Concurrent behaviors with architectural constraints or imperfect clocks (Deliverable 4.2)

S. Akshay[1,2], Loïc Hélouët[2], Claude Jard[1,2,5], Pierre-Alain Reynier[3], Didier Lime[4], Olivier.H.Roux[4]

[1] INRIA/IRISA Rennes, France
[2] ENS Cachan Bretagne, Rennes, France
[3] Laboratoire d'Informatique Fondamentale de Marseille, France
[4] IRCCYN, Nantes
[5] Université de Nantes, AtlanSTIC & LINA

Abstract. Time in specification models is often considered as *perfect*, that is a considered model is supposed implemented on an architecture that can provide arbitrary precision in time measurement without drift, execute tasks instantaneously at precisely defined dates, etc. Of course, such assumptions are usually not met by real implementations : clocks drift, tasks are launched at imprecise dates, and starting them may require a small delay after a decision, etc. In addition to imprecision, some architectural constraints (distribution, available resources,...) may affect the expected behavior of an idealized abstract model at implementation time.
For these reasons, it is of paramount importance to verify that the behavior of a model can be preserved even under some imprecision of time measurement, or when the implementation architecture imposes additional constraints, such as time sharing mechanisms among processes, etc. Another verification of paramount importance is to check wheter there exists at all some clock precision allowing for preservation of important behaviors of a system.
This document considers Time Petri nets as model for distributed systems, and studies robustness issues for this specification formalism. We first consider how architectural constraints, specified as another Petri net influence the behavior of a system. We then consider how imprecision in time, modeled as guard enlargement, influences the behavior of the original specification.

Table of Contents

1 Introduction

Robustness is a key issue for the implementation of systems. Very often, models of distributed systems are idealized, and in particular, time aspects are very often too perfect to be realistic. Indeed, time in specification models is often considered as *perfect*, and model often suppose that the implementation architecture can provide arbitrary precision in time measurement without drift, execute tasks instantaneously at precisely defined dates, etc. Of course, such assumptions are usually not met by real implementations : clocks drift, tasks are launched at imprecise dates, and starting them may require a small delay after a decision, etc. In addition to imprecision, some architectural constraints (distribution, available resources,...) may affect the expected behavior of an idealized abstract model at implementation time.

This is particularly harmful for critical systems, where models are used to check some safety properties. Systems which are not affected by imprecise time issues are called *robust*. Once a system is implemented on a given architecture, one may discover that it does not behave as expected: some specified behaviors are never met or unspecified behaviors appear. Hence, if a system is proved safe but is not robust, some safety properties proved on the idealized model may not be met by an impelmentation of the model.

Starting from a description of a system, one wants to ensure that the considered system can run as expected on a given architecture with resource constraints (e.g., processors, memory), scheduling schemes on machines implementing several components of the system, imprecision in clocks, possible failures and so on. When the impementation architecture is not precisely known, another interesting issue is to detect wheter there exists a way to make time measurement precise enough so that the overall behavior of the model (or its safety) is not affected.

This document considers Time Petri nets as model for distributed systems, and studies robustness issues for this specification formalism. We first define notations in section 2. We then consider how architectural constraints, specified as another Petri net influence the behavior of a system in Section 3. Last, we consider how imprecision in time, modeled as guard enlargement, influences the behavior of the original specification.

Note that this report describes published results [2, 1] and that for the sake of conciseness, we only mention the major results of these publications, without proofs. Complete proofs can be found in the corresponding publications.

2 Definitions

This xsection defines the notations and models that will be used throughout the report.

2.1 Notations

Let $\mathbb{Q}^+, \mathbb{R}^+$ denote the set of non-negative rationals and reals respectively. Then, $\mathcal{I}$ denotes the set of time intervals, i.e., intervals in $\mathbb{R}^+$ with end points in $\mathbb{Q}^+ \cup \{+\infty\}$. An interval $I \in \mathcal{I}$ can be open (I^-, I^+), closed $[I^-, I^+]$, semi-open $(I^-, I^+], [I^-, I^+)$ or unbounded $[I^-, +\infty), (I^-, +\infty)$, where I^- and $I^+ \in \mathbb{Q}^+$.

Let Σ be a finite alphabet, Σ^* is the set of finite words over Σ. We also use $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$ with ε (the empty word) not in Σ. The sets $\mathbb{N}$, ${\mathbb{Q}^+}_{\geq 0}$ and ${\mathbb{R}^+}_{\geq 0}$ are respectively the sets of natural, non-negative rational and non-negative real numbers.

A *timed word* over Σ is a finite sequence $w = (a_0, t_0)\ (a_1, t_1) \ldots (a_n, t_n)$ s.t. for every $0 \leq i \leq n$, $a_i \in \Sigma$, $t_i \in {\mathbb{R}^+}_{\geq 0}$ and $t_{i+1} \geq t_i$. In the following, we will equivalently write $w = (a, t)$ with $a = (a_i)_{0 \leq i \leq n}$ and $t = (t_i)_{0 \leq i \leq n}$.

An interval I of ${\mathbb{R}^+}_{\geq 0}$ is a ${\mathbb{Q}^+}_{\geq 0}$*-interval* iff its left endpoint belongs to ${\mathbb{Q}^+}_{\geq 0}$ and its right endpoint belongs to ${\mathbb{Q}^+}_{\geq 0} \cup \{\infty\}$. We set $I^{\downarrow} = \{x \mid x \leq y \text{ for some } y \in I\}$, the *downward closure* of I. We denote by $\mathcal{I}({\mathbb{Q}^+}_{\geq 0})$ the set of ${\mathbb{Q}^+}_{\geq 0}$-intervals of ${\mathbb{R}^+}_{\geq 0}$.

A *valuation* v over a finite set X is a mapping in ${\mathbb{R}^+}_{\geq 0}^X$. We note $\mathbf{0}$ the valuation which assigns to every clock $x \in X$ the value 0. For any value $d \in {\mathbb{R}^+}_{\geq 0}$, the valuation $v + d$ is defined by $(v + d)(x) = v(x) + d,\ \forall x \in X$.

2.2 Timed Transition Systems and Timed Automata

Definition 1 (Timed Transition System (TTS)). *A* timed transition system *over Σ_ε is a transition system $S = (Q, q_0, \rightarrow)$, where Q is the set of states, $q_0 \in Q$ is the initial state, and the transition relation $\rightarrow$ consists of delay moves $q \xrightarrow{d} q'$ (with $d \in {\mathbb{R}^+}_{\geq 0}$), and discrete moves $q \xrightarrow{a} q'$ (with $a \in \Sigma_\varepsilon$). Moreover, we require standard properties of time-determinism, additivity and continuity for the transition relation $\rightarrow$.*

TTSs describe systems combining discrete and continuous evolutions. They are used to define and compare semantics of TPNs and TA. With these properties, a *run* of S can be defined as a finite sequence of moves $\rho = q_0 \xrightarrow{d_0} q_0' \xrightarrow{a_0} q_1 \xrightarrow{d_1} q_1' \xrightarrow{a_1} q_2 \ldots \xrightarrow{a_n} q_{n+1}$ where discrete actions and delays alternate, and which starts in the initial configuration. To such a run corresponds a word $a_0 \ldots a_n$ over Σ_ε; we say that this word is accepted by S. The language of S is the set of words accepted by S.

Let us now define *timed automata*. First defined in [3], the model of timed automata associates a set of non-negative real-valued variables called *clocks* with a finite automaton. Let X be a finite set of clocks. We write $\mathcal{C}(X)$ for the set of *constraints* over X, which consist of conjunctions of atomic formulae of the form $x \bowtie c$ for $x \in X$, $c \in {\mathbb{Q}^+}_{\geq 0}$ and $\bowtie \in \{<, \leq, \geq, >\}$. We also define the proper subset $\mathcal{C}_{ub}(X)$ of *upper bounds* constraints over X where $\bowtie \in \{<, \leq\}$.

Definition 2 (Timed Automata (TA)). *A* timed automaton *$\mathcal{A}$ over Σ_ε is a tuple (L, ℓ_0, X, E, Inv) where L is a finite set of* locations, *$\ell_0 \in L$ is the* initial location, *X is a finite set of* clocks, *$Inv \in \mathcal{C}_{ub}(X)^L$ assigns an* invariant *to each location and $E \subseteq L \times \mathcal{C}(X) \times \Sigma_\varepsilon \times 2^X \times L$ is a finite set of* edges. *An edge $e = (\ell, \gamma, a, R, \ell') \in E$ represents a transition from location ℓ to location ℓ' labeled by a with constraint γ and reset $R \subseteq X$.*

Semantics. For $R \subseteq X$, the valuation $v[R]$ is the valuation v' such that $v'(x) = v(x)$ when $x \notin R$ and $v'(x) = 0$ otherwise. Finally, constraints of $\mathcal{C}(X)$ are interpreted over valuations: we write $v \models \gamma$ when the constraint γ is satisfied by v. The semantics of a TA $\mathcal{A} = (L, \ell_0, X, E, Inv)$ is the TTS $[\![\mathcal{A}]\!] = (Q, q_0, \rightarrow)$ where $Q = \{(\ell, v) \in L \times ({\mathbb{R}^+}_{\geq 0})^X \mid v \models Inv(\ell)\}$, $q_0 = (\ell_0, \mathbf{0})$ and $\rightarrow$ is defined by:

- **delay moves:** $(\ell, v) \xrightarrow{d} (\ell, v + d)$ if $d \in \mathbb{R^+}_{\geq 0}$ and $v + d \models Inv(\ell)$;
- **discrete moves:** $(\ell, v) \xrightarrow{a} (\ell', v')$ if there exists some $e = (\ell, \gamma, a, R, \ell') \in E$ s.t. $v \models \gamma$ and $v' = v[R]$.

The (untimed) language of $\mathcal{A}$ is defined as that of $[\![\mathcal{A}]\!]$ and is denoted by $\mathcal{L}(\mathcal{A})$.

2.3 Time Petri Nets

Introduced in [23], Time Petri nets (TPNs) are Petri nets which transitions are equipped with timing constraints. TPNs associate a time interval to each transition of a Petri net. As soon as a transition is enabled, a clock attached to this transition is reset and starts measuring time. A transition is then allowed to fire if it is enabled and if its clock's value lays within the time interval of the transition. When a TPN contains read arcs, places that are read can enable/disable a transition, but tokens from read places are not consumed at firing time. Transitions are represented as black rectangles, places as circles, flows as thick lines joining transitions ans places. Transitions can be labeled by an observable letter, or unobservable, and constraints are represented as intervals labeling transitions.

Definition 3 (Time Petri Nets (TPN)). *A time Petri net $\mathcal{N}$ over Σ_ε is a tuple (P, T, W, m_0, I) where P is a finite set of* places*, T is a finite set of* transitions *with $P \cap T = \emptyset$, $W : (P \times T) \cup (T \times P) \mapsto \mathbb{N}$ is the flow relation, $m_0 \in \mathbb{N}^P$ is the* initial *marking, and $I : T \mapsto \mathcal{I}(\mathbb{Q^+}_{\geq 0})$ associates with each transition a* firing interval*. We denote by $\alpha(t)$ (resp. $\beta(t)$) the lower bound (resp. the upper bound) of interval $I(t)$.*

We will frequently denote by ${}^\bullet(.) \in (\mathbb{N}^P)^T$ is the *backward* incidence mapping, that is the restriction of the flow relation to $P \times T$, and by $(.)^\bullet \in (\mathbb{N}^P)^T$ is the *forward* incidence mapping, that is the restriction of the flow relation to $(T \times P)$.

Time Petri nets define sequences of transitions firings and time elapsing. Transitions symbolize distinct events in a system. Now, one can add lableing to TPNs to increase the expressive power of the model, and model unobservable actions in a system.

Definition 4 (labeled Time Petri Nets (TPN)). *Let Σ_ε be an alphabet containing a special label ϵ. A labeled time Petri net (LTPN) $\mathcal{N}$ over Σ_ε is a tuple (N, λ) where N is a TPN, and $\lambda : T \rightarrow \Sigma_\varepsilon$ is a* labeling function.

Semantics. A *configuration* of a TPN is a pair (m, ν), where m is a *marking* in the usual sense, *i.e.* a mapping in $\mathbb{N}^P$, with $m(p)$ the number of tokens in place p. A transition t is *enabled* in a marking m if $m \geq {}^\bullet t$. We denote by $\mathit{En}(m)$ the set of enabled transitions in m. The second component of the pair (m, ν) is a valuation over $\mathit{En}(m)$ which associates to each enabled transition its age, *i.e.* the amount of time that has elapsed since this transition was last enabled. we choose the classical semantics defined as usual (see for instance [6]). An enabled transition t can be fired if $\nu(t)$ belongs to the interval $I(t)$. The result of this firing is as usual the new marking $m' = m - {}^\bullet t + t^\bullet$. Moreover, some valuations are reset. We say that the a transition t is *newly enabled* by firing of t from marking m, and write $\uparrow\!\mathit{enabled}(t', m, t)$ iff:

$$t' \in \mathit{En}(m - {}^\bullet t + t^\bullet) \wedge ((t' \notin \mathit{En}(m - {}^\bullet t)) \vee t = t')$$

Reset valuations correspond to newly enabled clocks. Thus, firing a transition is not an atomic step and the transition currently fired is always reset. The set $\mathit{ADM}(\mathcal{N})$ of *(admissible) configurations* consists of the pairs (m, ν) such that $\nu(t) \in I(t)^{\downarrow}$ for every transition $t \in \mathit{En}(m)$. Thus time can progress in a marking only when it does not leave the firing interval of any enabled transition. The semantics of a TPN $\mathcal{N} = (P, T, W, m_0, I)$ is a TTS $[\![\mathcal{N}]\!] = (Q, q_0, \rightarrow)$ where $Q = \mathit{ADM}(\mathcal{N})$, $q_0 = (m_0, \mathbf{0})$ and $\rightarrow$ is defined by:

- **delay moves:** $(m, \nu) \xrightarrow{d} (m, \nu + d)$ iff $\forall t \in \mathit{En}(m)$, $\nu(t) + d \in I(t)^{\downarrow}$,
- **discrete moves:** $(m, \nu) \xrightarrow{t} (m - {}^{\bullet}t + t^{\bullet}, \nu')$ iff $t \in \mathit{En}(m)$ is s.t. $\nu(t) \in I(t)$, $\forall t' \in \mathit{En}(m - {}^{\bullet}t + t^{\bullet})$, $\nu'(t') = 0$ if $\uparrow\!\mathit{enabled}(t', m, t)$ and $\nu'(t') = \nu(t)$ otherwise

The semantics of LTPN is defined alike, using $\lambda(t)$ as label of a transition rather than t. The (untimed) language of $\mathcal{N}$ is defined as the untimed language of $[\![\mathcal{N}]\!]$ and is denoted by $\mathcal{L}(\mathcal{N})$. The reachability set of $\mathcal{N}$, denoted $\mathsf{Reach}(\mathcal{N})$, is the set of markings $m \in \mathbb{N}^P$ such that there exists a reachable configuration (m, ν). A *bounded TPN* is a TPN $\mathcal{N}$ such that $\mathsf{Reach}(\mathcal{N})$ is finite.

A *safe TPN* is a TPN $\mathcal{N}$ where all configurations reachable in $[\![\mathcal{N}]\!]$ contain at most one token in every place. Very often, when a Petri net is safe, we will consider that the flow relation is a function from $W : (P \times T) \cup (T \times P)$ to $\{0, 1\}$.

TPNs can be represented by a bipartite graph. Places are symbolized by circles, transitions as a dark horizontal line. The flow relation is represented as an edge from a element x to element y if $W(x, y) \geq 1$ labeled by value $W(x, y)$. When $W(x, y)$ takes values in $\{0, 1\}$ we will simply draw an unlebeled edge fron x to y if $W(x, y) = 1$.

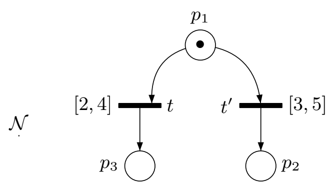


Fig. 1. An example of TPN

Figure 1 shows an example of safe TPN, with three places p_1, p_2, p_3, and two transitions t and t'. All flow relation are either $W(x, y) = 0$ or $W(x, y) = 1$. The time interval attached to transition t means that t should fire at earliest 2 time units after being enabled, and at latest 4 time units later. The time interval attached to transition t' means that t should fire at earliest 3 time units after being enabled, and at latest 5 time units later.

2.4 TPNs with read arcs

As for standard Petri nets, TPNs can be extended with read arcs, that is with a flow relation that does not enforce consumtion of tokens. This feature is useful to capture mechanisms such as guards, and in our context it will be used to specify how a given architecture constraints an original model by enabling or disabling transitions.

TPNS with read arcs are TPN in which some places can be read by transitions without consumption of the read tokens. Figure 2-a is an example of TPN with read arcs. Transitions are represented as black rectangles, places as circles, flows as thick lines joining transitions ans places, and dotted lines represent read arcs. Transitions can be labeled by an observable letter, or unobservable, and constraints are represented as intervals labeling transitions.

Definition 5 (place/transition net with read arcs). *A time Petri net with read arcs (TPNR for short) is a tuple $\mathcal{N} = (P, T, W, R, m_0, I)$ where $(P, T, W, m_0, \Lambda, I)$ is a TPN, and $R : (P \times T) \to \{0, 1\}$ s.t., $W^{-1}(1) \cap R^{-1}(1) = \emptyset$ is a new* flow relation

As for TPNs, we can add a labeling function λ to a TPNR. Every TPN (TPNR) can be seen as a union of an untimed Petri Net $N = (P, T, W, R)$ and of a timing function I. The untimed net N (with out without read arcs) will be called the *underlying net* of $\mathcal{N}$.

Semantics. A net with read arcs net defines a bipartite directed graph with two kinds of edges: there exists a (consume) arc from x to y (drawn as a solid line) iff $W(x, y) \geq 1$ and there exists a (read) arc from x to y (drawn as a dashed line) iff $R(x, y) = 1$. For all $x \in P \cup T$, we define the following sets: ${}^\bullet x = \{y \in P \cup T \mid W(y, x) = 1\}$ and $x^\bullet = \{y \in P \cup T \mid W(x, y) = 1\}$. For all $x \in T$, we define ${}^\circ x = \{y \in P \mid R(y, x) = 1\}$. These definitions extend naturally to subsets by considering union of sets. A transition $t \in T$ is said *enabled* by the marking m if $m(p) > 0$ for every place $p \in ({}^\bullet t \cup {}^\circ t)$. *En*$(()N, m)$ denotes the set of transitions of N enabled by m. As for TPNs, the firing of an enabled transition t produces a new marking m' computed as $\forall p \in P, m'(p) = m(p) - W(t, p) + W(p, t)$. We fix a marking m^0 of N called its *initial marking*. We say that a transition t' is in conflict with a transition t iff $({}^\bullet t \cup {}^\circ t) \cap ({}^\bullet t') \neq \emptyset$ (firing t' consumes tokens that enable t).

The semantics of a TPNR is also given as a timed transition system (TTS) [21]. This model contains two kinds of transitions: continuous transitions when time passes and discrete transitions when a transition of the net fires. A transition t_k is said *newly enabled* by the firing of the firable transition t_i from the marking m, and denoted $\uparrow$*En*$(()t_k, m, t_i)$, if the transition t_k is enabled by the new marking $(m \setminus {}^\bullet t_i) \cup t_i^\bullet$ but was not by $m \setminus ({}^\bullet t_i)$. We will denote by $\uparrow$*En*$(()m, t_i)$ the set of transitions newly enabled by the firing of t_i from m. A valuation is a map $\nu : T \to \mathbb{R}^+$ such that $\forall t \in T, \nu(t)$ is the time elapsed since t was last newly enabled. For $\delta \in \mathbb{R}^+$, $\nu + \delta$ denotes the valuation that associates $\nu(t) + \delta$ to every transition $t \in T$. Note that $\nu(t)$ is meaningful only if t is an enabled transition. $\mathbf{0}$ is the null valuation such that $\forall t, \mathbf{0}(t) = 0$.

The semantics of TPN $\mathcal{N}$ is defined as the TTS $(Q, q_0, \to)$ where a state of Q is a couple (m, ν) of a marking and valuation of $\mathcal{N}$, $q_0 = (m_0, \mathbf{0})$ and $\to \in (Q \times (T \cup \mathbb{R}^+) \times Q)$ is the transition relation describing continuous and discrete transitions. The continuous transition relation is defined $\forall \delta \in \mathbb{R}^+$ by:

$$(m, \nu) \xrightarrow{\delta} (m, \nu') \text{ iff } \nu' = \nu + \delta \text{ and } \forall t_k \in \textit{En}(()m), \begin{cases} \nu'(t_k) \leq I_s^+(t_k) \text{ and } I_s(t_k) \text{is of the form } [a, b] \text{ or (a,b]} \\ \nu'(t_k) < I_s^+(t_k) \text{ and } I_s(t_k) \text{is of the form [a,b) or (a,b)} \end{cases}$$

Intuitively, time can progress iff letting time elapse does not violate the upper constraint $I_s^+(t)$ of any transition t. The discrete transition relation is defined $\forall t_i \in T$ by:

$$(m,\nu) \xrightarrow{t_i} (m',\nu') \text{ iff } \begin{cases} t_i \in \mathit{En}(()m), m' = (m \setminus {}^\bullet t_i) \cup t_i^\bullet \\ \nu(t_i) \in I_s(t_i), \\ \forall t_k, \nu'(t_k) = 0 \text{ if } \uparrow\!\mathit{En}(()t_k, m, t_i) \text{ and } \nu(t_k) \text{ otherwise.} \end{cases}$$

Intuitively, transition t_i can fire if it was enabled for a duration included in the time constraint $I_s(t)$. Firing t_i from m resets the clocks of newly enabled transitions.

A *run* of a TTS is a sequence of the form $p_1 \xrightarrow{\alpha_1} p_2 \xrightarrow{\alpha_2} \dots \xrightarrow{\alpha_n} p_n$ where $p_1 = q_0$, and for all $i \in \{2..n\}$, $(p_{i-1}, \alpha_i, p_i) \in \rightarrow$ and $\alpha_i = t_i \in T$ or $\alpha_i = \delta_i \in \mathbb{R}^+$. Each finite run defines a sequence over $(T \cup \mathbb{R}^+)^*$ from which we can obtain a *timed word over* T of the form $w = (t_1, d_1)(t_2, d_2)\dots(t_n, d_n)$ where each t_i is a transition and $d_i \in \mathbb{R}^+$ the time at which transition t_i is fired. More precisely, if the sequence of $\alpha_i's$ read by the run are of the form $\delta_0\delta_1\dots\delta_{k_1}t_1\delta_{k_1+1}\delta_{k_1+2}\dots\delta_{k_2}t_2\dots t_n$, then the timed word obtained is $(t_1, d_1)\dots(t_n, d_n)$ where $d_i = \sum_{0\le j\le k_i}\delta_j$. We define a *dated run* of a TPN $\mathcal{N}$ as the sequence of the form $q_1 \xrightarrow{(d_1,t_1)} q_2 \dots \xrightarrow{(d_n,t_n)} q_n$, where d_i's are the dates as defined above and each q_i is the state reached after firing t_i at date d_i.

We denote by $\mathcal{L}_{tw}(\mathcal{N})$ the timed words over T generated by the above semantics. This will be called the timed (transition) language of $\mathcal{N}$. We denote by $\mathcal{L}_w(\mathcal{N})$ the untimed language of sequences of transitions obtained by projecting onto the first component. Furthermore, given a timed word w over T, if we consider a subset of transitions $X \subseteq T$, we can project w onto X to obtain a timed word over X. We will denote this projected language by $\mathcal{L}_{tw}(\mathcal{N})|_X$. For simplicity, we did not consider final states in our TTS, and hence define prefix-closed languages as is standard in Petri nets. Our results will still continue to hold with an appropriate definition of final states.

In the rest of this section, we will limit the study of robustness to TPNRs where the underlying PN is *1-safe*, i.e., nets such that $\forall p \in P,\ m(p) \le 1$, for all reachable markings m in the underlying PN. Hence, we will also consider flows that take values in $\{0,1\}$. The reason for using a property of the underlying net is that deciding if an untimed PN is 1-safe is PSPACE-complete, whereas checking if a TPN (and consequently a TPNR) is bounded is undecidable [29]. Reachability of a marking m in a safe net is also PSPACE-complete [11]. For safe Petri nets a place contains either 0 or 1 token, hence we identify a marking m with the set of places p such that $m(p) = 1$.

2.5 Timed (bi)simulation :

Different models are frequently compared according to their untimed languages, to their timed languages. However, it is well known that language equivalence does not capture all operational differences between models. A more discriminating comparison among timed specification is through timed bisimulation.

Let $S = (Q, q_0, \rightarrow)$ and $S' = (Q', q_0', \rightarrow')$ be two TTSs. A relation $\mathcal{R} \subseteq Q \times Q'$ is a *timed simulation* if and only if, $(q_0, q_0') \in \mathcal{R}$ and for every $\sigma \in \Sigma_\epsilon \cup R$, $q_1 \in Q$, $q_1' \in Q'$ such that $(q_1, q_1') \in \mathcal{R}$, if $q_1 \xrightarrow{\sigma} q_2$, then there exists q_2' such that $q_1' \xrightarrow{\sigma} q_2'$ and $(q_2, q_2') \in \mathcal{R}$. We will say that S' *simulates* S and write $S \preceq S'$ when such a relation $\mathcal{R}$

among states of S and S' exists. If in addition $\mathcal{R}^{-1}$ is a timed simulation relation from S' to S, then we say that $\mathcal{R}$ is a timed bisimulation. We say that S and S' are *timed bisimilar* when such a relation $\mathcal{R}$ among states of S and S' exists, and write $S \approx S'$.

3 Robustness of Time Petri Nets under architectural constraints

This section addresses robustness issues in Time Petri Nets (TPN) under constraints imposed by an external architecture. Within this context, the main objective is to check whether a timed specification, given as a TPN behaves as expected when subject to additional architectural constraints (for example, the use of resources,time and scheduling constraints,...). The constraints are specified by another TPN that constrains the specification via read arcs.

Hence, the composition of a TPN and of architectural constraints define a TPNR. Surprisingly, imposing constraints on a system may allow new transitions to fire, new markings to become reachable, etc. In this section, we want to check a robustness property that verifies that the constrained net does not exhibit new timed or untimed behaviors. Thus, if the implementation features can only restrict (but not enlarge) the set of original behaviors, we say the model is robust with respect to the implementation constraints.

We show that this property is not always guaranteed but that checking for it is always decidable in 1-safe TPNs. We further show that checking if the set of untimed behaviors of the constrained and specification nets are the same is also decidable. Next we turn to the more powerful case of labeled 1-safe TPNs with silent transitions. We show that checking for the robustness property is undecidable even when restricted to 1-safe TPNs with injective labeling, and exhibit a sub-class of safe TPNs (with silent transitions) for which robustness is guaranteed by construction. We demonstrate the practical utility of this sub-class with a case-study and prove that it already lies close to the frontiers of intractability.

Note that a large part of the litterature devoted to robustness problems use timed automata as model, and consider robustness properties such as invariance of behaviors under small time perturbations. We use the term "robustness" is a more general context: we consider preservation of specified behaviors when new architectural constraints (scheduling policies, resources, ...) are imposed.

We consider bipartite architectures: a specification of a distributed system is given as a TPN, called the *ground net* and the architectural constraints are specified by another TPN, called the *controller*. The controller net can read places of the ground net, but cannot consume tokens from the ground net, and vice versa. The net obtained by considering the ground net in the presence of the controller is called the *controlled net*.

Though this problem resembles supervisory control, there are some important differences. Supervisory control is used to restrict the behaviors of a system in order to meet some (safety) property P. The input of the problem is the property P, a description of the system, and the output a controller that restricts system: the behavior of a system under control is a subset of the original specification satisfying P. In our setting, there is no property to ensure, but we want to preserve as much as possible the specified behaviors. We will show in the example below that architectural contraints may add behaviors to the specification. This situation can be particularly harmful, especially when the architecture changes for a system that has been running properly on a former architecture. New faults that were not expected may appear, even when the overall performance of the architecture improves. Detecting such situations is a difficult task that should be automated. The last difference with supervisory control is that we do not ask for synthesis of a controller. In our setting, the controller represents the architectural constraints, and is part of the input of the robustness problem. The question is then whether the ground net preserves its behaviors when controlled.

More specifically, we consider the following questions. We first ask if the untimed language of the controlled net is contained in the untimed language of the ground net. This problem is called *untimed robustness*. Next, we ask if the untimed language is exactly the same despite control, called the *untimed equivalence problem*. The last problem considered is *timed robustness*, which asks if the timed language of the controlled net is contained in the timed language of the ground net.

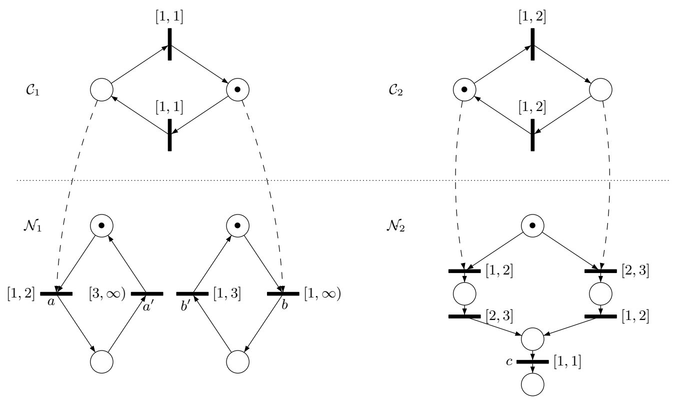


Fig. 2. Illustrative examples (a) and (b) - (unlabeled transitions depict silent moves)

Let us consider the example of Figure 2-a. It contains a ground net $\mathcal{N}_1$, with four transitions a, a', b, b', and a controller $\mathcal{C}_1$, that acts as a global scheduler allowing firing of a or b. In $\mathcal{N}_1$, transitions a, a' and b, b' are independant. The net $\mathcal{N}_1$ **is not** timed robust w.r.t. the scheduling imposed by $\mathcal{C}_1$: in the controlled net, a can be fired at time 3 which is impossible in $\mathcal{N}_1$ alone. However, if we consider the restriction of $\mathcal{N}_1$ to b, b', the resulting subnet is timed robust w.r.t $\mathcal{C}_1$. Figure 2-b shows a ground net $\mathcal{N}_2$ with four unobservable transitions, and one observable transition c. This transition can be fired at different dates, depending on wheter the first transition to fire is the left transition (with constraint $[1, 2]$) or the right transition (with constraint $[2, 3]$) below the initially marked place. The net $\mathcal{C}_2$ imposes that left and right transitions are not enabled at the same time, and switches the enabled transition from time to time. With the constraints imposed by $\mathcal{C}_2$, c is firable at date 5 in the controlled net but not at date 6 while it is firable at both dates 5 and 6 in $\mathcal{N}_2$ alone. This example is timed robust w.r.t $\mathcal{C}_2$, as it allows a subset of its original behaviors.

Our results are the following: The problem of checking untimed robustness for 1-safe TPNs is decidable. The timed variant of this problem is decidable for 1-safe TPNs, under the assumption that there are no ϵ transitions and the labeling of the ground net is injective. However, with arbitrary labeling and silent transitions this problem becomes undecidable. Further, even with injective labeling, timed robustness is undecidable as soon as the ground net contains silent transitions. We then show a natural relaxation on the way transitions are controlled and constrained, which ensures timed robustness of nets. In the untimed setting we also consider the stronger notion of equivalence of untimed languages and show that it is always decidable to check this property with or without silent transitions. The rest of the section is organized as follows: Section 3.1 introduces the robustness under architectural constraints problems. Section 3.2 shows decidability of this robustness problem in the untimed setting, or when nets are unlabelled. Section 3.3 shows that this problem becomes undecidable in the timed setting as soon as silent transitions are introduced. Section 3.4 shows conditions on ground nets and control schemes ensuring timed robustness. Section 3.5 provides a small case-study to show the relevance of our robustness condition, before concluding with Section 3.6.

Several papers deal with control of Petri Nets where transitions are divided into untimed controllable and uncontrollable transitions. Among them, Holloway and Krogh [18] first proposed an efficient method to solve a control problem for a subclass of Petri Nets called *safe marked graph*. Concerning TPNs, [15] propose a method inspired by the approach of Maler [22]. The controller is synthesized as a feedback function over the state space. However, in all these papers, the controller is given as a feedback law, and it is not possible to design a net model of the controlled system. To overcome this problem, [17] propose a solution using *monitors* to synthesise a Petri Net that models the closed-loop system. The method is extended to real time Supervisory Control in [28]. The supervisor uses enabling arcs (which are equivalent to read arcs) to enable or block a controllable transition. In [30], robustness is addressed in a weaker setting called *schedulability*: given an TPN N, the question is whether the untimed language of N, and the language of the underlying untimed net (i.e. without timing constraints) is the same. This problem is addressed for acyclic nets, or with restricted cyclic behaviors.

3.1 Formalization of robustness under architectural constraints

Let us consider two safe Time Petri nets (with read arcs) $\mathcal{N} = (P_\mathcal{N}, T_\mathcal{N}, W_\mathcal{N}, R_\mathcal{N}, I_\mathcal{N}, m^0_\mathcal{N})$ and $\mathcal{C} = (P_\mathcal{C}, T_\mathcal{C}, W_\mathcal{C}, R_\mathcal{C}, I_\mathcal{N}, m^0_\mathcal{C})$.

$\mathcal{C}$ models time constraints and resources of an architecture. One can expect these constraints to restrict the behaviors of the original net (we will show however that this is not always the case), that is $\mathcal{C}$ could be seen as a controller. Rather than synchronizing the two nets (as is often done in supervisory control), we define a relation $R \subseteq (P_\mathcal{C} \times T_\mathcal{N}) \cup (P_\mathcal{N} \times T_\mathcal{C})$, connecting some places of $\mathcal{C}$ to some transitions of $\mathcal{N}$ and vice versa. The resulting net $\mathcal{N}^{(\mathcal{C},R)}$ is still a place/transition net defined by $\mathcal{N}^{(\mathcal{C},R)} = (P_\mathcal{N} \cup P_\mathcal{C}, T_\mathcal{N} \cup T_\mathcal{C}, W_\mathcal{N} \cup W_\mathcal{C}, R_\mathcal{N} \cup R_\mathcal{C} \cup R, I_\mathcal{N} \cup I_\mathcal{C}, m^0_\mathcal{N} \cup m^0_\mathcal{C})$. We call $\mathcal{N}$ the *ground net*, $\mathcal{C}$ the *controller net* and $\mathcal{N}^{(\mathcal{C},R)}$ the *controlled net*.

The reason for choosing this relation is two-fold. Firstly, the definition of control above preserves the formalism as the resulting structure is a time Petri net as well. This allows us to deal with a single formalism throughout the paper. Secondly, one can define several types of controllers. By allowing read arcs from the controller to the ground net only, we model blind controllers, whose states evolve independently of the ground net's state. The net in Figure 2(a) is an example of such a controlled net. Conversely, if read arcs are allowed from the ground net to the controller, controller's state changes depending on the current state of the ground net. For the sake of clarity, all examples in the paper have blind controllers, but both types of control are possible.

Our goal is to compare the behaviors of $\mathcal{N}$ with its behaviors when controlled by $\mathcal{C}$ under R, i.e., $\mathcal{N}^{(\mathcal{C},R)}$. Therefore, the language of (timed and untimed) transitions, i.e., $\mathcal{L}_{tw}(\mathcal{N}), \mathcal{L}_{tw}(\mathcal{C}), \mathcal{L}_{w}(\mathcal{N}), \mathcal{L}_{w}(\mathcal{C})$, are as usual but when talking about the language of the controlled net, we will always mean the language projected onto transitions of $\mathcal{N}$, i.e., $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)})|_{T_\mathcal{N}}$ or $\mathcal{L}_{w}(\mathcal{N}^{(\mathcal{C},R)})|_{T_\mathcal{N}}$. Abusing notation, we will write $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)})$ (similarly $\mathcal{L}_{w}(\mathcal{N}^{(\mathcal{C},R)})$) to denote their projections onto $T_\mathcal{N}$.

We will now formally define and motivate the problems that we consider in this work.

Definition 6. *Given 1-safe TPNs $\mathcal{N}$ and $\mathcal{C}$, and a set of read arcs $R \subseteq (P_\mathcal{C} \times T_\mathcal{N}) \cup (P_\mathcal{N} \times T_\mathcal{C})$, $\mathcal{N}$ is said to be* untimed robust *under $(\mathcal{C}, R)$ if $\mathcal{L}_w(\mathcal{N}^{\mathcal{C},R}) \subseteq \mathcal{L}_w(\mathcal{N})$.*

For time Petri nets, the first problem we consider is the *untimed robustness* problem, which asks whether a given TPN $\mathcal{N}$ is untimed robust under $(\mathcal{C}, R)$. This corresponds to checking whether the controlled net $\mathcal{N}^{(\mathcal{C},R)}$ only exhibits a subset of the (untimed) behaviors of the ground TPN $\mathcal{N}$. The second question addressed is the *untimed equivalence* problem, which asks if the untimed behaviors of the controlled net $\mathcal{N}^{(\mathcal{C},R)}$ and ground net $\mathcal{N}$ are the same, i.e., if $\mathcal{L}_w(\mathcal{N}^{\mathcal{C},R}) = \mathcal{L}_w(\mathcal{N})$. In fact these questions can already be asked for "untimed Petri nets", i.e., for Petri nets without the timing function I_s and we also provide results for this setting.

Note however that untimed robustness only says that every *untimed* behavior of the controlled net $\mathcal{N}^{(\mathcal{C},R)}$ is also exhibited by the ground net $\mathcal{N}$. However some *timed* behaviors of the controlled net $\mathcal{N}^{(\mathcal{C},R)}$ may *not* be timed behaviors of the ground net $\mathcal{N}$. For obvious safety reasons, one may require that a controlled system does not allow new behaviors, timed or untimed. Surprisingly, controlling a TPN with another net may introduce additional behaviors on the ground net (this will be shown on an example in the next subsection). This contradicts the intuition that adding a controller restricts the possible behaviors of the controlled system, but indeed, adding ressource constraints or time sharing may allow firing of transitions that were not allowed, increase the set of reachable markings of the ground net, ... This counterintuitive situation is particcularly dangerous. Thus, we would like to check that even when considering timed behaviors, the set of timed behaviors exhibited by the controlled net $\mathcal{N}^{(\mathcal{C},R)}$ is a subset of the set of timed behaviors exhibited by the ground net $\mathcal{N}$. We call this the *timed robustness property*.

Definition 7. *Given 1-safe TPNs $\mathcal{N}$ and $\mathcal{C}$, and a set of read arcs $R \subseteq (P_\mathcal{C} \times T_\mathcal{N}) \cup (P_\mathcal{N} \times T_\mathcal{C})$, $\mathcal{N}$ is said to be* timed robust *under $(\mathcal{C}, R)$ if $\mathcal{L}_{tw}(\mathcal{N}^{\mathcal{C},R}) \subseteq \mathcal{L}_{tw}(\mathcal{N})$.*

One can further ask if the timed behaviors are exactly the same, which means that the controller is useless. Brought back to our setting, it means that the architectural constraints do not affect the executions of the system, nor their timings. While untimed equivalence of unconstrained and constrained systems seems a reasonable notion, timed equivalence is rarely met, and hence seems a too restrictive requirement. We will see in Section 3.3 that introducing silent transitions gives a new meaning to these notions.

3.2 Controlling (time) Petri nets

Let us first consider untimed 1-safe Petri nets. Let N be an untimed net, and C be an untimed controller. We can observe that C can only restrict the behaviors of N, under *any* choice of R. Hence N is always untimed robust under (C, R). Furthermore one can effectively check if the controlled net has the same untimed language as the ground net, by building their marking graphs, and then checking inclusion. Thus, the robustness and equivalence problems are decidable for untimed nets.

Proposition 1. *Let N, C be two* untimed *1-safe Petri nets. Then,*

1. *For any $R \subseteq (P_C \times T_N) \cup (P_N \times T_C)$, N is untimed robust under (C, R).*
2. *For a fixed set of read arcs $R \subseteq (P_C \times T_N) \cup (P_N \times T_C)$ checking if $\mathcal{L}_w(N) = \mathcal{L}_w(N^{(C,R)})$ is PSPACE-complete.*

Part $1)$ comes from the fact that a controller only restrics the set of reachable markings. Part $2)$ comes after demonstration that it is sufficient to show inclusion $\mathcal{L}_w(N) \subseteq \mathcal{L}_w(N^{(C,R)})$, which can be done by exploration of the marking graph of the controlled net.

This property of untimed Petri nets has a counterpart for time Petri nets: let us consider *unconstrained* nets $\mathcal{N}$ and $\mathcal{C}$, i.e., such that $I_{\mathcal{N}}(t) = [0, \infty)$ for every $t \in T_{\mathcal{N}}$, and $I_{\mathcal{C}}(t) = [0, \infty)$ for every $t \in T_{\mathcal{C}}$. Let N and C be the underlying nets of $\mathcal{N}$ and $\mathcal{C}$. One can easily show that for any R, $\mathcal{L}_w(\mathcal{N}^{\mathcal{C},R}) \subseteq \mathcal{L}_w(\mathcal{N})$. As any timed word $w = (a_1, d_1) \ldots (a_n, d_n)$ in $\mathcal{L}_{tw}(\mathcal{N}^{\mathcal{C},R})$ (resp. in $\mathcal{L}_{tw}(\mathcal{N})$) is such that $a_1 \ldots a_n \in \mathcal{L}_w(N^{C,R})$ (resp. $\mathcal{L}_w(N)$) where each $d_1, \ldots d_n$ can be arbitrary dates, we also have $\mathcal{L}_{tw}(\mathcal{N}^{\mathcal{C},R}) \subseteq \mathcal{L}_{tw}(\mathcal{N})$. Thus, unconstrained time Petri nets are also untimed robust.

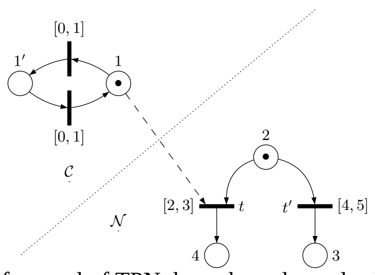


Fig. 3. An example of control of TPN through read-arcs leading to new behaviors

The question for Time Petri Nets is whether the controlled TPN only restricts the set of behaviors of the original TPN. Unlike in the untimed case, in the timed setting the controlled TPN may exhibit more (in fact, different set of) behaviors than the ground TPN, because of the urgency requirement of TPNs. Consider the example in Figure 3. The ground net $\mathcal{N}$ always fires t in the absence of the controller $\mathcal{C}$ but in the presence of $\mathcal{C}$ with R as in the picture, transition t is never fired and t' is always fired. Thus set of (timed and untimed) behaviors of $\mathcal{N}$ and $\mathcal{N}^{(\mathcal{C},R)}$ are disjoint. Discrepancies between untimed languages can be checked using the state class graph construction [6, 21]. This gives the following theorem and its corollary:

Theorem 1. *For 1-safe TPNs, the untimed robustness problem is PSPACE-complete.*

Corollary 1. *For 1-safe TPNs, the untimed equivalence problem is PSPACE-complete.*

Next we consider timed robustness properties for TPNs. Then, we have

Theorem 2. *For 1-safe TPNs, the timed robustness problem is decidable.*

Proof (sketch). Let $\mathcal{N}$ and $\mathcal{C}$ be 1-safe TPNs, and R be a set of read arcs. We can check if $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)}) \subseteq \mathcal{L}_{tw}(\mathcal{N})$ by using the state class timed automata construction from [21]. It is shown that from the state class graph construction of a 1-safe TPN, $\mathcal{N}$, we can build a deterministic timed automaton $\mathcal{A}$ over the alphabet $T_{\mathcal{N}}$, called the state class timed automaton, such that $\mathcal{L}_{tw}(\mathcal{N}) = \mathcal{L}_{tw}(\mathcal{A})$. As a result, $\mathcal{L}_{tw}(\mathcal{N})$ can be complemented and its complement is accepted by some timed automaton $\mathcal{A}'$, which is computed from $\mathcal{A}$ (see [3] for details of complementation of deterministic timed automata). On the other hand, the state class timed automaton $\mathcal{B}$ constructed from $\mathcal{N}^{(\mathcal{C},R)}$ is over the language $T_{\mathcal{N}} \cup T_{\mathcal{C}}$. By projecting this language onto $T_{\mathcal{N}}$, we obtain the timed (transition) language $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)})$. We remark that the timed automaton corresponding to the projection, denoted $\mathcal{B}'$, can be easily obtained by replacing all transitions of $\mathcal{C}$ in the timed automaton $\mathcal{B}$ by ϵ-transitions [3, 5]. Now we just check if $\mathcal{L}_{tw}(\mathcal{B}') \cap \mathcal{L}_{tw}(\mathcal{A}') = \emptyset$, which is decidable in PSPACE [3] (in the sizes of $\mathcal{A}'$ and $\mathcal{B}'$). $\square$

3.3 Controlling TPNs with silent transitions

We now consider ground nets which may have silent or ϵ-transitions. The (timed and untimed) language of the ground net contains only sequences of observable (i.e., not ϵ) transitions and the robustness question asks if the controller introduces new timed behaviors with respect to this language of observable transitions. From a modeling perspective, robustness means that sequence of important actions remain unchanged with architectural constraints, and hence this property should hold. Silent transitions can be used to model unimportant or unobservable transitions in the ground net. In this setting, it is natural to require that control does not add to the language of important/observable transitions, while it may allow new changes in other transitions.

An example of such a control is given in the introduction in Figure 2 (b). In that example, the ground net has a unique critical (visible) action c. All other transitions are left unlabeled and so we do not care if the timed or untimed behaviors on those transitions are different in the ground and controlled nets. Then the timed robustness problem asks if c can occur in the controlled net at a date when it was not allowed to occur in the ground net. A more practical example will be studied in detail in Section 3.5.

With this as motivation, we introduce the class of *ϵ-TPN*, which are TPNs where some transitions may be silent, i.e. labeled by ϵ. The behavior of such nets is deterministic except on silent actions: from a configuration, if a discrete transition that is not labeled ϵ is fired, then the net reaches a unique successor marking.

Definition 8. *Let Σ be a finite set of labels containing a special label ϵ.*

1. *An LTPN over Σ is a structure $(\mathcal{N}, \lambda)$ where $\mathcal{N}$ is a TPN and $\lambda : T_{\mathcal{N}} \to \Sigma$ is the labeling function.*
2. *An ϵ-TPN is an LTPN $(\mathcal{N}, \lambda)$ over Σ such that, for all $t \in T_{\mathcal{N}}$, if $\lambda(t) \neq \epsilon$ then $\lambda(t) \neq \lambda(t')$ for any $t' \neq t \in T_{\mathcal{N}}$.*

For an ϵ-TPN or LTPN $\mathcal{N}$, its *timed* (resp. *untimed*) *language* denoted $\mathcal{L}_{tw}(\mathcal{N}, \lambda)$ (resp. $\mathcal{L}_{w}(\mathcal{N}, \lambda)$) is the set of timed (resp. untimed) words over $\Sigma \setminus \{\epsilon\}$ generated by the timed (resp. untimed) transition system , by ignoring the ϵ labels. A TPN $\mathcal{N}$ from Definition 5 can be seen as the LTPN $(\mathcal{N}, \lambda)$ over Σ such that for all $t \in T_{\mathcal{N}}$, $\lambda(t) = t$, that is, λ is the identity map. An ϵ-TPN can be seen as an LTPN $(\mathcal{N}, \lambda)$ over $\Sigma = T_{\mathcal{N}} \cup \{\epsilon\}$ such that $\lambda(t) = t$ or $\lambda(t) = \epsilon$ for all $t \in T_{\mathcal{N}}$.

We are interested in the problem of *checking timed robustness*, i.e.,

Definition 9. *Given two ϵ-TPNs $(\mathcal{N}, \lambda)$ and $(\mathcal{C}, \lambda')$ over Σ and a set of read arcs R from $(P_{\mathcal{C}} \times T_{\mathcal{N}}) \cup (P_{\mathcal{N}} \times T_{\mathcal{C}})$,*

- *the controlled ϵ-TPN $(\mathcal{N}, \lambda)^{(\mathcal{C},R)}$ is defined as the ϵ-TPN $(\mathcal{N}^{(\mathcal{C},R)}, \lambda'')$ over Σ where $\lambda''(t) = \lambda(t)$ for $t \in T_{\mathcal{N}}$ and $\lambda''(t) = \epsilon$ for $t \in T_{\mathcal{C}}$.*
- *the timed robustness problem asks if $\mathcal{L}_{tw}((\mathcal{N}, \lambda)^{\mathcal{C},R}) \subseteq \mathcal{L}_{tw}(\mathcal{N})$.*

Note that the labels in $\mathcal{C}$ are ignored (i.e., replaced by ϵ), since robustness only compares labels of the ground nets. We remark that untimed robustness and even untimed equivalence are decidable for ϵ-TPNs and LTPNs, since Theorem 1 still holds in the presence of ϵ or labels (indeed, the algorithm (see [1] for details) uses a construction called a state class graph, which is an untimed automaton). However, the result does not extend to timed robustness, and we can show that this problem is undecidable for ϵ-TPNs and LTPNs.

Theorem 3. *Checking timed robustness is undecidable for ϵ-TPNs (and LTPNs).*

The complete proof can be found in [1], but sketching its content highlights the power of labels. The first step of the proof reuse results from [4] which shows that LTPNs are as powerful, language-wise, as timed automata. These results mean that the universality problem for timed automata reduces to the universality problem for LTPNs, and hence universality for LTPNs is undecidable. Then, we can show that LTPNs can be simulated by ϵ-TPNs. Figure 4 beliow shows how to encode a labeled pair of transitions with transitions in an ϵ-TPN.

Thus, ϵ-TPNs are expressively as powerful as LTPNs. The last step of the proof shows that checking universality of a labeled net (which is undecidable) can be reduced to checking timed robustness of a related net.

Note that checking if $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)}) = \mathcal{L}_{tw}(\mathcal{N})$, i.e., the timed language equivalence is a weaker notion in the context of ϵ-TPNs than in TPN (it only requires preserving timing for important observable actions), and hence could be relevant. For instance in Figure 2(b), we may want to check if c can occur in the controlled net at every date at which it can occur in the ground net (even if the other ϵ-transitions are perturbed). Unfortunately, this simpler problem is also undecidable (it is an immediate corollary of the above theorem 6.

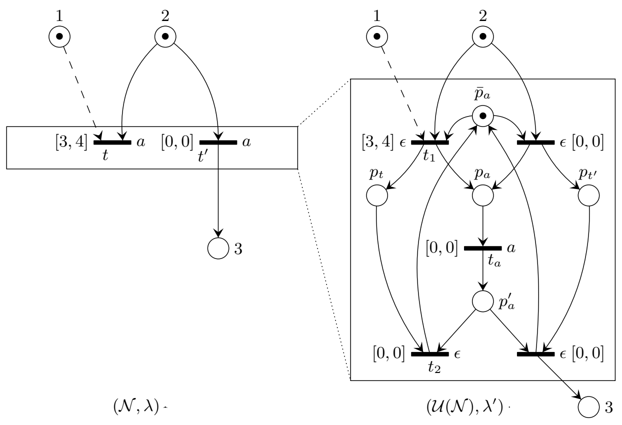


Fig. 4. Construction of a ϵ-TPN equivalent to a LTPN.

3.4 Ensuring robustness in TPNs with silent transitions

The situation for ϵ-TPNs is unsatisfactory since checking timed robustness is undecidable. Hence, we are interested in restrictions that make this problem decidable, or ensuring that this property is met by construction. In this section, we will show that we can restrict the controlling set of read-arcs to ensure that a net is always timed robust. Indeed, it is natural to expect that a "good" controller never introduces new behaviors and we would like to ensure this.

Here, we consider the restriction in which all transitions of the ground nets that have controller places in their preset are not urgent, i.e., the time constraint on the transition is $[\alpha,\infty)$ or (α,∞) for some $\alpha \in \mathbb{Q}^+$. We call such controlled nets *R-restricted ϵ-TPNs*. In this case we will show that R-restricted ϵ-TPNs are always timed robust (as in the case of untimed PNs shown in Proposition 1). That is,

Theorem 4. *Let $\mathcal{N}$ and $\mathcal{C}$ be two ϵ-TPNs, and R be a set of read arcs such that for every $(p,t) \in R \cap (P_{\mathcal{C}} \times T_{\mathcal{N}})$, $I_s(t)^+ = \infty$, then $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)}) \subseteq \mathcal{L}_{tw}(\mathcal{N})$.*

Note that while timed robustness is ensured for nets and control schemes that fulfill conditions of theorem 4, timed equivalence remains undecidable for such nets.

The condition in Theorem 4 is quite restrictive but relaxing it rapidly leads to undecidability:

Proposition 2. *The timed robustness problem is undecidable for ϵ-TPNs with at least one read arc from a place of the controller to any transition t of the ground net such that $I_s(t)^+ \neq \infty$.*

Again, one can show that this problem can be used to encode a universality problem.

3.5 A small case study

We consider a heater-cooler system depicted in Figure 5. This system improves the hardness of a particular material by first heating and then cooling it. The heater-cooler is equipped with two sensors: $Toohot$ is raised when the heater reaches its maximal temperature. If it occurs, the heating stops automatically. $Cold$ is raised when the temperature is cold enough in the cooling stage. If it occurs, the cooler stops automatically. The

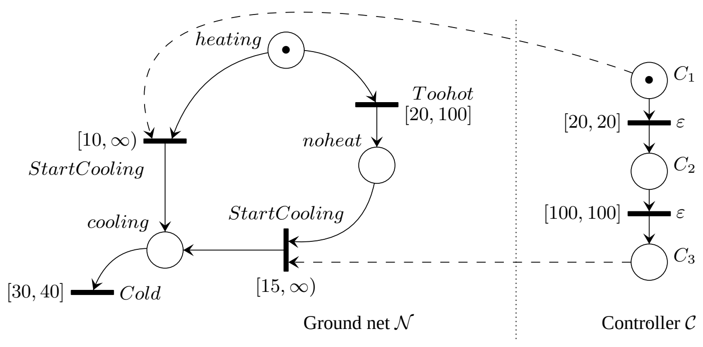


Fig. 5. Case Study

heater-cooler starts in the $heating$ state and the operator can push the $StartCooling$ button if the constraints of the system allow it.

We assume architectural constraints imposing that the $StartCooling$ action is not allowed after 20 t.u. in the heating stage, and also disallowed before the date 120 t.u. if the $toohot$ sensor has been raised. The constraints are encoded as a controller $\mathcal{C}$, and read arcs as shown in Figure 5.

We can show that $\mathcal{L}_w(\mathcal{N}^{\mathcal{C},R}) = \mathcal{L}_w(\mathcal{N})$. Hence, $\mathcal{N}$ is untimed robust and even untimed equivalent under $(\mathcal{C}, R)$. The net $\mathcal{N}$ is not an ϵ-TPN, but it was shown that any LTPN can be converted to an ϵ-TPN. Furthemore, the transformation only uses new transitions for the ground net, and no additional read arcs (as illustrated in Figure 4). Hence, the resulting net is R-restricted, so according to Theorem 4, we have $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)}) \subseteq \mathcal{L}_{tw}(\mathcal{N})$ and then $\mathcal{N}$ is timed robust under $(\mathcal{C}, R)$.

3.6 Conclusion and discussion

We have defined and studied notions of timed and untimed robustness as well as untimed equivalence for time Petri nets. We are interested in whether we can check or/and guarantee these properties for timed and untimed behaviors. We summarize the results obtained in the table below.

	TPN	R-restricted ϵ-TPN	ϵ-TPN	LTPN
Untimed robustness	Pc (thm 1)	G (thm 4)	Pc	Pc
Untimed equivalence	Pc (cor 1)	Pc	Pc	Pc
Timed robustness	D (thm 2)	G (thm 4)	U (thm 3)	U (thm 3)

U stands for undecidable, D for decidable, Pc for PSPACE-complete, and G for guaranteed.

Overall, with injective labels and no ϵ, robustness is decidable. We think that timed robustness of TPN is EXPSPACE-complete, but this need to be proved. However from a modeling perspective it is important to allow silent transitions. With silent transitions, untimed properties are still tractable, but timed properties become hard to check. To overcome this problem, we proposed a sufficient condition to guarantee timed robustness which we showed is already at the border of undecidability. To show its practical relevance, we designed a small case-study. We also show that untimed equivalence is easily decidable in all the cases. As for timed equivalence, this property is undecidable in most cases. This is not really a surprise nor a limitation, as asking preservation of timed behavior under architectural constraints is a rather strong requirement.

As further discussion, we remark that other criteria can be used for comparing the controlled and ground nets such as (timed) bisimulation or weak bisimulation. While this would be an interesting avenue to explore, a priori, they seem to be more restrictive and hence less viable from a modeling perspective. Possible extensions could be to define tractable subclasses of nets, for instance by considering semantic properties of the net rather that syntactic conditions ensuring decidability. It also seems possible to consider robustness of nets *up to some small delay*. Formally, we can fix a delay as a small positive number δ, and define $\mathcal{L}_{tw}^{\delta}(\mathcal{N}) = \{(w_1, t_1) \dots (w_n, t_n) \mid \exists (w_1, t_1') \dots (w_n, t_n') \in \mathcal{L}_{tw}(\mathcal{N}), \forall i \in 1 \dots n, |t_i' - t_i| \leq \delta\}$. Then a possible extension of the definitions is to consider δ-robustness under $\mathcal{C}, R$ as the timed inclusion $\mathcal{L}_{tw}(\mathcal{N}^{(\mathcal{C},R)}) \subseteq \mathcal{L}_{tw}^{\delta}(\mathcal{N})$.

4 Robustness of Time Petri Nets under Guard Enlargement

Robustness of timed systems aims at studying whether infinitesimal perturbations in clock values can result in new discrete behaviors. A model is robust if the set of discrete behaviors is preserved under arbitrarily small (but positive) perturbations. We tackle this problem for Time Petri Nets (TPNs for short) by considering the model of parametric guard enlargement which allows time-intervals constraining the firing of transitions in TPNs to be enlarged by a (positive) parameter.

We show that TPNs are not robust in general and checking if they are robust with respect to standard properties (such as boundedness, safety) is undecidable. We then extend the marking class timed automaton construction for TPNs to a parametric setting, and prove that it is compatible with guard enlargements. We apply this result to the (undecidable) class of TPNs which are robustly bounded (i.e., whose finite set of reachable markings remains finite under infinitesimal perturbations): we provide two decidable robustly bounded subclasses, and show that one can effectively build a timed automaton which is timed bisimilar even in presence of perturbations. This allows us to apply existing results for timed automata to these TPNs and show further robustness properties.

As already mentionned in Section 1, the semantics of timed models such as timed automata [3] (TA), time Petri nets [23] is idealized. Indeed, in implementations of timed systems, clock values are discretized, which may lead to approximations of real clock values. Furthermore, in distributed systems, the clocks of two different processes may evolve at slightly different rates. As a result, the extreme precision of the models leads to unexpected outcomes when there is even a slight imprecision at the level of implementation. A solution to discover if imprecision may cause a problem is to verify properties of models under perturbation, that is introduce perturbations in the models.

For timed automata, a model of guard enlargement has been extensively studied in the last decade [24, 7, 8, 13, 9, 26]. In [14], it is proven that this model of perturbation covers the both issues of discretization and drift of clocks, by reducing the implementability problem to the analysis of the enlarged semantics.

In this section, we tackle the problem of robustness under small perturbations in the distributed and timed setting of time Petri nets. Our aim is to study the effect of small enlargement of intervals that are attached to transitions of a TPN. We address mainly two problems. The first is the *robust boundedness* problem, which consists in deciding, for a given bounded TPN, whether there exists a positive enlargement for which the set of reachable markings is finite. The second problem considered in this section is *robust untimed language preservation*, which consists in deciding whether there exists a positive enlargement for which the untimed language remains unchanged. As mentioned, robustness issues have been well studied for TA. Hence, a possible way to address the robustness problem for TPNs is to translate TPNs to TA, and reuse existing techniques. However, we show in this section that results on TA do not always extend to TPN. For instance, robust safety, that is avoidance of some bad configuration under perturbation, is decidable in TAs, but not for TPNs. The objectives of this paper are to consider robustness issues for TPNs, and to study to what extent results proven for TA can be applied on TPN.

We first show that the phenomenon of accumulation of perturbations, which Puri exhibited in TA in [24], also occurs in TPN, but in a slightly different way. In a TPN, firing of transitions which are not causally related may occur systematically at distinct dates in a non-perturbed model, and after accumulation of some delays, become concurrent in the perturbed model. This has two consequences: first, reachable markings of a net may change under perturbation. Second, a bounded net may become unbounded under perturbation. This makes a huge difference with the TA model which is defined over a finite set of locations which does not change under perturbation. We show an example of a TPN whose unbounded perturbed semantics cannot be captured by a finite timed automaton. We then use this example to prove that the two problems we consider are undecidable.

There are several translations from TPN to TA [16, 10, 21, 12]. We study which of these translations can be used to lift robustness results on TA to the model of TPN. In particular, we prove that the marking class timed automaton construction of [12] is compatible with guard enlargement, in the sense that the property of timed bisimulation is preserved when guards are enlarged by the same parameter in the TA and in the TPN. We use this result to exhibit subclasses of bounded TPNs for which robust boundedness and language preservation are decidable.

This section is organized as follows: Section 4.1 introduces our time perturbation model for TPNs, and the robust boundedness and language preservation problems. Section 4.2 shown that in general, TPNs do not have robust languages, and are not robustly bounded either. This section also exhibits a trivial class of TPNs, namely the class of *sequential TPNs*, for which robustness problems are decidable.

Section 4.3 shows that many robustness issues are undecidable for TPNs. Section 4.4 presents a robust translation from TPNs to TA, i.e. compatible with guard enlargement. Sections 4.5 and 4.6 build on this result to exhibit decidable subclasses of TPNs, before conclusion. Missing proofs can be found in a complete version of this work published in [2].

4.1 Perturbations in TPN and robustness problems

The perturbation definition proposed hereafter is inspired from the perturbations in timed automata [7, 8, 13], that we describe hereafter. We start by fixing a parameter $\Delta \in \mathbb{R}^+_{\geq 0}$. Given a constraint $g \in \mathcal{C}(X)$, we define its Δ-enlargement as the constraint obtained by replacing any atomic formulae of the formulae $x \bowtie c$ for $x \in X$, $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >\}$, by the formulae $x \bowtie c + \Delta$ if $\bowtie \in \{<, \leq\}$, and by the formulae $x \bowtie c - \Delta$ if $\bowtie \in \{\geq, >\}$. Note that this transformation is a purely syntactic one. Now, given a timed automaton $\mathcal{A}$, we denote by $\mathcal{A}_\Delta$ the TA obtained by replacing every constraint by its Δ-enlarged version (both in guards and invariants). This model of perturbation verifies the following monotony property: for TA $\mathcal{A}$ and any $\Delta \leq \Delta' \in \mathbb{R}^+_{\geq 0}$, we have $[\![\mathcal{A}_\Delta]\!] \preceq [\![\mathcal{A}_{\Delta'}]\!]$. In the sequel, we will denote by $\mathsf{Reach}(\mathcal{A}_\Delta)$ the locations of $\mathcal{A}$ reachable in $[\![\mathcal{A}_\Delta]\!]$. We will also use the following important result:

Proposition 3 ([9]). *Let $\mathcal{A}$ be a timed automaton and S be a subset of locations of $\mathcal{A}$. One can decide whether there exists $\Delta \in \mathbb{Q}^+_{>0}$ such that $\mathsf{Reach}(\mathcal{A}_\Delta) \cap S = \emptyset$.*

We can now introduce a similar model of perturbation for Time Petri nets. Given an interval $I \in \mathcal{I}(\mathbb{Q}^+{}_{\geq 0})$, we denote by I_Δ the interval obtained by replacing its lower bound α by the bound $\max(0, \alpha - \Delta)$, and its upper bound β by the bound $\beta + \Delta$. Given a TPN $\mathcal{N}$, we denote by $\mathcal{N}_\Delta$ the TPN obtained by replacing every interval I by the interval I_Δ. We can then easily prove that the desired monotony property holds, entailing that if the system verifies a safety property for some perturbation Δ_0, it will also verify this property for any $\Delta \leq \Delta_0$:

Proposition 4. *Let $\mathcal{N}$ be a TPN and $\Delta \leq \Delta' \in \mathbb{R}^+{}_{\geq 0}$. We have $[\![\mathcal{N}_\Delta]\!] \preceq [\![\mathcal{N}_{\Delta'}]\!]$.*

These propositions are important as they intuitively state a “faster is better” property. Indeed, decreasing the perturbation also decreases the set of possible additional behaviors in the perturbed model of the system. This monotony property also entails that if the system verifies a safety property for some perturbation Δ_0, it will also verify this property for any $\Delta \leq \Delta_0$.

We now define robustness problems on TPNs in a way which is consistent with the monotony property stated above.

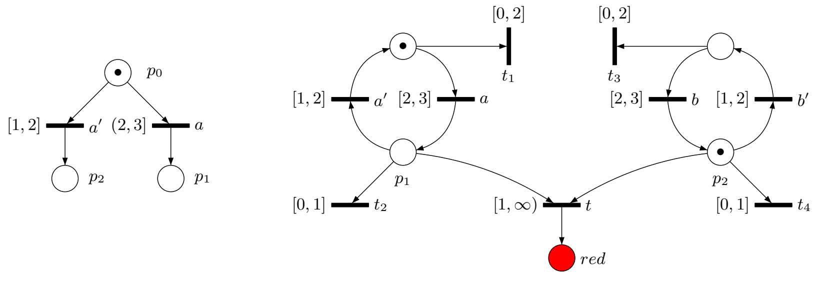
Robust Boundedness: Given a bounded TPN $\mathcal{N}$, does there exist $\Delta \in \mathbb{Q}^+{}_{>0}$ such that $\mathcal{N}_\Delta$ is bounded?

Robust Untimed language preservation: Given a bounded TPN $\mathcal{N}$, does there exist $\Delta \in \mathbb{Q}^+{}_{>0}$ such that $\mathcal{L}(\mathcal{N}_\Delta) = \mathcal{L}(\mathcal{N})$?

We call a TPN $\mathcal{N}$ *robustly bounded* if there exists $\Delta \in \mathbb{Q}^+{}_{>0}$ such that $\mathcal{N}_\Delta$ is bounded. This problem is strongly related to the problem of robust safety asking, given a bounded TPN $\mathcal{N}$ with set of places P, and a marking $m \in \mathbb{N}^P$, whether there exists $\Delta \in \mathbb{Q}^+{}_{>0}$ s.t., $\mathsf{Reach}(\mathcal{N}_\Delta)$ does not cover m. In fact, our undecidability and decidability results for robust boundedness will easily extend to this problem. However, the situation differs for robust untimed language preservation and so we treat this problem separately.

4.2 Examples of robust and non-robust TPNs

Consider the example in Figure 6(a). Due to the open interval and urgency condition (according to the semantics of TPNs, a' has to fire at most 2 time units after enabling), any enlargement of guards would result in reaching place p_1 which is not reachable in the non-enlarged semantics (from this simple example net, we can easily construct examples that are not robustly bounded or robustly safe). In this example, the firing domain of transition a (the set of configurations $\{(p_0, \nu) \mid \nu(a') \in [1, 2]\}$) is a *neighbor* of the reachable configuration $(p_0, \nu(a') = 2)$. By neighbor, we mean that any positive enlargement makes transition a fireable. This is the simplest form of non-robustness which can be easily checked for in bounded TPNs as one can compute a symbolic representation of the reachability set (using the state-class graph construction [6, 21] for instance). Further by requiring that all intervals must be closed, one may avoid this situation. Now assuming there are no transitions whose firing domain is a neighbor of the reachability set, one can prove that under a *bounded time horizon* (as defined for timed automata in [27],i.e. within a bounded number of steps) any net is robust, i.e., one can pick a sufficiently small $\Delta > 0$ to ensure that no new behavior occurs.



(a) The TPN $\mathcal{N}_0$. (b) The TPN $\mathcal{N}_1$.

Fig. 6. Two TPNs exhibiting new discrete behaviors under infinitesimal perturbations.

However, closing intervals does not make bounded nets robust, as illustrated by the following example. The remaining case concerns TPNs in which new behaviors are not neighbors of the reachability set, considered for an unbounded time horizon. In this case a new behavior cannot appear directly from a reachable configuration, and there must be several discrete firings before this new behavior is witnessed. Further the number of steps may depend on $\varDelta$: the smaller $\varDelta$ is, the larger will be the number of steps required. Intuitively, the new behavior is due to an accumulation of clock perturbations, rather than a single clock perturbation. Puri [24] gave an example of TA that exhibits accumulations, encoded using time between consecutive resets. However, for TPNs, this encoding does not work since the clocks are always reset when a transition is newly enabled.

We exhibit a TPN where accumulation is due to concurrency in Figure 6(b). This example can be simplified using singleton intervals, but we avoid this to show that accumulation may arise even without singletons. With the usual semantics, the red state in $\mathcal{N}_1$ is not reachable as transition t is never fireable. Indeed, one can verify that any run of $\mathcal{N}_1$ which does not fire transitions t_1, t_2, t_3 or t_4 always fires transition a (resp. a', b', b) at time $3k+2$ (resp. $3k+3$, $3k+1$, $3k+3$), for some integer k. By observing the time intervals of transitions t, a' and b', one can deduce that to be able to fire transition t, one has to fire simultaneously the transitions a and b, which is impossible.

Consider the net $(\mathcal{N}_1)_\varDelta$, for some positive $\varDelta$. We can prove that in this case, it is possible to fire simultaneously transitions a and b. In $(\mathcal{N}_1)_\varDelta$, one can delay the firing of transition a by up to $\varDelta$ time units. As a consequence, it is easy to verify that after n iterations of the loop aa', the timestamp of the firing of the last occurrence of a can be delayed by up to $n \cdot \varDelta$ time units. Choosing any $n \geq \frac{1}{\varDelta}$, we obtain the result. In particular, the red place is reachable in $(\mathcal{N}_1)_\varDelta$, for any positive $\varDelta$.

To see this, consider any reachable marking in any run which does not fire transitions t_1, t_2, t_3 or t_4. In any such run, there is a token in place p_1 exactly during the intervals $[3k-1, 3k]$ for all $k = 1, 2, \ldots$. Similarly, there is a token in p_2 exactly during the intervals $[3\ell, 3\ell+1]$ for all $\ell = 0, 1, \ldots$. As a result, there is a token in p_1 and p_2 enabling transition t only at every time instant $3k$ (for $k = 1, 2, \ldots$). But to fire t, we require at least 1 time unit to elapse once the transition is enabled, which never happens and hence t is never fireable. Thus, the red place is never reached.

However in the ϵ-enlarged semantics, we have the following run reaching the red place. As before, transition a fires at time $3k-1$ for $k = 1, 2, \ldots$. Thus the token is in place p_1 during the interval $[3k-1, 3k]$ for every $k = 1, 2, \ldots$. Letting $k = 2n+1$, this implies that there is a token at p_1 in the interval $[6n+2, 6n+3]$. On the other hand, each firing of transition b is delayed by some $\frac{1}{n} \leq \epsilon$ (indeed such an integer n exists given any fixed $\epsilon > 0$). All other transitions are not delayed and are fired exactly as they would have been under the exact semantics. Then, there is a token in place p_2 during the time intervals $[3\ell + \frac{\ell}{n}, 3\ell + \frac{\ell}{n} + 1]$ for $\ell = 0, 1, \ldots$. Letting $\ell = 2n$, this implies that there is a token in p_2 in the time interval $[6n+2, 6n+3]$. As a result transition t is enabled in the time interval $[6n+2, 6n+3]$ and so is fireable at the time instant $6n+3$, thus reaching the red place.

The accumulation in the above example is due to concurrent loops in the TPN. When we disallow such concurrency, we obtain a very simple class of *sequential TPNs* which is a strict subclass of timed automata. A TPN $\mathcal{N}$ is *sequential* if it satisfies the following property: for any reachable configuration (m, ν), and for any transitions $t, t' \in T$ that are fireable from (m, ν) (i.e. such that $t, t' \in En(m)$, $\nu(t) \geq \alpha(t)$ and $\nu(t') \geq \alpha(t')$), t and t' are in conflict, i.e. there exists a place p such that $m(p) < {}^{\bullet}t(p) + {}^{\bullet}t'(p)$.

The following theorem states robustness properties of sequential TPNs and their relation to timed automata.

Theorem 5. *We have the following properties:*

(i) *Checking whether a bounded TPN $\mathcal{N}$ is sequential is decidable.*
(ii) *If $\mathcal{N}$ is a sequential bounded TPN, then it can be translated into a timed automaton which resets every clock on each transition.*
(iii) *If $\mathcal{N}$ is sequential, then there exists $\Delta \in \mathbb{Q}_{>0}$ such that $\mathsf{Reach}(\mathcal{N}_\Delta) = \mathsf{Reach}(\mathcal{N})$ and $\mathcal{L}(\mathcal{N}_\Delta) = \mathcal{L}(\mathcal{N})$.*

Decidability follows from the construction of a state class graph, that is an automaton which states memorize markings and constraints on timed elapsed since a transition was enabled. Such graph recognizes the same language as the original net, and obviously can be used to compute the set of reachable markings. Building this graph is possible as soon as the TPN is bounded. Checking sequentiality of a net can be done in time linear in the size of the state class graph. The second and third properties follow from the observation that in a sequential TPN, each time a discrete transition is fired, each transition that is enabled in the new/resulting marking is newly enabled, and no memory is needed to remember the time since last enabling of a transition when moving to a new state. Thus, if this enabling tile is memorized by a clock, all clocks are reset at each transition, and this implies property (ii). Further, since clocks are reset, there is intuitively no memory in clock values. This forbids accumulation mechanisms, and considering as small enough enlargement (for instance $\Delta < \frac{1}{2}$) suffices to ensure that exactly the same transitions are enabled in $[\![\mathcal{N}]\!]$ and in $[\![\mathcal{N}_\Delta]\!]$.

The class of sequential nets is quite restrictive, as it does not allo for concurrent transitions in a net. For instance, the net example of Figure 6(b) is not sequential.

4.3 Undecidability results

We use the TPNs of Figure 6 to prove undecidability of robustness and untimed language preservation for bounded TPNs.

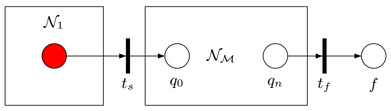


Fig. 7. TPN $\mathcal{N}_2$ obtained by combining $\mathcal{N}_1$ and $\mathcal{N}_\mathcal{M}$.

Theorem 6. *The problems of robust boundedness and robust untimed language preservation are undecidable for bounded TPN.*

Undecidability comes very easily, using the standard encoding of Minsky machines with TPNs (see for instance [25], or the appendix B in [2]). Given a Minsky machine $\mathcal{M}$, one can build a TPN $\mathcal{N}_\mathcal{M}$ such that $\mathcal{N}_\mathcal{M}$ is bounded iff $\mathcal{M}$ is, and $\mathcal{N}_\mathcal{M}$ covers some marking m iff $\mathcal{M}$ reaches its final state q_n. Moreover, the TPN $\mathcal{N}_\mathcal{M}$ is sequential (it encodes the behavior of $\mathcal{M}$ which is sequential).

The gadgets from Figure 6 and theorem 5 on sequential TPNs allow to extend all undecidability results attached to Minsky machines and their TPN encoding to robustness issues. We combine the TPNs $\mathcal{N}_1$ from Figure 6(b) and $\mathcal{N}_\mathcal{M}$ as depicted on Figure 7 to obtain the TPN $\mathcal{N}_2$. First note that $\mathcal{N}_2$ is a bounded TPN: without perturbation, transition t is never fired, and thus the set of reachable markings is finite. Second, we label transition t_f by a and every other transition by ε [6]. As $\mathcal{N}_\mathcal{M}$ is sequential, by Theorem 5(iii) it follows that (1) $\mathcal{N}_2$ is robustly bounded iff $\mathcal{N}_\mathcal{M}$ is bounded and (2) $\mathcal{N}_2$ robustly preserves its untimed language iff $\mathcal{N}_\mathcal{M}$ does not cover state m. We note that for (2), $\mathcal{N}_\mathcal{M}$ may not be bounded (if $\mathcal{M}$ is not bounded), however the statement still holds since Theorem 5(iii) does not require the boundedness assumption.

Thanks to undecidability of halting and boundedness of Minsky machines, the problems we considered are undecidable Remark that the above results also show that robust safety is undecidable, as $\mathcal{N}_2$ covers marking $\{f\}$ iff $\mathcal{N}_\mathcal{M}$ covers marking m.

4.4 A robust translation from TPN to TA

Robustness issues were first studied for timed automata, and several translations of TPN into TA exist in literature. It is hence natural to study which of these translations are compatible with robustness. A way to reduce robustness problems for TPNs to robustness problems for TA is to show that an existing timed bisimulation between TPN and its TA translation is preserved under perturbation. We now present a translation which verifies this property.

This construction is close to the marking class timed automaton construction of [12] but differs in two aspects. First, for efficiency reasons [12] reduce the number of clocks of the TA they build, and therefore use clock sharing techniques of [21], which may increase the number of locations. For ease of presentation, we do not consider this optimization, but our results also apply for this setting. Second, the construction of [12] was only stated for TPN whose underlying Petri net (i.e., the Petri net obtained by ignoring

[6] The reduction can be adapted to avoid the use of ε by labeling every other transition by b, and adding a gadget which can perform abitrarily many b's. It can however not be adapted to the setting of injective labeling, see Section 4.6.

the timing information in the given TPN) is bounded. We present the construction in a more general framework: we consider a TPN $\mathcal{N}$ which is not necessarily bounded and we consider as input a finite set of markings M. The construction is then restricted to the set M, and we can prove that it is correct for the set of behaviors of $\mathcal{N}$ which always remain within M. In the sequel, we will instantiate M depending on the context. For TPNs whose underlying PN is bounded, the construction of [12] is recovered by letting M be the set of reachable markings of this PN. We begin with a definition and a proposition that can be infered immediately:

Definition 10. *Let $\mathcal{N} = (P, T, \Sigma_\varepsilon, {}^\bullet(.), (.)^\bullet, m_0, \Lambda, I)$ be a TPN, $M \subseteq \mathbb{N}^P$ be a set of markings such that $m_0 \in M$, and let $[\![\mathcal{N}]\!] = (Q, q_0, \rightarrow)$ be the semantics of $\mathcal{N}$. The* M-bounded semantics of $\mathcal{N}$, *denoted $[\![\mathcal{N}]\!]_{|M}$, is defined as the restriction of the TTS $[\![\mathcal{N}]\!]$ to the set of states $\{(m, \nu) \in Q \mid m \in M\}$.*

Proposition 5. *Let M be a set of markings of a TPN $\mathcal{N}$ containing the initial marking. If $\mathsf{Reach}(\mathcal{N}) \subseteq M$, then $[\![\mathcal{N}]\!]_{|M} = [\![\mathcal{N}]\!]$.*

Now, let $(\mathcal{N}, \lambda)$ be a LTPN, and $M \subseteq \mathbb{N}^P$ be a finite set of markings such that $m_0 \in M$. The *marking timed automaton of $\mathcal{N}$ over M*, denoted $\mathcal{A}_M$, is defined as $\mathcal{A}_M = (M, m_0, X, \Sigma_\varepsilon, E, Inv)$, where $X = \{x_t \mid t \in T\}$, for each $m \in M$, $Inv(m) = \bigwedge_{t \in En(m)} x_t \leq \beta(t)$, and there is an edge $m \xrightarrow{g,a,R} m' \in E$ iff there exists $t \in T$ such that $t \in En(m)$, $m' = m - {}^\bullet t + t^\bullet$, g is defined as the constraint $x_t \in I(t)$, $a = \lambda(t)$ and $R = \{x_{t'} \mid t' \in \uparrow enabled(t', m, t) = \texttt{true}\}$. With this we have the following theorem:

Theorem 7. *Let $\mathcal{N}$ be a TPN, M be a finite set of markings containing the initial marking of $\mathcal{N}$, and $\mathcal{A}_M$ be the marking timed automaton of $\mathcal{N}$ over M. Then for all $\Delta \in \mathbb{Q}^+{}_{\geq 0}$, we have $[\![\mathcal{N}_\Delta]\!]_{|M} \approx [\![(\mathcal{A}_M)_\Delta]\!]$.*

Proof (Sketch). We can prove by induction that the following relation R is a timed bisimulation. Let (m, ν) (resp. (ℓ, v)) denote a state of the TTS $[\![\mathcal{N}_\Delta]\!]_{|M}$, i.e. $(m, \nu) \in Adm(\mathcal{N}_\Delta)$ with $m \in M$ (resp. denote a state of $[\![(\mathcal{A}_M)_\Delta]\!]$). We define $(m, \nu) R (\ell, v)$ if and only if $m = \ell$, and $\forall t \in En(m), \nu(t) = v(x_t)$. $\square$

Other TA constructions. The construction proposed in [21] builds a state class timed automaton incrementally using a forward exploration of reachable markings of a bounded TPN. Gardey et al [16] use a similar forward-reachability technique to build the reachable state space of TPN, where equivalence classes for clock valuations are encoded as zones. However, as in TPN $\mathcal{N}_1$ of Figure 6, new configurations in an *enlarged semantics* might be reached after accumulation of small delays. Hence, new reachable markings are not necessarily obtained in one enlarged step from a configuration in the non-enlarged semantics. Thus, forward techniques as in [21, 16] cannot be directly extended to obtain enlarged semantics and we need a more syntactic translation which builds an over-approximation of the reachable markings (of the TPN) as in Theorem 7.

Cassez et al [10] propose a different syntactic translation from unbounded TPNs by building a timed automaton for each transition, and then synchronizing them using a supervisor. The resulting timed automaton is bisimilar to the original model, but states contain variables, and hence the automaton may have an unbounded number of locations. It may be possible to extend this approach to address robustness problems, but as we focus on bounded TPNs, we leave this for future work.

4.5 Robustly bounded TPNs

This section focuses on the class of robustly bounded TPNs. By Theorem 6, we know that checking membership in this class is undecidable. We present two decidable subclasses, as well as a semi-decision procedure for the whole class. We first consider the subclass of TPNs whose *underlying Petri net* is bounded:

Proposition 6. *The set of TPN whose underlying net is bounded is a decidable subclass of robustly bounded TPNs. Further, for each net $\mathcal{N}$ of this class, one can construct a finite timed automaton $\mathcal{A}$ such that $[\![\mathcal{N}_\Delta]\!] \approx [\![\mathcal{A}_\Delta]\!]$ for all $\Delta \geq 0$.*

The decidability follows from that of boundedness for (untimed) Petri nets [19], as timing constraints can only restrict the set of untimed transitions. The second part of the above proposition follows from Theorem 7.

We now exhibit another subclass of robustly bounded TPNs whose underlying Petri nets can be unbounded. In fact, this class is incomparable with the above defined subclass. The following technical result is central in our approach:

Lemma 1. *Let $\mathcal{N}$ be a TPN, and M be a finite set of markings. Determining whether there exists $\Delta > 0$ such that $\mathsf{Reach}(\mathcal{N}_\Delta) \subseteq M$ is decidable.*

To prove this result, we define $\widetilde{M} = M \cup \{m' \mid \exists m \in M, t \in T, m' = m - {}^\bullet t + t^\bullet\}$ the (finite) set of markings reachable from M in at most one-step in the underlying Petri net. We then show that any run obtained after enlargement leading to a new marking necessarily goes throug a set of markings $\widetilde{M} \setminus M$.

In other terms, we show that

$$\mathsf{Reach}(\mathcal{N}_\Delta) \subseteq M \iff \mathsf{Reach}((\mathcal{A}_{\widetilde{M}})_\Delta) \subseteq M$$

Then, determining whether there exists $\Delta > 0$ such that the right hand side of the equivalence holds is decidable thanks to Proposition 3.

We consider the following subclass of bounded TPNs:

Definition 11. *A bounded TPN $\mathcal{N}$ is called Reach-Robust if $\mathsf{Reach}(\mathcal{N}_\Delta) = \mathsf{Reach}(\mathcal{N})$ for some $\Delta > 0$. We denote by* RR *the class of Reach-Robust TPNs.*

RR is the class of bounded TPNs whose set of reachable markings is invariant under some guard enlargement. It is easy to see that these nets are robustly bounded. More interestingly, checking membership in this class is decidable, i.e., given a bounded TPN $\mathcal{N}$ we can decide if there is a positive guard enlargement under which the set of reachable markings remains unchanged. This follows from Lemma 1, by instantiating the finite set of markings M with $\mathsf{Reach}(\mathcal{N})$:

Theorem 8. *RR is a decidable subclass of robustly bounded TPNs.*

We can now address properties of the general class of robustly bounded TPN.

Lemma 2. *The set of robustly bounded TPNs is recursively enumerable. Moreover, given a robustly bounded TPN $\mathcal{N}$, we can build effectively a timed automaton $\mathcal{A}$ such that there exists $\Delta_0 > 0$ for which, $\forall 0 \leq \Delta \leq \Delta_0$, $[\![\mathcal{N}_\Delta]\!] \approx [\![\mathcal{A}_\Delta]\!]$.*

Observe that a TPN $\mathcal{N}$ is robustly bounded iff there exists a finite set of markings M and some $\Delta > 0$ such that $\mathsf{Reach}(\mathcal{N}_\Delta) \subseteq M$. Thus by naively enumerating the *set* of finite sets of markings and applying the algorithm of Lemma 1 at each step of the enumeration, we obtain a semi-decision procedure (to check membership) for the class of robustly bounded TPNs. For the second result, observe that if $\mathcal{N}$ is known to be robustly bounded, then this semi-decision procedure terminates and computes a finite set of markings M and there is a value Δ_0 such that $\mathsf{Reach}(\mathcal{N}_{\Delta_0}) \subseteq M$. Therefore, for any $\Delta \leq \Delta_0$, $\mathsf{Reach}(\mathcal{N}_\Delta) \subseteq M$. By Proposition 5, this entails $[\![\mathcal{N}_\Delta]\!]_{|M} = [\![\mathcal{N}_\Delta]\!]$. In addition, by Theorem 7, we have $[\![\mathcal{N}_\Delta]\!]_{|M} \approx [\![(\mathcal{A}_M)_\Delta]\!]$ where $\mathcal{A}_M$ is the marking timed automaton of the TPN $\mathcal{N}$. Thus we have $\forall 0 \leq \Delta \leq \Delta_0,\ [\![\mathcal{N}_\Delta]\!] \approx [\![(\mathcal{A}_M)_\Delta]\!]$. This result allows us to transfer existing robustness results for timed automata to TPNs. We will illustrate the use of this property in the following section.

4.6 Untimed language robustness in TPNs

We now consider the robust untimed language preservation problem, which was shown undecidable in general in Theorem 6. We show that for the subclass of *distinctly labeled bounded TPNs* (i.e., labels on transitions are all distinct, and different from ε) this problem becomes decidable.

Definition 12. *A bounded TPN $\mathcal{N}$ is called Language-Robust if $\mathcal{L}(\mathcal{N}_\Delta) = \mathcal{L}(\mathcal{N})$ for some $\Delta > 0$. We denote by LR the class of Language-Robust nets and by LR$_{\neq}$ (resp. RR$_{\neq}$) the subclass of LR (resp. RR) with distinct labeling.*

We first compare the class RR (for which checking membership is decidable by Theorem 8) with the class LR (where, as already noted, checking membership is undecidable by Theorem 6). We can then observe that:

Proposition 7. *The classes RR and LR are incomparable w.r.t. set inclusion. Further, the class LR$_{\neq}$ is strictly contained in the class RR$_{\neq}$.*

Finally, we show that the problem of robust untimed language preservation becomes decidable under this assumption:

Theorem 9. *The class LR$_{\neq}$ is decidable, i.e., checking if a distinctly labeled bounded TPN is in LR is decidable.*

The proof of this result comes from decidability of the RR, and consequently also of the RR$_{\neq}$ subclass of TPN, and from the inclusion of LR$_{\neq}$ into RR$_{\neq}$. When a net is not in RR$_{\neq}$, then it is not in LR$_{\neq}$. Otherwise, $\mathsf{Reach}(\mathcal{N})$ is bounded, and by Lemma 2, we can build a timed automaton $\mathcal{A}$ which is timed bisimilar to $\mathcal{N}$ for small perturbations. This entails that this TA preserves its untimed language under small perturbations iff $\mathcal{N}$ does. Thus we have reduced the problem of checking if $\mathcal{N}$ is in LR$_{\neq}$ to checking if

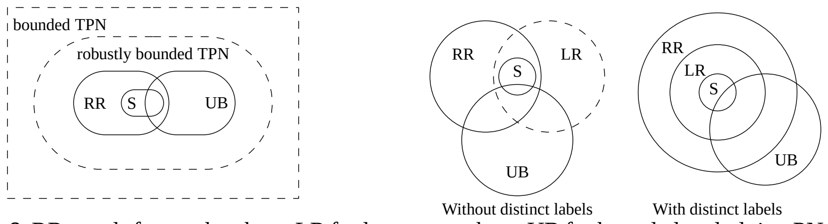


Fig. 8. RR stands for reach-robust, LR for language-robust, UB for bounded underlying PNs, S for sequential bounded TPNs. Dotted lines represent undecidable and solid lines decidable classes.

the timed automaton $\mathcal{A}$ constructed from $\mathcal{N}$ is language-robust, which is decidable for timed automata.

The results of this section and relation between subclasses of TPNs are summarized in the above diagram. We say that a class is decidable (resp. undecidable) when membership of a net in that class is a decidable problem (resp. undecidable). Note that the larger part of the results obtained in this section come from a transfer of positive results from the TA setting to TPNs. Unsurprisingly, several problems become undecidable in TPNs due to unboundedness. This does not mean however that unboundedness necessarily leads to undecidability of all robustness issues in general. As future work, we would like to show positive results in an unbounded setting and we believe that this would require a different approach and new techniques.

5 Conclusions and future work

The work performed during the first period of the IMPRO project shows that robustness with respect to architectural constraints (modeled as a controller net) is decidable for safe nets in the following cases :

- When the considered robustness property (untimed language inclusion or equivalence) does not address time issues.
- When the considered net and its controller do not contain epsilon transitions, or equivalently are not labeled nets. This means that in the considered Petri net models, transitions model explicitly a single event in the execution flow of a system, and not the occurrence of an action that may occur at several places in the control flow.

Let us now discuss the adequacy of the model with needs of developpers. Controllers can be used to model time sharing architectures. When several processes share a processor or a ressource, this sharing can be modeled by assigning to each process a place that is read by all transitions from the process, and moving a token among these particular places according to a scheduling policy. However, this is not yet satisfactory, as removing a token from a place means forgetting the time elapsed by a transition waiting for some delay before firing. This is bothering when one tries to model the duration of a task. A recent model have been proposed by members of IMPRO to memorize time elapsed in a place by token, but at the cost of high udecidability [20].

For the timed robustness case w.r.t enlargement, boundedness of nets is again the key issue. When a net is bounded, then one can compute an equivalent timed automaton, and reuse all robustness results for TA to answer Petri nets robustness questions (markings or language preservation, ...). The question of robustness for unbounded nets is still open. While boundedness of a timed Petri net is in general undecidable, one can expect to find decidable subclasses of unbounded nets for which robustness issues are decidable. Of course, this should call for completely new techniques, as translation to timed automata works only in a bounded context.

References

1. S. Akshay, Loïc Hélouët, Claude Jard, Didier Lime, and Olivier H. Roux. Robustness of time petri nets under architectural constraints. In *FORMATS*, volume 7595 of *Lecture Notes in Computer Science*, pages 11–26. Springer, 2012.
2. S. Akshay, Loïc Hélouët, Claude Jard, and Pierre-Alain Reynier. Robustness of time petri nets under guard enlargement. In *RP*, volume 7550 of *Lecture Notes in Computer Science*, pages 92–106. Springer, 2012.
3. Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
4. Béatrice Bérard, Franck Cassez, Serge Haddad, Didier Lime, and Olivier H. Roux. Comparison of the expressiveness of timed automata and time Petri nets. In Paul Pettersson and Wang Yi, editors, *3rd International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS 2005)*, volume 3829 of *Lecture Notes in Computer Science*, pages 211–225, Uppsala, Sweden, September 2005. Springer-Verlag.
5. Beatrice Berard, Antoine Petit, Volker Diekert, and Paul Gastin. Characterization of the expressive power of silent transitions in timed automata. *Fundam. Inform.*, 36(2–3):145–182, 1998.
6. Bernard Berthomieu and Michel Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE transactions on software engineering*, 17(3):259–273, March 1991.
7. Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust model-checking of linear-time properties in timed automata. In *Proc. of LATIN'06*, volume 3887 of *LNCS*, pages 238–249, 2006.
8. Patricia Bouyer, Nicolas Markey, and Pierre-Alain Reynier. Robust analysis of timed automata *via* channel machines. In *Proc. of FoSSaCS'08*, volume 4962 of *LNCS*, pages 157–171. Springer, 2008.
9. Patricia Bouyer, Nicolas Markey, and Ocan Sankur. Robust model-checking of timed automata via pumping in channel machines. In *Proc. of FORMATS'11*, volume 6919 of *LNCS*, pages 97–112, 2011.
10. Franck Cassez and Olivier H. Roux. Structural translation from time petri nets to timed automata. *Journal of Systems and Software*, 79(10):1456–1468, 2006.
11. A. Cheng, J. Esparza, and J. Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1-2):117–136, 1995.
12. Davide D'Aprile, Susanna Donatelli, Arnaud Sangnier, and Jeremy Sproston. From time Petri nets to timed automata:An untimed approach. In *TACAS'07*, volume 4424 of *LNCS*, pages 216–230, 2007.
13. Martin De Wulf, Laurent Doyen, Nicolas Markey, and Jean-François Raskin. Robust safety of timed automata. *Formal Methods in System Design*, 33(1-3):45–84, 2008.

14. Martin De Wulf, Laurent Doyen, and Jean-François Raskin. Systematic implementation of real-time models. In *Formal Methods (FM'05)*, volume 3582 of *LNCS*, pages 139–156. Springer, 2005.
15. Guillaume Gardey, Olivier (F.) Roux, and Olivier (H.) Roux. Safety control synthesis for time Petri nets. In *8th International Workshop on Discrete Event Systems (WODES'06)*, pages 222–228, Ann Arbor, USA, July 2006. IEEE Computer Society Press.
16. Guillaume Gardey, Olivier H. Roux, and Olivier F. Roux. A zone-based method for computing the state space of a time Petri net. In *Proc. of FORMATS'03*, volume 2791 of *LNCS*, pages 246–259, 2003.
17. A. Giua, F. DiCesare, and M. Silva. Petri net supervisors for generalized mutual exclusion constraints. In *Proc. 12th IFAC World Congress*, pages 267–270, Sidney, Australia, jul 1993.
18. L. E. Holloway and B. H. Krogh. Synthesis of feedback control logic for a class of controlled Petri nets. *IEEE Trans. on Automatic Control*, 35(5):514–523, may 1990.
19. R.M. Karp and R.E. Miller. Parallel program chemata. *In JCSS*, 3:147–195, 1969.
20. Didier Lime, Claude Martinez, and Olivier H. Roux. Shrinking of time Petri nets. *Journal of Discrete Event Dynamic Systems (jDEDS)*, 2013.
21. Didier Lime and Olivier (H.) Roux. Model checking of time Petri nets using the state class timed automaton. *Journal of Discrete Events Dynamic Systems - Theory and Applications (DEDS)*, 16(2):179–205, 2006.
22. Oded Maler, Amir Pnueli, and Joseph Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *Proc. STACS '95*, number 900 in LNCS, pages 229–242. Springer–Verlag, 1995.
23. Philip M. Merlin. *A Study of the Recoverability of Computing Systems*. PhD thesis, University of California, Irvine, CA, USA, 1974.
24. Anuj Puri. Dynamical properties of timed automata. *In DEDS*, 10(1-2):87–113, 2000.
25. Pierre-Alain Reynier and Arnaud Sangnier. Weak time petri nets strike back! In *CONCUR*, volume 5710 of *Lecture Notes in Computer Science*, pages 557–571, 2009.
26. Ocan Sankur. Untimed language preservation in timed systems. In *Proc. of MFCS'11*, LNCS. Springer, 2011.
27. Mani Swaminathan, Martin Fränzle, and Joost-Pieter Katoen. The surprising robustness of (closed) timed automata against clock-drift. In *TCS 2008*, pages 537–553. Springer, 2008.
28. M. Uzam, A.H. Jones, and I. Yucel. Using a Petri-net-based approach for the real-time supervisory control of an experimental manufacturing system. *Journal of Electrical Engineering and Computer Sciences*, 10(1):85–110, 2002.
29. V. Valero, D. Frutos-Escrig, and F. Cuartero. On non-decidability of reachability for timed-arc Petri nets. In *Proc. 8th International Workshop on Petri Nets and Performance Models (PNPM 99)*, 1999.
30. Dianxiang Xu, Xudong He He, and Yi Deng. Compositional schedulability analysis of real-time systems using time Petri nets. *IEEE Transactions on Software Engineering*, 28(10):984 – 996, 2002.