = \{x_1, \ldots, x_n\}$,
- $\Sigma_i = T_i$ (transitions of the i^{th} subnet),
- E_i is the set of edges (p, g, t, r, p') s.t. $t \in T_i$, $\{p\} = {}^\bullet t \cap P_i$, $\{p'\} = t^\bullet \cap P_i$, $g \equiv x_i \geq efd(t)$, and $r = \{x_i\}$,
- $Inv_i : P_i \to \mathscr{B}(C, P)$ 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.

That is, $Inv_i(p)$ ensures that we cannot overtake the latest firing delay of an enabled transition which is in the post-set of p. Notice that $Inv_i(p)$ uses the extended syntax (see



(a) Decomposition in processes



$x_1 \leq \infty$
ℓ_0
$x_2 \leq 2$
ℓ_2
$x_1 \geq 0$ a $\{x_1\}$
$x_1 \geq 0$ b $\{x_1\}$
$x_2 \geq 0$ b $\{x_2\}$
$x_2 \geq 1$ c $\{x_2\}$
$x_1 \leq 2$ $\wedge Inv(\ell_1, b)$
ℓ_1
$Inv(\ell_3, b)$
ℓ_3
$x_1 \geq 2$ d $\{x_1\}$
ℓ_4
$Inv(\ell_1, b) \equiv \neg\ell_3 \vee x_1 \leq 0 \vee x_2 \leq 0$
$Inv(\ell_3, b) \equiv \neg\ell_1 \vee x_1 \leq 0 \vee x_2 \leq 0$

(b) Resulting NTA

Fig. 8 Translation of the TPN of Fig. 2

Subsection 3.1): automaton 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.

In the rest of this section, we first prove that this translation is correct w.r.t. the preservation of the distributed timed language and we discuss the size of the resulting NTA, then we show that the use of the extended syntax is necessary and we identify some cases when the local syntax is sufficient.

Proposition 2 *The initial 1-bounded time Petri net $\mathscr{N}$ and the network of timed automata $\mathscr{S}$ which results from the translation have the same distributed timed language (are timed bisimilar with the same distributions of actions).*

Proof A marking of $\mathscr{N}$ can be identified with a vector of current locations of $\mathscr{S}$. A place may correspond to several locations in the NTA, but in this case, if it is active in one automaton, then it is active in all the automata where it appears. Indeed, for any transition t, any place in $t^\bullet$ is in a component (because the net is covered) and t is also in this component (because the components are P-closed). Therefore, the firing of t in $\mathscr{N}$ corresponds to a synchronization on t in $\mathscr{S}$.

For any i in $[1..n]$, we note $p_i = M \cap P_i$ the current location of automaton A_i. We first show the following equivalence:

$$v \models \bigwedge_{1 \leq i \leq n} Inv_i(p_i) \iff \left(\forall t \in T, {}^\bullet t \subseteq M \implies v(t) \leq lfd(t)\right) \tag{1}$$

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

Moreover the guard $g_i(t)$ associated with the edge labeled by t in automaton A_i, is built so that $g_i(t) \equiv x_i \geq efd(t)$, and again, when t is enabled, $v(t) = \min_{i \in I_t}\big(v(x_i)\big)$, which gives:

$$\forall t \in T, {}^\bullet t \subseteq M \implies \left(v \models \bigwedge_{i \in I_t} g_i(t) \iff v(t) \geq efd(t)\right) \tag{2}$$

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

$$(M,v)\ \mathscr{R}\ (M,\nu) \iff \left(\forall t \in T, {}^\bullet t \subseteq M \implies \nu(t) = \min_{i\in I_t}\big(v(x_i)\big)\right)$$

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

We first observe that $(M_0,v_0)\ \mathscr{R}\ (M_0,\nu_0)$ and we show that, from any correspondent states, $(M,v)\ \mathscr{R}\ (M,\nu)$, 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 $(M,v+d)\ \mathscr{R}\ (M,\nu+d)$.

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 $\mathscr{S}$ and $(M,v+d)\ \mathscr{R}\ (M,\nu+d)$.

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 $e = (e_1,\dots,e_n) \in (E_1 \cup \{\bullet\}) \times \cdots \times (E_n \cup \{\bullet\})$ 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 $v' \models \bigwedge_i Inv_i(p'_i)$.

$(M,v)\ \mathscr{R}\ (M,\nu)$ 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. $M'' = (M \setminus {}^\bullet t) \cup t^\bullet = M'$, 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 the firing of 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 $(M',v')\ \mathscr{R}\ (M',\nu')$.

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 $(M',v')\ \mathscr{R}\ (M',\nu')$.

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

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 last paragraph of Sect. 4.2). 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.

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

For example, Fig. 9 shows two timed traces W and W' representing the beginning of two possible runs, without synchronization, for the TPN $\mathscr{N}$ of Fig. 2. Any NTA $\mathscr{S}$ using the local syntax and accepting W and W' would also accept the timed trace built by composing the projection of W onto π_1 and the projection of W' onto π_2 (see Fig. 9). But this timed trace is not accepted by $\mathscr{N}$.

To prove Prop. 3, 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 W can be defined as a tuple $(w, proc)$ where w is a timed linearization of W, see Fig. 4 and its caption that gives one timed linearization of the timed trace represented in the figure.

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

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

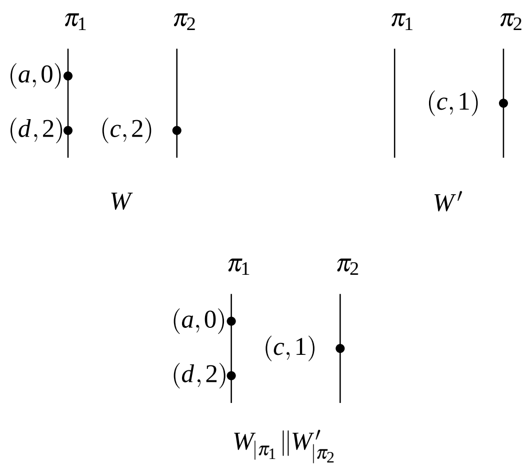


Fig. 9 Two accepted timed traces and one non accepted timed trace for the TPN of Fig. 2

– 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, W such that for each i in $[1..n]$, if Σ_i denotes the set of actions that appear in w_i, then $W_{|\Sigma_i} = w_i$.

We denote by $\mathscr{S}$ a network of n timed automata $(A_1,\ldots,A_n)$, and by $R_\theta(\mathscr{S})$ the set of all timed traces representing admissible runs of $\mathscr{S}$, without synchronization, and stopping at date θ.

Lemma 1 *Let $\mathscr{S}$ be a network of n timed automata that do not read the state of the other automata, then, for any timed traces $W_1,\ldots,W_n \in R_\theta(\mathscr{S})$ (not necessarily different), $W_{1|\pi_1} \parallel \cdots \parallel W_{n|\pi_n} \in R_\theta(\mathscr{S})$.*

Proof (Lemma 1) 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 (Prop. 3) Assume that the two automata corresponding to the two processes of the TPN $\mathscr{N}$ of Fig. 2 are not able to read the current location and the clock of the other automaton. Then, for any two timed traces W and W', representing two admissible runs without synchronization, stopping at date θ, the timed trace $W_{|\pi_1} \parallel W'_{|\pi_2}$ represents also an admissible run.

If we choose, as in Fig. 9, $W = (w,proc)$ and $W' = (w',proc)$, 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 $W_{|\pi_1} \parallel W'_{|\pi_2} = \big((a,0)(c,1)(d,2),proc\big)$ (see Fig. 9) should represent an admissible run for $\mathscr{S}$ and $\mathscr{N}$. Which is false because as soon as c has been performed, b must be performed immediately. Therefore, the local syntax (see Subsection 3.1) must be extended. □

5.4 TPNs with good decompositional properties

Prop. 3 states that in general any NTA $\mathscr{S}$ having the same distributed timed language as a given TPN $\mathscr{N}$, uses the extended syntax defined in Subsection 3.1, i.e. the automata of $\mathscr{S}$

have to read information about the state of the others. This creates a dependency between the automata, which is not as strong as in the case of a synchronization on a common action, since it is asymmetric: only one automaton reads. Still, we are interested in identifying the cases where the automata do not need to read information about the state of their neighbours, which we regard as a good decompositional property.

We did not find an algorithm that decides in general if TPN $\mathcal{N}$ has this property and we do not know if it is decidable. However, we present a simple sufficient condition, which can be detected by reachability analysis on the marking TA of $\mathcal{N}$. We show how our construction can be easily adapted in this case, to avoid reading information about other automata.

A class of TPN with good decompositional properties.

Proposition 4 *Let $\mathcal{N}$ be a 1-bounded TPN which is decomposable, and such that for any transition t, there exists a place p in ${}^\bullet t$ which is always the last place to be marked among ${}^\bullet t$ when t becomes enabled, then there exists an NTA $\mathcal{S}$ with the local syntax and with the same distributed timed language as $\mathcal{N}$.*

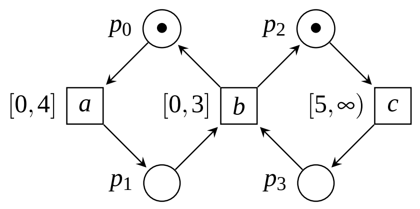
Proof We use the same translation as before and choose to add $Inv(t)$ only in $Inv_i(p)$ (this can be done, as explained in the third step of the translation). By construction, $Inv(t) \equiv \big((\bigwedge_{p' \in {}^\bullet t} p') \Rightarrow \min\limits_{k \in I_t}(x_k) \leq lfd(t)\big)$. In this 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 $\min\limits_{k \in I_t}(v(x_k)) = v(x_i) = v(t)$. 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. □

This property can be expressed in CTL and checked on the marking TA: for any transition t and for any place $p \in {}^\bullet t$ we check whether the formula $AG(p \in M \Leftrightarrow {}^\bullet t \subseteq M)$ is satisfied (the formula has to hold for at least one place of ${}^\bullet t$).

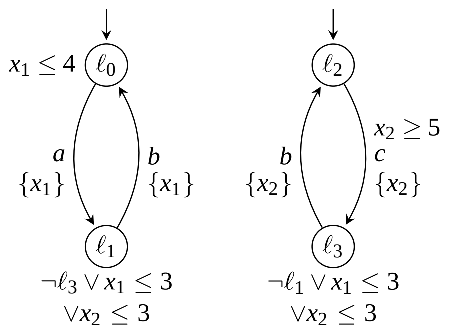
For example, consider the TPN of Fig. 10(a). Without studying the timing constraints, the translation gives the NTA of Fig. 10(b), where the invariants of locations ℓ_1 and ℓ_3 read the state of the other automaton. But when we look at the timing constraints, we can see that location ℓ_1 is always activated before location ℓ_3, i.e. $\ell_3 \Rightarrow \ell_1$, that is b is enabled as soon as ℓ_3 is marked. Therefore, the invariant associated with b can be placed on ℓ_3 only and simplified. Indeed, there is no need to read ℓ_1 since we know it is marked and no need to read x_1 since $\min(x_1, x_2) = x_2$. Eventually, we get the NTA of Fig. 10(c).

Some more complicated examples. We believe that the class of TPN with good decompositional properties that we described above captures most of the practical cases in which one can avoid reading information about other components. The idea is that most often, when there is a variable delay before several components synchronize on a common action, this delay is due to one of the components (which may typically be waiting for some input), while the other components are simply waiting; then the invariant that triggers the synchronization can be associated to the component that is responsible for the delay, and it will not need to read any information about the state of the others: it can assume that the others are ready to synchronize.

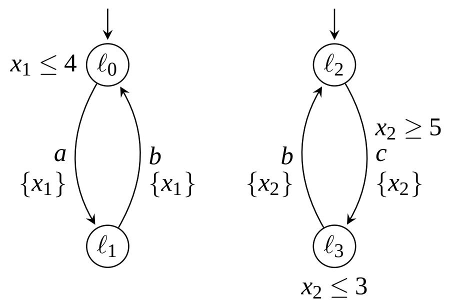
On the other hand, if the delay is really due to several components, then it is very likely that none of the components have enough information locally to be able to trigger the synchronization without reading information about the state of the others. This observation is not always verified: we now show an example for that, but we try to convince the reader that this kind of examples is not very likely to occur in practice.



(a) Initial TPN



(b) Result of the procedure of Subsection 5.1



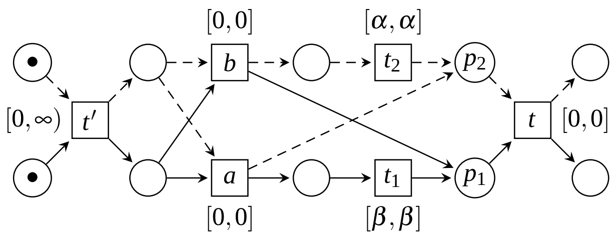
(c) Result of the translation when observing that b is always enabled as soon as p_3 is marked

Fig. 10 A TPN that can be translated in an NTA with the local syntax

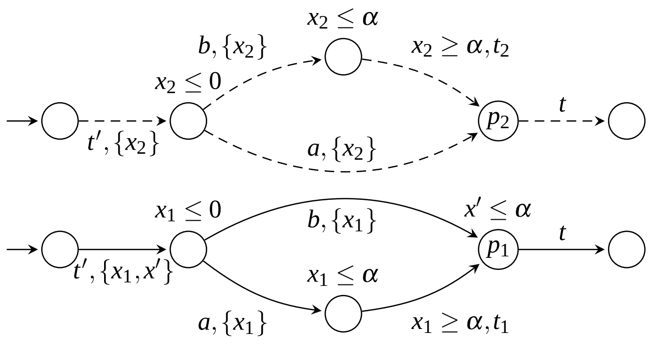
Consider the example depicted in Fig. 11(a), where α and β are parameters for the values of the constants. This TPN can be decomposed into two components (see the example of Fig. 6(a), which is very similar). These two components will be translated into two automata A_1 (plain lines in the figure), with clock x_1, and A_2 (dashed lines), with clock x_2. Here, after the occurrence of t', either a occurs or b occurs.

For the first example, we assume that $\alpha = \beta$. Then, whatever transition occurs between a and b, t will be enabled α time units after the firing of t'. Therefore a clock x' can be added in one of the automata, reset when t' fires, and used in the invariant of one of the input locations of t as the condition $x' \leq \alpha$ (see Fig. 11(b)).

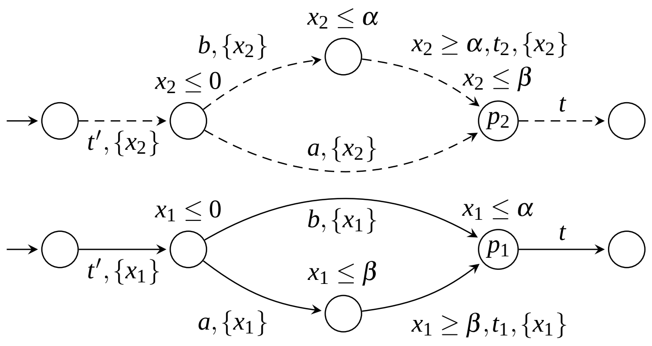
Now, let us assume that $\alpha \neq \beta$. If a occurs, then p_2 is marked immediately, and p_1 is marked β time units later. In this case, p_1 must be disabled immediately and p_2 must be disabled after β time units. If b occurs, then p_1 is marked immediately, and p_2 is marked α time units later. In this case, p_2 must be disabled immediately and p_1 must be disabled after α time units. Therefore, in order to respect the latest firing delay of t, when t is enabled, it suffices to attach $x_2 \leq \beta$ to p_2 and $x_1 \leq \alpha$ to p_1 (see Fig. 11(c)).



(a) Initial TPN



(b) $\alpha = \beta$. NTA with the local syntax but one more clock



(c) $\alpha \neq \beta$. NTA with the local syntax

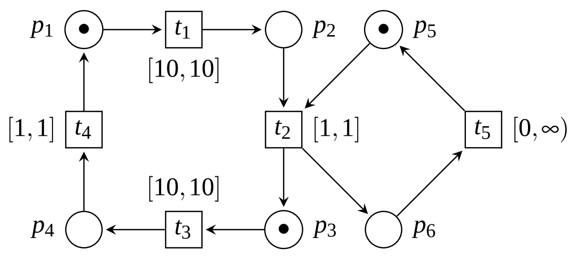
Fig. 11 A TPN that can be translated in an NTA with a local syntax. The arcs of the two components are drawn differently

6 Discussion and examples

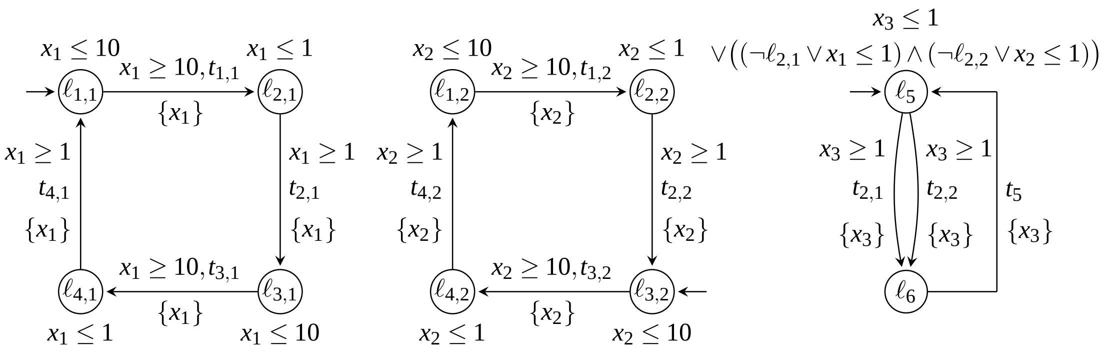
6.1 Dealing with decomposable TPNs whose untimed support is k-bounded

The translation procedure was given for TPNs whose untimed support is a decomposable PN such that each S-subnet is initially marked with one token, but we mentioned the possibility to translate also TPNs whose untimed support is decomposable and such that the S-subnets may not be marked or be marked with more than one token. Below, we describe the procedure on an example.

Consider a net such that an S-subnet is initially marked with more than one token. The untimed support of the TPN of Fig. 12(a) is decomposable into the two S-subnets generated by $\{p_1, p_2, p_3, p_4\}$ and $\{p_5, p_6\}$. Since one S-subnet is initially marked with two tokens, it corresponds to two processes π_1 and π_2 with the same structure. Moreover, since a transition



(a) Initial TPN with two S-subnets but three processes



(b) Resulting NTA where two automata have the same structure but different initial locations

Fig. 12 A TPN whose support is a decomposable PN such that one S-subnet is initially marked with 2 tokens and its translation into an NTA

needs only one token in each one of its input places to be enabled, π_1 and π_2 need not know the state of each other. That is, each one of them will model the course of one token in the net.

In Fig. 12(b), we labeled differently the actions in the first two automata, to denote that they do not synchronize with each other. And since the third process synchronizes on t_2, the edge labeled by t_2 in the associated automaton is duplicated to denote the two possible synchronizations with $t_{2,1}$ and $t_{2,2}$.

Notice that this approach also applies to Petri nets such that an S-subnet N_i is not marked initially (and hence will never be marked). There is no process corresponding to N_i and there will be no corresponding TA. Moreover, for any other S-subnet N_j that shares a transition with N_i, this transition will never fires, and this is ensured by the fact that edges are duplicated in as many versions as possible synchronizations, and since there is no possible synchronization, there will be no edge denoting this transition in the TA associated with N_j.

6.2 Dealing with bounded TPNs whose untimed support is unbounded

The conservation of the weighted sum of the tokens in an S-invariant (see [13]) shows that unbounded PN are not decomposable. Moreover, not all 1-bounded PNs are decomposable, although we think that, most models of real systems are.

However, our method can be adapted to some temporally 1-bounded TPNs whose untimed support is unbounded. The idea is to modify the underlying unbounded net so that it becomes decomposable and to adapt the timing information in the NTA to preserve the semantics of the original TPN $\mathcal{N}$. We use complementary places: for a place p, the complementary place $\overline{p}$, is built such that ${}^\bullet\overline{p} = p^\bullet \setminus {}^\bullet p$, $\overline{p}^\bullet = {}^\bullet p \setminus p^\bullet$, and $\overline{p}$ is marked iff p is not. For a place p, let the predicate $NC(p)$ denote that p is not covered by any S-component, i.e.

$$NC(p) \iff \left(\forall X : P \to \{0,1\}, X \cdot \mathbf{N} = \mathbf{0} \implies X(p) = 0\right).$$

Then, we can transform the untimed unbounded PN $\mathcal{N}_{untimed} = (P,T,F,M_0)$ into a bounded PN $\mathcal{N}'_{untimed} = (P',T,F',M'_0)$ where

- $P' = P \cup \{\overline{p} \mid NC(p)\}$, i.e. for each place p that is not covered by any S-component, a complementary place $\overline{p}$ is added,
- $F' = F \cup \{(\overline{p},t) \mid NC(p) \wedge (t,p) \in F\} \cup \{(t,\overline{p}) \mid NC(p) \wedge (p,t) \in F\}$,
- $M'_0 = M_0 \cup \{\overline{p} \mid NC(p) \wedge p \notin M_0\}$.

For example, consider the 1-bounded TPN $\mathcal{N}$ of Fig. 13(a) without the dashed items (taken from [23]). Its untimed support is unbounded, but the timing constraints prevent there being more that one token in p_s. Even though the net is not decomposable without modification, in the structure of the net, we can identify three parts: the S-subnets generated by $\{p_1, p_2\}$, and $\{p_3, p_4\}$ and the subnet generated by $\{p_s\}$ which is not a valid component, because it is not an S-net. Therefore, we add a complementary place to p_s to make the untimed PN 1-bounded, by restricting the number of tokens in place p_s to 1. With this new place, V has to wait for the occurrence of P before occurring again. That is, the boundedness that was ensured by the timing constraints in $\mathcal{N}$, is now ensured in the untimed PN $\mathcal{N}'_{untimed}$ by the complementary places. Notice also that the following proposition holds.

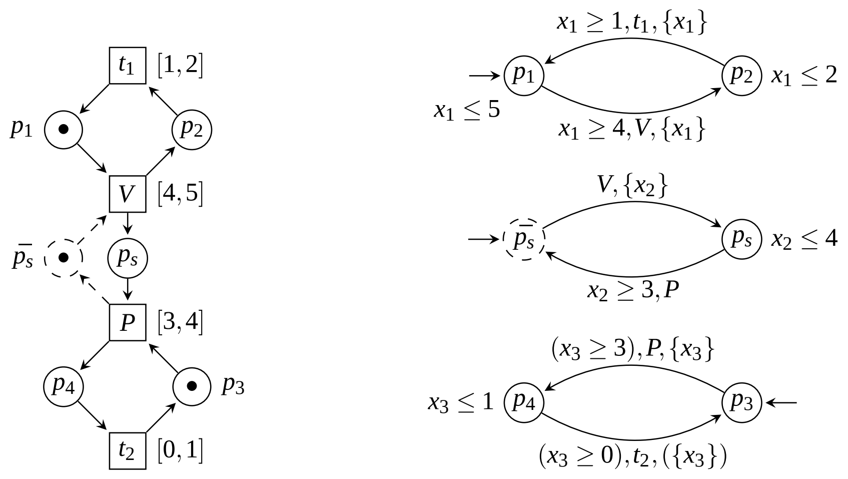
Proposition 5 *A timed run of $\mathcal{N}$ from which the occurrence dates are removed is a run of $\mathcal{N}'_{untimed}$.*

Proof We define a relation $\mathcal{R}$ which associates a (valid) state (M,ν) of $\mathcal{N}$ with a state M' of $\mathcal{N}'_{untimed}$, and show that $\mathcal{R}$ is a simulation. Namely,

$$(M,\nu)\,\mathcal{R}\,M' \iff M = M' \setminus \{\overline{p} \mid NC(p) \wedge p \notin M\}.$$

First, $(M_0,\nu_0)\,\mathcal{R}\,M'_0$ holds. Second, assume that t is firable from state (M,ν) which is $\mathcal{R}$-related to state M'. Then t is also enabled in $M' = M \cup \{\overline{p} \mid NC(p) \wedge p \notin M\}$. Indeed, in $\mathcal{N}'_{untimed}$, if there is a complementary place $\overline{p}$ in the input places of t, then in $\mathcal{N}$, $p \in t^\bullet \setminus {}^\bullet t$, and since the TPN $\mathcal{N}$ is 1-bounded, $p \notin M$ and $\overline{p} \in M'$. When t fires in $\mathcal{N}$, it leads to state (M_1,ν_1) such that $M_1 = (M \setminus {}^\bullet t) \cup t^\bullet$ (regardless of ν_1). And when t fires in $\mathcal{N}'_{untimed}$, it leads to marking M'_1 such that
$$\begin{aligned}M'_1 &= (M' \setminus {}^{\bullet\prime}t) \cup t^{\bullet\prime}\\ &= \left((M \cup \{\overline{p} \mid NC(p) \wedge p \notin M\}) \setminus ({}^\bullet t \cup \{\overline{p} \mid NC(p) \wedge p \in t^\bullet \setminus {}^\bullet t\})\right)\\ &\quad \cup \left(t^\bullet \cup \{\overline{p} \mid NC(p) \wedge p \in {}^\bullet t \setminus t^\bullet\}\right),\end{aligned}$$
because by definition of $\mathcal{N}'_{untimed}$, $\overline{p} \in {}^{\bullet\prime}t \Leftrightarrow p \in t^\bullet \setminus {}^\bullet t$ and $\overline{p} \in t^{\bullet\prime} \Leftrightarrow p \in {}^\bullet t \setminus t^\bullet$. Since $\{\overline{p} \mid NC(p)\}$ is disjoint from M, ${}^\bullet t$ and $t^\bullet$, this can be simplified in
$$M'_1 = \left((M \setminus {}^\bullet t) \cup t^\bullet\right) \cup \{\overline{p} \mid NC(p) \wedge p \in \left(\overline{M} \setminus (t^\bullet \setminus {}^\bullet t)\right) \cup ({}^\bullet t \setminus t^\bullet)\}$$
and lastly, since $\left(\overline{M} \setminus (t^\bullet \setminus {}^\bullet t)\right) \cup ({}^\bullet t \setminus t^\bullet) = (\overline{M} \cup {}^\bullet t) \setminus t^\bullet = \overline{(M \setminus {}^\bullet t) \cup t^\bullet} = \overline{M_1}$,
$M'_1 = M_1 \cup \{\overline{p} \mid NC(p) \wedge p \notin M_1\}$. Therefore $(M_1,\nu_1)\,\mathcal{R}\,M'_1$. □

But if the timing delays of $\mathcal{N}$ are added to $\mathcal{N}'_{untimed}$, both TPNs will not have the same timed semantics. For instance, on our example, the timed word $(V,4)(t_1,5)(P,7)(t_2,8)(V,9)$ is no longer accepted. However, the transformation is only used to find a decomposition of the net and now our translation can be adapted.



(a) A temporally 1-bounded TPN (without $\overline{p_s}$)

(b) Resulting NTA where information between parenthesis can be removed

Fig. 13 Translation of a structurally unbounded TPN

Proposition 6 *Let $\mathcal{N}$ be a 1-bounded TPN whose untimed support is unbounded. If the net $\mathcal{N}'_{untimed}$ defined above is decomposable, then there exists an NTA with the same distributed timed language as $\mathcal{N}$.*

Proof If $\mathcal{N}'_{untimed}$ is decomposable, we choose a decomposition such that each $\{p, \overline{p}\}$ forms a component. Then we adapt the translation: each component corresponds to an automaton and the timing information is added in the same way as in Subsection 5.1, but without considering the new places because the time spent in these places is not relevant for the semantics of the TPN. That is, for each new place $\overline{p}$, there is no clock reset in the ingoing edges of $\overline{p}$, no guard on the outgoing edges of $\overline{p}$, no invariant on $\overline{p}$, and $\overline{p}$ appears in no invariant. In this way, we get an NTA with the same distributed timed language as the initial TPN. □

In the context of our example, this results in the NTA of Fig. 13(b). We decide to attach $Inv(P)$ to p_s, and since we notice that, in $\mathcal{N}$, if p_s is marked, then p_3 is also marked (i.e., in the NTA $\min(x_2,x_3) = x_2$), we simplify this invariant: $Inv(P) \equiv (p_s \wedge p_3) \Rightarrow \min(x_2,x_3) \leq 4 \equiv p_s \Rightarrow x_2 \leq 4$, and therefore $Inv(p_s) \equiv Inv(P) \wedge p_s \equiv x_2 \leq 4$.

6.3 Reverse translation

Let us now consider the reverse translation, i.e. from an NTA to a TPN. There exist translations, for example in [5], 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 mapping between 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 [18]. We can even strengthen this by

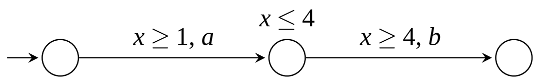


Fig. 14 A TA that cannot be translated in a time S-net with one token

proving that some TA with one clock cannot be translated in finite time S-net with one token (see Prop. 7). Therefore, only a very small class of TA can be translated.

Proposition 7 *Time S-nets with one token are strictly less expressive than TA with one clock.*

Proof Assume that the TA A of Fig. 14 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 $d_1 = 4 - \theta_1$ 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 A has *infinitely* many. Thus, A cannot be translated in a time S-net with one token. $\square$

If we impose for example that each TA has one clock which is reset on each edge, that the invariant are of the form $x \leq n$ and that the guards are of the form $x \geq m$, then the TAs can be translated into time S-nets, but even in this simple case, the composition of these components into a TPN with the same semantics as the initial NTA is not always possible.

6.4 Conclusion and outlook

Usability in practice. We have translated some example time Petri nets with the translation proposed in [10] and with our translation, and we have used UPPAAL (see [21]) to check a reachability property on the resulting networks of timed automata.

Although our translation only works for TPNs whose untimed support is bounded, 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, since the components are clearly identified, and closer to the original model.

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 [10], 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 [10], the number of active clocks is equal to the number of enabled transitions in the corresponding marking (Theorem 3 in [10]). Therefore, we can have fewer active clocks if there are some conflicts.

Lastly, we have shown an extension of the translation procedure to deal with some bounded TPNs whose support cannot be decomposed. Once we get the structure of the automata, the method that assigns the time constraints can be applied with only some minor modifications.

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.

Acknowledgements This work is partially supported by the FARMAN project EMoTiCon funded by ENS Cachan and the French ANR projects DOTS and ImpRo.

References

1. Akshay, S., Bollig, B., Gastin, P.: Automata and logics for timed message sequence charts. In: Foundations of Software Technology and Theoretical Computer Science (FSTTCS), *LNCS*, vol. 4855, pp. 290–302. Springer, New Delhi, India (2007)
2. Akshay, S., Bollig, B., Gastin, P., Mukund, M., Narayan Kumar, K.: Distributed timed automata with independently evolving clocks. In: International Conference on Concurrency Theory (CONCUR), *LNCS*, vol. 5201, pp. 82–97. Springer, Toronto, Canada (2008)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126**(2), 183–235 (1994)
4. Balaguer, S., Chatain, Th., Haar, S.: A concurrency-preserving translation from time Petri nets to networks of timed automata. In: International Symposium on Temporal Representation and Reasoning (TIME), pp. 77–84. IEEE Computer Society Press, Paris, France (2010)
5. Bérard, B., Cassez, F., Haddad, S., Lime, D., Roux, O.H.: When are timed automata weakly timed bisimilar to time Petri nets? Theoretical Computer Science **403**(2-3), 202–220 (2008)
6. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time Petri nets. IEEE Transactions on Software Engineering **17**(3), 259–273 (1991)
7. Berthomieu, B., Ribet, P.O., Vernadat, F.: The tool TINA – construction of abstract state spaces for Petri nets and time Petri nets. International Journal of Production Research **42**(14), 2741–2756 (2004)
8. Bozga, M., Daws, C., Maler, O., Olivero, A., Tripakis, S., Yovine, S.: Kronos: a model-checking tool for real-time systems. In: International Conference on Computer Aided Verification (CAV), *LNCS*, vol. 1427, pp. 546–550 (1998)
9. Byg, J., Joergensen, K., Srba, J.: An efficient translation of timed-arc Petri nets to networks of timed automata. In: International Conference on Formal Engineering Methods, *LNCS*, vol. 5885, pp. 698–716. Springer-Verlag (2009)
10. Cassez, F., Roux, O.H.: Structural translation from time Petri nets to timed automata. Journal of Systems and Software (2006)
11. Cerans, K., Godskesen, J.C., Larsen, K.G.: Timed modal specification - theory and tools. In: International Conference on Computer Aided Verification (CAV), *LNCS*, vol. 697, pp. 253–267. Springer (1993)
12. Colom, J.M., Silva, M.: 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, pp. 79–112. Springer-Verlag, London, UK (1991)
13. Desel, J., Esparza, J.: Free choice Petri nets. Cambridge University Press, New York, USA (1995)
14. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific Publishing Co., Inc., River Edge, NJ, USA (1995)
15. Gardey, G., Lime, D., Magnin, M., Roux, O.H.: Romeo: A tool for analyzing time Petri nets. In: International Conference on Computer Aided Verification (CAV), *LNCS*, vol. 3576, pp. 418–423. Springer (2005)
16. Gardey, G., Roux, O.H., Roux, O.F.: State space computation and analysis of time Petri nets. Theory and Practice of Logic Programming **6**(3), 301–320 (2006)
17. Hack, M.: Analysis of production schemata by Petri nets. Master's thesis, Massachusetts Institute of Technology, Cambridge, USA (1972)

18. Henzinger, T.A., Kopke, P.W., Wong-Toi, H.: The expressive power of clocks. In: International Colloquium on Automata, Languages and Programming (ICALP), pp. 417–428 (1995)
19. Jensen, K., Kristensen, L.M., Wells, L.: Coloured petri nets and cpn tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer (STTT) **9**(3-4), 213–254 (2007)
20. Lanotte, R., Maggiolo-Schettini, A., Peron, A.: Timed cooperating automata. Fundamenta Informaticae **43**, 153–173 (2000)
21. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer (STTT) **1**(1-2), 134–152 (1997)
22. Lautenbach, K.: Liveness in Petri nets. Tech. rep., Gesellschaft fr Mathematik und Datenverarbeitung, Bonn, Germany (1975)
23. Lime, D., Roux, O.H.: Model checking of time Petri nets using the state class timed automaton. Journal of Discrete Event Dynamic Systems (jDEDS) **16**(2), 179–205 (2006)
24. Lugiez, D., Niebert, P., Zennou, S.: A partial order semantics approach to the clock explosion problem of timed automata. Theoretical Computer Science **345**(1), 27–59 (2005)
25. Merlin, P.M.: A study of the recoverability of computing systems. Ph.D. thesis, University of California, Irvine (1974)
26. Niebert, P., Qu, H.: Adding invariants to event zone automata. In: International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS), *LNCS*, vol. 4202, pp. 290–305. Springer (2006)
27. Sifakis, J., Yovine, S.: Compositional specification of timed systems (extended abstract). In: Symposium on Theoretical Aspects of Computer Science (STACS), pp. 347–359. Springer-Verlag, London, UK (1996)