

Unfolding of time Petri nets for quantitative time analysis

Medesu Sogbohossou and David Delfieu

Institut de Recherche en Communications et Cybernétique de Nantes,
1, rue de la Noë - BP 92 101 - 44321 Nantes FRANCE
{medesu.sogbohossou,david.delfieu}@irccyn.ec-nantes.fr
<http://www.irccyn.ec-nantes.fr>

Abstract. The verification of properties on a Time Petri net is often based on the state class graph. For a highly concurrent system, the construction of a state graph often faces to the problem of combinatory explosion. This problem is due to the systematic interleaving of concurrent transitions. This paper proposes a new method of unfolding limiting as much as possible interleavings. This unfolding keeps a partial order execution and interleaving are hold on when their allow to put all conflicts in a total order. The method gives a way to generate a complete finite prefix unfolding based on the construction of the exhaustive set of scenarios on a safe time Petri net. Moreover, the method allows various quantitative time analyses based on the complete finite prefix unfolding and on a permanent time database.

Key words: Time Petri net, unfolding, partial order, conflicts, quantitative time analyse

1 Introduction

Petri nets are interesting formalism with expressing power of true parallelism or concurrency in discrete events systems. Time Petri nets (TPN) [1] are timed extension often used to modelize realtime systems. Thanks to a static interval constraining occurrence time of each transition, the model permits to modelize applications with watchdogs or time switches. The verification of properties is often based on the state class graph [2–4] of the TPN. For a highly concurrent system, the construction of a state graph often faces to the problem of combinatory explosion, because of the systematic interleaving of the concurrent transitions. An alternative could be to adopt a partial order semantics.

The technique of the unfolding [7–9] exhibits behaviors as an acyclic net respecting the partial order. Some works extend the unfolding technique to TPN. In [10], the technique is defined for the restrictive class of TPNs called *time independent-choice TPNs*, in which any conflict is resolved independently of time. In [11], a generalization is made for safe TPNs: a firing in a conflict causes as much events as combinations of some concurrent places in the unfolding, each combination defining a partial state. A partial state shows how to obtain the time constraints on the conflicting firing, and its concurrent places are linked to the corresponding event by read arcs. This principle increases quickly the number of events in the unfolding in case of a great number of such partial states.

Another method of unfolding was proposed in the context of the reduction of a finite TPN module [12]. A time domain is assessed for each output event of the module. This work is restricted to nets in which the temporal assessment on an event in conflict never depends on the occurrence of a concurrent event. The main objective of this paper is to propose a new method of unfolding with some improvements by comparison with methods in [10–12]. It is based on the notion of *scenario* presented in [13] and corresponding to a partial order execution such that all conflicts are put in a total order. The new method gives a way to generate a complete finite prefix unfolding based on the construction of the scenarios of a safe TPN. The applications could be the checking of the validness of a firing schedule, the estimation of the delay between two events or states of the system, or the computing of the timing profile about any process of a TPN.

The next section presents basic definitions on (time) Petri nets and on unfolding of untimed Petri nets. The section 3 presents a way to carry out a temporal assessment on the events of a non sequential run of TPN, not

based on the classical interleaving semantics. Section 4 shows that the interleavings of the concurrent events in conflict in the process is necessary to temporally characterize a process, and the resulting process of one such interleaving is defined as a *scenario*. This section also shows how to compute the scenarios to get a TPN unfolding. Section 5 deals with the question of finiteness of a TPN unfolding capturing all quantitative time informations. Section 6 exposes the applications of a complete and finite TPN unfolding for making various quantitative time analyses.

2 Unfolding

2.1 Petri net

A Petri net $\mathcal{N} = \langle \mathcal{P}, \mathcal{T}, \mathcal{W} \rangle$ is a triple with: \mathcal{P} , the finite set of places, \mathcal{T} , the finite set of transitions ($\mathcal{P} \cup \mathcal{T}$ are nodes of the net; $\mathcal{P} \cap \mathcal{T} = \emptyset$), and $\mathcal{W} : \mathcal{P} \times \mathcal{T} \cup \mathcal{T} \times \mathcal{P} \rightarrow \mathbb{N}$, the flow relation defining arcs (and their valuations) between nodes of \mathcal{N} . In the sequel, we will consider only *1-valued* Petri net.

The pre-set (resp. post-set) of a node x is denoted $\bullet x = \{y \in \mathcal{P} \cup \mathcal{T} \mid \mathcal{W}(y, x) > 0\}$ (resp. $x^\bullet = \{y \in \mathcal{P} \cup \mathcal{T} \mid \mathcal{W}(x, y) > 0\}$).

A marking of a Petri net \mathcal{N} is the mapping $m : \mathcal{P} \rightarrow \mathbb{N}$. A transition $t \in \mathcal{T}$ is said *enabled* by m iff: $\forall p \in \bullet t, m(p) \geq \mathcal{W}(p, t)$. This is denoted: $m \xrightarrow{t}$. Firing of t leads to the new marking m' ($m \xrightarrow{t} m'$): $\forall p \in \mathcal{P}, m'(p) = m(p) - \mathcal{W}(p, t) + \mathcal{W}(t, p)$. The initial marking is denoted m_0 .

A Petri net is *k-bounded* iff $\forall m, m(p) \leq k$ (with $p \in \mathcal{P}$). It is said *safe* when *1-bounded*. This paper is restricted to safe nets.

Two transitions are in a *structural conflict* when they share at least one pre-set place; a conflict is *effective* when these transitions are enabled by a same marking.

2.2 Branching process

In [9], the notion of *branching process* is defined as an initial part of a run of a Petri net respecting its partial order semantics and possibly including non deterministic choices (conflicts). It is another Petri net that is acyclic and the largest branching process of an initially marked Petri net is called *the* unfolding of this net. Resulting net from an unfolding is a labeled occurrence net, a Petri net whose places are called *conditions* (labeled with their corresponding place name in the original net) and transitions are called *events* (labeled with their corresponding transition name in the original net).

An occurrence net $\mathcal{O} = \langle \mathcal{B}, \mathcal{E}, \mathcal{F} \rangle$ is a 1-valued arcs Petri net, with \mathcal{B} the set of conditions, \mathcal{E} the set of events, and \mathcal{F} the flow relation (1-valued arcs), such that: $|\bullet b| \leq 1$ ($\forall b \in \mathcal{B}$), $\bullet e \neq \emptyset$ ($\forall e \in \mathcal{E}$), and \mathcal{F}^+ (the transitive closure of \mathcal{F}) is a strict order relation, from which \mathcal{O} is acyclic. $Min(\mathcal{O}) = \{b \mid b \in \mathcal{B}, |\bullet b| = 0\}$ is the minimal conditions and corresponds to the initial marking. Also, $Max(\mathcal{O}) = \{x \mid x \in \mathcal{B} \cup \mathcal{E}, |x^\bullet| = 0\}$ are maximal nodes.

Three kinds of relations could be defined between the nodes of an occurrence net. First, two nodes $x, y \in \mathcal{B} \cup \mathcal{E}$ are in *strict causal relation* if $(x, y) \in \mathcal{F}^+$, and will be denoted $x \prec y$. $\forall b \in \mathcal{B}$, if $e_1, e_2 \in b^\bullet \subseteq \mathcal{E}$ ($e_1 \neq e_2$), then e_1 and e_2 are in *conflict relation*, denoted $e_1 \# e_2$. More generally, for $x, y \in \mathcal{B} \cup \mathcal{E}$, if $e_1 \prec x$ et $e_2 \prec y$, then $x \# y$, $x \# e_2$ and $e_1 \# y$. Symbol \wr represents *concurrency relation*: $x \wr y$ iff $\neg((x \prec y) \vee (y \prec x) \vee (x \# y))$.

A *cut* is a set of conditions all in mutually concurrency relation. A *B-cut* is a maximal cut according to inclusion and represents a marking of the original net.

The *local configuration* of an event e , denoted $[e]$, is the set of events e' such that $e' \preceq e$ (relation \preceq is defined on \mathcal{F}^* where \mathcal{F}^* is the transitive and reflexive closure of \mathcal{F}).

Let be a net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{W})$. $\langle \mathcal{N}, m_0 \rangle$ admits $Unf = \langle \mathcal{O}, \lambda \rangle$ as branching process (or unfolding) if:

- $\mathcal{O} = \langle \mathcal{B}, \mathcal{E}, \mathcal{F} \rangle$ is an occurrence net;
- λ is the labeling function such that $\lambda : \mathcal{B} \cup \mathcal{E} \rightarrow \mathcal{P} \cup \mathcal{T}$. It verifies the following properties:
 - $\lambda(\mathcal{B}) \subseteq \mathcal{P}$, $\lambda(\mathcal{E}) \subseteq \mathcal{T}$;

- $\lambda(\text{Min}(\mathcal{O})) = m_0$;
- for all $e \in \mathcal{E}$, restriction of λ to $\bullet e$ (resp. $e \bullet$) is a bijection between $\bullet e$ (resp. $e \bullet$) and $\bullet \lambda(e)$ (resp. $\lambda(e) \bullet$).
We have $\bullet \lambda(e) = \lambda(\bullet e)$ and $\lambda(e) \bullet = \lambda(e \bullet)$, what means that λ preserves the environment of a transition.

The resulting labeled occurrence net is defined up to isomorphism, what means that we obtain a same structure of net up to the name of nodes, and with same labeling of nodes.

A *causal net* \mathcal{C} is an occurrence net $\langle \mathcal{C} = \mathcal{B}, \mathcal{E}, \mathcal{F} \rangle$ with an adding restriction: $\forall b \in \mathcal{B}, |b \bullet| \leq 1$. So, all conflicts are resolved in a causal net, and this net corresponds to a partial order run in an unfolding. The set of events of a causal net is called a *process*. Notice that only the order relations \prec and \wr are admitted between events of a process. Let be $\mathcal{C}_i = \langle \mathcal{B}_i, \mathcal{E}_i, \mathcal{F}_i \rangle$, a causal net of an unfolding $\text{Unf} = \langle \mathcal{O}, \lambda \rangle$ with $\mathcal{O} = \langle \mathcal{B}, \mathcal{E}, \mathcal{F} \rangle$: $\mathcal{E}_i \subseteq \mathcal{E}$ and $\mathcal{B}_i \subseteq \mathcal{B}$. The set of neighboring events in conflict with $e \in \mathcal{E}_i$ is denoted $\text{Conf}(e)$: $\text{Conf}(e) = \{x \in \mathcal{E} \setminus \mathcal{E}_i \mid \bullet x \cup \bullet e \subseteq \mathcal{B}_i \wedge \bullet x \cap \bullet e \neq \emptyset\}$. In the next sections, the set $\text{Conf}(e)$ defines the *conflict constraints* of e in a process of TPN. Usually, the net \mathcal{N} is free of a sink transition ($t \in \mathcal{T}, t \bullet \neq \emptyset$), what prevents to have an event in $\text{Max}(\mathcal{C}_i)$. So, in the next sections, the set of the conditions of $\text{Max}(\mathcal{C}_i)$ (that is a B-cut) is called the *state* of the causal net \mathcal{C}_i .

2.3 Time Petri net

A time Petri net (TPN)[14] is a net where each transition carries an interval allowing the modeling of an uncertain delay.

Definition 2.1. A TPN $\mathcal{N} = \langle \mathcal{N}', \text{efd}, \text{lfd} \rangle$ is a Petri net \mathcal{N}' with:

$$\begin{aligned} \text{efd} : \mathcal{T} &\longrightarrow \mathbb{Q}^+ && (\text{earliest firing delay}) \\ \text{lfd} : \mathcal{T} &\longrightarrow \mathbb{Q}^+ \cup \{\infty\} && (\text{latest firing delay}) \end{aligned}$$

The time interval $[\text{efd}(t), \text{lfd}(t)]$ is attached to each transition t . The *enabling date* of a transition is the occurrence date of the last firing that allows its enabling. It can be considered that a clock is started at the enabling date of each transition t , and the time elapses densely. This transition cannot be fired before the delay $\text{efd}(t)$ is elapsed, and must be inevitably fired after the delay $\text{lfd}(t)$ ($\text{efd}(t) \leq \text{lfd}(t)$), except if the firing of another transition in an effective conflict with t disables it. So, a time Petri net can express the concept of urgency typical to realtime systems. An illustration is given in figure 1(a). Suppose that t_1 and t_2 are enabled at the date 0 by the initial marking. t_1 could not be fired after the date 3 because t_2 is fired at the latest at this date. Thus, the effective time domain of a firing could be modified by a conflict (here $[2, 3]$ instead of $[2, 7]$ for t_1).

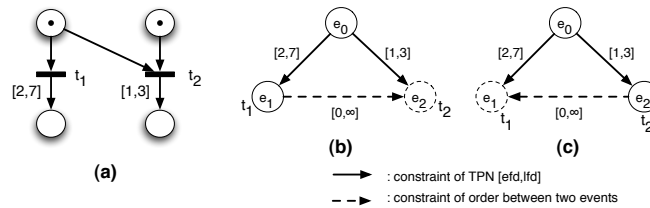


Fig. 1. Illustration of the strong semantics

The usual semantics of TPN (used in this paper) is initially defined in [14] and is called *strong semantics*. Another semantics is called *weak semantics* where there is no obligation to fire a transition before the upper bound lfd of its firing interval [15]. Only the strong semantics allows to model a watchdog in real-time systems.

For the representation of all reachable markings in a state graph, the interleaving semantics is expressed in terms of firing sequences, and causes the state explosion problem. Let be $\sigma = t_1 \cdot t_2 \cdot \dots \cdot t_i \cdot \dots \cdot t_n$, a

sequence of fireable transitions from the initial marking m_0 ($m_0 \xrightarrow{\sigma}$), and $\underline{\theta} = (\theta_1, \theta_2, \dots, \theta_i, \dots, \theta_n)$ the tuple of the corresponding firing dates. Usually, the firing date θ_i of t_i is relative to the one of the subsequent firing t_{i-1} (θ_1 is relative to the date $\theta_0 = 0$). A *firing schedule* is the pair $(\sigma, \underline{\theta})$. A firing schedule $(\sigma, \underline{\theta})$ is *feasible* iff it exists a sequence σ such as the transitions in the sequence are successively fireable at the corresponding relative dates in $\underline{\theta}$.

The partial order semantics (used for the branching processes) is more suitable for expressing the concurrency in a behavior (or run) of a Petri net and avoids the systematic interleavings causing the state explosion problem.

3 Temporal assessment on events

A causal net (or process) expresses a behavior of TPN in the partial order semantics. In the interleaving semantics, a date is relative to the previous firing, what is unsuitable for independent firings (i.e. concurrent events) in the partial order semantics. To keep the temporal independence between the concurrent events of the process, a simple way consists in choosing a global time reference like the instant of production of the initial state. This date matches to a fictitious event e_0 and is supposed equal to 0. Thus, an absolute firing date is associated to each event of the process.

Let e_x be an event with $\lambda(e_x) = t$. According to the weak semantics, its absolute occurrence date denoted D_x is given by:

$$D_x = d(t) + \max(\{D_y \mid e_y \in \bullet(\bullet e_x) \cup \{e_0\}\}) \quad (1)$$

with $d(t) \in [efd(t), lfd(t)]$. Any event $e_y \in \bullet(\bullet e_x) \cup \{e_0\}$ will be called an enabling event of e_x . The absolute time domain of e_x is the interval $[D_{x_{min}}, D_{x_{max}}]$ such that $D_{x_{min}} = efd(t) + \max(\{D_{k_{min}} \mid e_k \in \bullet(\bullet e_x)\})$ and $D_{x_{max}} = lfd(t) + \max(\{D_{k_{max}} \mid e_k \in \bullet(\bullet e_x)\})$.

In a general way, the expression 1 remains valid and sufficient in the strong semantics if e_x is not subjected to any conflict, the evaluations about enabling events being supposed already known.

In case of conflict, the time domain of an event could become more restrictive because of the strong semantics (as shows the example figure 1), or even, the event is impossible with no valid time domain. For instance, the realtime programs could contain some execution loop where finiteness depends on the time bound of some repetitive action. In term of unfolding of a TPN, the validness of each underlying event¹ in conflict must be checked. So, a method for the quantitative temporal assessment (such as one used to obtain the canonical form of a firing domain of a state class [21], based on the shortest path algorithm also called Floyd-Warshall algorithm [16,17]) is needed to know if a valid time domain exists for each event under a conflict.

The time domain of an event e_x depends on two kinds of temporal constraints:

- the *own constraints* of e_x due to the delay specification (efd/lfd) of the transition and the time domains of the enabling events;
- the *vicinity constraints* of e_x due to the time domains of the events liable to preempt e_x , that is the events in $Conf(e_x)$. It is assumed that all the conflict constraints of an event are known before to assess its time domain. This is reached by sufficiently unfold the net.

About the vicinity constraints of e_x , notice that $Conf(e_x)$ does not indicate the events that will be really enabled at the moment of the occurrence of e_x , but gives the events can *potentially* preempt e_x .

A process of TPN can be translated into a *simple temporal problem* (STP) [17] that helps the assessment of time difference between events. A temporal constraint between two events corresponds to a convex interval. The network of the STP can be represented by a *directed constraint graph*, where nodes represent variables of events, and an edge between two nodes indicates the interval of the constraint. The absolute date variable D_x of an event e_x is relative to 0, the date of the reference event e_0 .

Let be $a_{ij} \leq D_j - D_i \leq b_{ij}$, the constraint between two events e_i and e_j . The edge from e_i to e_j (notice that only one edge could be defined between two nodes) is labeled by the interval $[a_{ij}, b_{ij}]$ in the constraint graph.

¹ an *underlying event* is an event of the unfolding of the underlying net.

The constraint graph is translated into a *distance graph* that gets the same nodes as the constraint graph, and an edge $e_i \rightarrow e_j$ labeled by $[a_{ij}, b_{ij}]$ is split into two edges: an edge $e_i \rightarrow e_j$ labeled by b_{ij} , and an edge $e_j \rightarrow e_i$ labeled by $-a_{ij}$. The application of the shortest path algorithm determines the intersection of the intervals of the possible paths between all pair of events, and tells if the problem is *consistent*.

For the temporal assessment of an event e_i to choose in a conflict, since e_i preempts the events in $Conf(e_i)$, an edge $e_i \rightarrow e_j$ ($e_j \in Conf(e_i)$) labeled by the interval $[0, \infty]$ will express the order constraint. So, the directed constraint graph of the net figure 1(a) is given figure (b) if e_1 is produced, or figure (c) if e_2 is produced. The application of the shortest path algorithm on the resulting distance graph gives $D_1 \in [2, 3]$ for (b) and $D_2 \in [1, 3]$ for (c).

For a process of TPN, there exists at least as much paths reaching an event as causes (the enabling events). But, in a constraint graph, multiple causes for an event (expressing an intersection) does not have the same meaning as for TPNs (see the equation 1). Using the method in [17] requires only one enabling event; then, in case of multiple causes, only one effective cause is considered (the last enabling event produced), what means that the other causes are previously produced: a constraint $[0, \infty]$ is defined from each other cause toward the effective cause. All combinations must be considered.

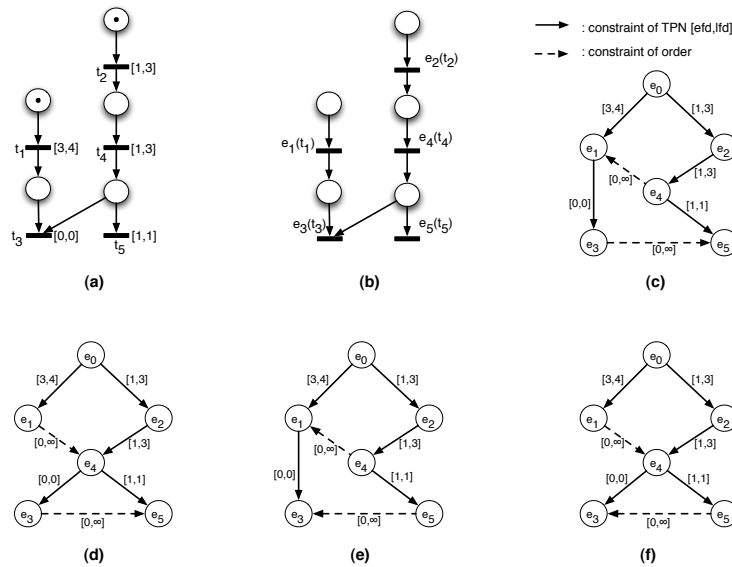


Fig. 2. Example of unfolding (b) of a net (a) and the temporal assessment (c,d) on e_5 (e,f)

An example is given figure 2: the unfolding of a net (a) is given at (b). The two events e_5 and e_3 are in conflict and e_5 has two enabling events e_1 and e_4 . When e_3 is the preempting event in the process $\{e_1, e_2, e_4, e_3\}$, the temporal assessment gives $D_3 \in [3, 4]$ for the first interleaving $e_4; e_1$ (c), and $D_3 \in [3, 6]$ for the second $e_1; e_4$ (d). More generally, the (convex) time domain of an event is the union of the valid domains obtained for the different combinations. So, $D_3 \in [3, 6]$. When e_5 is the preempting event in the process $\{e_1, e_2, e_4, e_5\}$, the temporal assessment gives $D_5 \in [3, 4]$ for the first interleaving $e_4; e_1$ (c), the second interleaving $e_1; e_4$ (d) is inconsistent and so $D_5 \in [3, 4]$. Notice that the conflict-free evaluations with the formulae 1 gives $D_3 \in [3, 6]$ and $D_5 \in [3, 7]$; the example shows that it is the event with the smallest D_{max} (here e_3) that constrains more the other events in the conflict (here e_5) by the means of the order constraint (the arc labeled $[0, \infty]$). In all, three valid combinations is used here whereas the systematic interleaving semantics (used in the state class graph

where the concurrent events, e_1 with e_2 and e_1 with e_4 , are interleaved) will give five valid combinations. The method saves some interleavings, even more since there is a great number of concurrent events.

If a non sequential process is composed of n events, and each event e_i gets n_i causes, the number of combinations could be exponential ($\prod_{i=1}^n n_i$). Each combination is computed in a polynomial time by the application of the shortest path algorithm.

It is assumed that the time diverges during a run of a TPN. This implies that any concurrent but temporal predecessor of an event e_x ends up occurring. Thus, the valid assessment on an (effective) event e_x in a process also implies that it will necessarily occur.

4 Computing of TPN processes

4.1 Need of conflicts interleaving

In [18] was presented a method allowing to analyze in the weak semantics a behavior of a TPN defined as a *scenario* (a multiset of transitions). Unlike the unfolding technique that can compute all the alternative behaviors, only one run is considered in the form of a predetermined scenario. Thanks to a temporal labeling of propositions during the proof of a scenario translated in Linear Logic [19], symbolic expressions on firing dates are derived for the analysis. An extension for the strong semantics is made in [13]: it is based on the observation that the symbolic date of a firing in a structural conflict can depend on the order of the resolution of the conflicts. In the next subsection, this paper will formalize the concept of scenario in the context of the unfolding of a TPN.

As shows the net figure 3(a), the concurrent firings t_1 and t_3 form a process; if t_1 is fired before t_3 (figure 3(c)), the time domains are respectively $[1, 4]$ and $[1, 5]$, but if t_3 is fired before t_1 (figure 3(d)), the domains are respectively $[0, 4]$ and $[1, 6]$. The reason of the difference is that the conflict constraints set depend on the occurrence of a concurrent event in conflict. In the first case, $Conf(e_1) = \{e_2\}$ and $Conf(e_3) = \{\}$; in the second case, $Conf(e_1) = \{\}$ and $Conf(e_3) = \{e_2\}$.

Let be e , the event in conflict whose time domain has to be assessed, and the set $B_{conf}(e) = \bullet Conf(e) \setminus (\bullet e \cap \bullet Conf(e))$ containing the preconditions of the events in $Conf(e)$ that are concurrent to e . $B_{conf}(e)$ decides the effective constraints $Conf(e)$ applied on e .

Proposition 4.1. *The existence of a condition in the set $B_{conf}(e)$ could be called into question by the occurrence of a concurrent event to e subject to conflict and done before e .*

For instance, in the first case of the previous example, the occurrence of e_1 before e_3 makes unavailable the condition b_2 in the set $B_{conf}(e_3)$, so e_2 can not constrain e_3 .

An easy way to know the concurrent events that decide $B_{conf}(e)$ is to interleave all the solvings of conflicts in the process. Thus, according to the position of e in an interleaving, all the concurrent events in conflict (called *concurrent choices*) occurred before e are known, and so, $Conf(e)$ could be determined as a single set.

To summarize, the relations (\prec and \wr) between the events in a process is not sufficient for the temporal characterization in a single way; there is needed to be enriched by a new relation, a total order relation between the concurrent choices. In a constraint graph, this is translated by an edge between two following concurrent choices labeled by the interval $[0, \infty]$ (see the figures 3 (c) and (d) for the given examples).

Generally, during the process of the unfolding of a net, the concurrent nodes can be known only in a certain order. Thus, the knowledge of the neighboring events in conflict is not certain when producing an event. So, an event in a conflict is not always detectable at the moment it is produced. For the sake of simplicity, any event in a structural conflict of the TPN will be subjected to the interleavings.

4.2 Scenario

Let be $\mathcal{N} = \langle \mathcal{N}', efd, lfd, m_0 \rangle$, a marked TPN where $\mathcal{N}' = \langle \mathcal{P}, \mathcal{T}, \mathcal{W} \rangle$. $Unf_F = \langle \mathcal{O}_F, \lambda_F \rangle$ is the *the* unfolding of \mathcal{N}' with $\mathcal{O}_F = \langle \mathcal{B}_F, \mathcal{E}_F, \mathcal{F}_F \rangle$. $\langle \mathcal{C}_i = \mathcal{B}_i, \mathcal{E}_i, \mathcal{F}_i \rangle$ is a prefix of causal net in \mathcal{O}_F ($\mathcal{B}_i \subset \mathcal{B}_F$ and $\mathcal{E}_i \subset \mathcal{E}_F$).

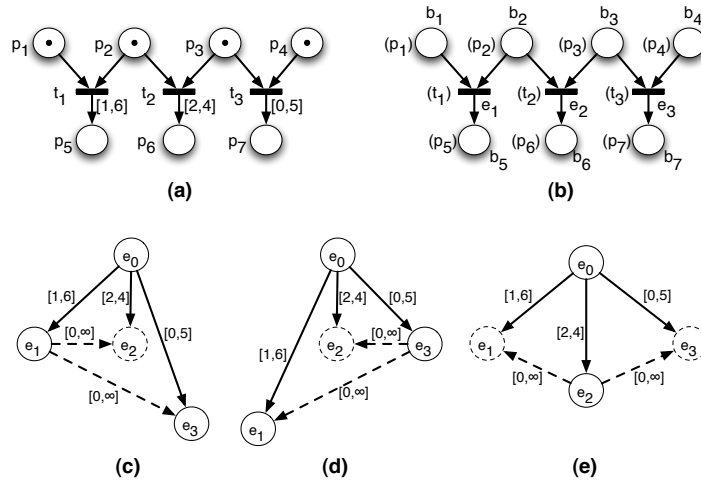


Fig. 3. Necessity of interleavings in case of conflict

A transition t of \mathcal{N} in a structural conflict iff: $|(\bullet t) \bullet| > 1$. $Sconf(\mathcal{E}_i)$ is the set of events in a structural conflict: $Sconf(\mathcal{E}_i) = \{e \in \mathcal{E}_i \mid \lambda_F(e) = t, t \in \mathcal{T} \wedge (\bullet t) \bullet > 1\}$. Since only events in $Sconf(\mathcal{E}_i)$ are interleaved, a *strict order relation* $\rightsquigarrow \subseteq Sconf(\mathcal{E}_i) \times Sconf(\mathcal{E}_i)$ is defined. Two events in conflict in Unf_F are not necessarily in a structural conflict. Such a conflict not structural is due to the non safeness of the underlying net. To avoid this problem, the underlying net must also be safe. To ensure safeness when unfolding, it is sufficient to check every time when a new post-condition is created that there is no previous concurrent condition with the same label in \mathcal{P} .

In a process of a TPN, an event will be denoted x , and is an extension of a corresponding underlying event e if it exists a feasible firing schedule containing this event.

Let be \mathcal{X} , a set of events of a TPN process. \mathcal{C}_i is the related causal net if the relation between \mathcal{X} and \mathcal{E}_i is a bijection $\mu : \mathcal{X} \mapsto \mathcal{E}_i$.

The causal net \mathcal{C}_i with many concurrent choices can allow many interleavings of these choices. A process of the TPN with an imposed sequence of the choices will be called a scenario. A *scenario* \mathcal{S} of \mathcal{C}_i is the set \mathcal{X} respecting a defined strict order relation \rightsquigarrow : $\mathcal{S} = \langle \mathcal{X}, \rightsquigarrow \rangle$.

A scenario is characterized by the set(s) of binary temporal constraints issued from the STP(s) of the process on its events. In the sequel, the temporal information attached to an event e to form an event x of a scenario is not specified. It could be all the binary temporal relations between x and the events that determine or constraint its occurrence, or just the absolute time domain of x .

4.3 Computation of scenarios

All the causal nets can be generated from the initial state $Min(\mathcal{O}_F)$ by producing the enabled concurrent events one by one. The first ancestor of all causal nets is the net \mathcal{C}_0 only constituted by the initial conditions $Min(\mathcal{O}_F)$. Later, the fictitious event e_0 will be considered (by abuse) belonging to \mathcal{C}_0 : $\mathcal{E}_0 = \{e_0\}$.

The *possible extensions* of a causal net \mathcal{C}_i , denoted $PE(\mathcal{E}_i)$, are the set of new events that can be derived from its maximal concurrent conditions $Max(\mathcal{C}_i)$: $PE(\mathcal{E}_i) = \{e \in \mathcal{E}_F \mid \bullet e \subset Max(\mathcal{C}_i)\}$.

To extend \mathcal{C}_i , an event $e_{i+1} \in PE(\mathcal{E}_i)$ is produced and added to \mathcal{E}_i , and it is assumed that the condition(s) e_{i+1}^\bullet are produced immediately after and added to \mathcal{B}_i . $\mathcal{C}_{i+1} = \langle \mathcal{B}_{i+1}, \mathcal{E}_{i+1}, \mathcal{F}_{i+1} \rangle$, with $\mathcal{E}_{i+1} = \mathcal{E}_i \cup \{e_{i+1}\}$ and $\mathcal{B}_{i+1} = \mathcal{B}_i \cup e_{i+1}^\bullet$, is a *derivative*² net of the *prefix* net \mathcal{C}_i .

² the word *derivative* has the sense of the extension of another causal net (or scenario) containing the latter.

The objective is to compute all the scenarios of a time Petri net by ensuring a valid temporal assessment on each produced event in a scenario before constructing its causal successors. The method is based on the rule expressed by the following proposition:

Proposition 4.2. *To extend a causal net, the priority is given to the production of the events (and their post-conditions) free of structural conflict.*

Definition 4.2. *A causal net \mathcal{C}_i is said in a choice state iff: $\forall e \in PE(\mathcal{E}_i), e \in Sconf(\mathcal{E}_F)$.*

Let be $\mathcal{S}_j = \langle \mathcal{X}_j, \rightsquigarrow_j \rangle$ a scenario of \mathcal{C}_i in a choice state, with: $\mu_j : \mathcal{X}_j \mapsto \mathcal{E}_i$. To each event $e_{i+1} \in PE(\mathcal{E}_i)$ corresponds one derivative net \mathcal{C}_{i+1} with the *derivative scenario* $\mathcal{S}_{j+1} = \langle \mathcal{X}_{j+1}, \rightsquigarrow_{j+1} \rangle$ such as: $\mathcal{X}_{j+1} = \mathcal{X}_j \cup \{x_{j+1}\}$, $\mu_{j+1} : \mathcal{X}_{j+1} \mapsto \mathcal{E}_{j+1}$ ($\forall x \in \mathcal{X}_{j+1}$, if $x = x_{j+1}$ then $\mu_{j+1}(x) = e_{i+1}$, else $\mu_{j+1}(x) = \mu_j(x)$), and $\rightsquigarrow_{j+1} = \rightsquigarrow_j \cup \{(e, e_{i+1}) \mid e \in Sconf(\mathcal{E}_i)\}$.

Assumption 41 *Any run of the TPN is made of a finite number of events.*

This assumption will be relaxed later to take into account a class of systems with infinite runs, without questioning the theorems in the sequel.

From a net \mathcal{C}_i , some event $e_{i+1} \in PE(\mathcal{E}_i)$ without structural conflict is produced, and every extension e_{i+1} matches to a derivative net \mathcal{C}_{i+1} . Since any run is finite, a process of successive extensions from \mathcal{C}_0 will end up by a derivative net \mathcal{C}_n such as no more event without structural conflict is producible. If $PE(\mathcal{E}_n)$ is not empty, then \mathcal{C}_n is in the *first* choice state.

There is only one derivative net \mathcal{C}_n in a first choice state. Indeed, since no choice is made in \mathcal{C}_n , there is no alternative to this execution. The events of $PE(\mathcal{E}_n)$ are the first possible choices in the possible runs of the underlying net \mathcal{N}' . Notice that all the conflict-free events \mathcal{E}_n will necessary occur in the timed extension \mathcal{N} , even if the time specification could forbid some orderings between concurrent events.

Theorem 4.1. *The proposition 4.2 is adequate for identifying all the interleavings of choices in the unfolding Unf_F of \mathcal{N}' .*

Proof. By induction:

- From the first choice state, all first possible choices are known, and making one of the choices leads to a distinct scenario.
- Let be the m^{th} choice state (that means $(m-1)$ choices have been made previously in the unfolded scenario and causal net to reach this state). It is assumed that all the possible choices from this state are known, and each choice will be made in a corresponding derivative scenario.
- Let be e_n , the choice made in the m^{th} state. After production of e_n , all the next choices having the causal predecessor e_n are found (by applying the rule of the proposition 4.2), and let be the $(m+1)^{th}$ choice state reached when a conflict-free event is no more producible. Any other next possible choice (that is not derived from e_n) in this state is a concurrent event of e_n enabled but not produced from the m^{th} choice state. Likewise, each choice will be made in a corresponding derivative scenario.

Thus, all possible choices are known from any choice state, and in a next choice state, all the possible choices that are feasible after a previous choice are also known. Then, all the interleavings of choices are known when the corresponding scenarios are completely unfolded.

According to the assumption 41, any choice state is reached in a finite time. Since an interleaving of choices in a causal net corresponds to a scenario of the TPN, the method allows to extract all possible scenarios of the TPN, provided that each choice is submitted to an assessment to ensure that it is not prevented by another event in conflict.

4.4 Method for adequate temporal assessment

A temporal assessment is necessary to know if an event is possible in some feasible firing schedule or in some TPN process. The temporal assessment of a conflict-free event only depends on its own constraints (see the subsection 3). For an event in conflict, the vicinity constraints must be enriched with the informations about the total ordering of the choices, that may impose adding temporal restrictions on the event. An interleaving will be impossible if an event could never respect its temporal order (according to the relation \rightsquigarrow). With the method presented in this work, the objective is to guarantee that for each event produced in a scenario, a valid temporal assessment is done before producing its successors. This supposes that all temporal constraints due to the interleaving of conflicts and to the conflict constraints are known before the assessment. The proposition 4.2 guarantees that all these constraints are known in a choice state (see the proof of the theorem 4.1).

Let be: a causal net \mathcal{C}_i in a choice state, the corresponding scenario \mathcal{S}_j , and the next choice x_{j+1} (with $\mu_{j+1}(x_{j+1}) = e_{i+1} \in PE(\mathcal{E}_i)$) in a derivative scenario \mathcal{S}_{j+1} . The *extended conflict constraints* of the event e_{i+1} is the set: $EConf(e_{i+1}) = \{e \mid e \in Sconf(\mathcal{E}_i) \vee e \in PE(\mathcal{E}_i) \wedge e \neq e_{i+1}\} = Conf(e_{i+1}) \cup \{e \mid e \wr e_{i+1}, e \in Sconf(\mathcal{E}_i) \cup PE(\mathcal{E}_i)\}$. So, the vicinity constraints are extended to the temporal effect of the concurrent choices.

All events in \mathcal{E}_i is assumed already getting a valid temporal assessment. The temporal assessment on x_{j+1} not only depends on the events in $\bullet\bullet e_{i+1}$, but also depends the ability to pass e_{i+1} after the past concurrent choices in \mathcal{E}_i and before any other future choice in $PE(\mathcal{E}_i)$. The assessment is done by applying the shortest path algorithm on the resulting STPs. The constraint graphs about the events of the scenario \mathcal{S}_j are completed by the following steps to obtain the scenario \mathcal{S}_{j+1} :

- in relation with an effective cause, define the constraint $[efd, lfd]$ for e_{i+1} and the other events in $PE(\mathcal{E}_i)$;
- if a past choice is in the local configuration of e_{i+1} , then the temporal order is naturally taken into account; else (the past choice is concurrent to e_{i+1}), the temporal order \rightsquigarrow is guaranteed by defining a constraint $[0, \infty]$ from this past choice in \mathcal{E}_i to e_{i+1} ;
- to express that e_{i+1} preempts any other choices in $PE(\mathcal{E}_i)$, a constraint $[0, \infty]$ is defined from e_{i+1} to each other choices in $PE(\mathcal{E}_i)$. These constraints guarantee the relation \rightsquigarrow with any future concurrent choice.

The derivative scenario \mathcal{S}_{j+1} is valid if it exists at least one consistent constraint graph.

For illustrations, the unfolding of the net figure 3(a) gives three scenarios:

- at (c): $\mathcal{S}_1 = \langle \mathcal{X}_1, \rightsquigarrow_1 \rangle$ with $\mathcal{X}_1 = \{x_1, x_2\}$, $\mu_1(x_1) = e_1$, $\mu_1(x_2) = e_3$, and $\rightsquigarrow_1 = \{(e_1, e_3)\}$;
- at (d): $\mathcal{S}_2 = \langle \mathcal{X}_2, \rightsquigarrow_2 \rangle$ with $\mathcal{X}_2 = \{x_3, x_4\}$, $\mu_2(x_3) = e_3$, $\mu_2(x_4) = e_1$, and $\rightsquigarrow_2 = \{(e_3, e_1)\}$;
- at (e): $\mathcal{S}_3 = \langle \mathcal{X}_3, \rightsquigarrow_3 \rangle$ with $\mathcal{X}_3 = \{x_5\}$, $\mu_3(x_5) = e_2$ and $\rightsquigarrow_3 = \{\}$.

Notice that the scenarios \mathcal{S}_1 and \mathcal{S}_2 get the same causal net, and correspond to the interleavings of the choices e_1 and e_3 . The unfolding of a TPN (like the example 3(b)) represents all the temporally possible events and conditions of the unfolding of the underlying net.

5 Cut-off issue

In practice, a system modeled with TPN often expresses behaviors with infinite runs. The purpose is to provide an algorithm to construct a finite unfolding of such TPN, that terminates when all the reachable states are represented, and to generate all prefixes of scenarios contained into the unfolding for permitting quantitative time analyses: in this way, the unfolding is said *complete*.

For a system with infinite runs, the finite construction of a state class graph [20, 21] is based on the concept of equivalence between state classes. In a state class, the temporal constraints are only relating to the firing that directly leads to this class, and do not cover all the sequence that leads to this class. It is certainly possible to retrieve the temporal profile of a sequence by a treatment on all the classes crossed by this sequence [22], but such a quantitative analysis is based on the interleaving semantics.

The state class graph is finite if and only if the TPN is bounded. Since the boundedness property is undecidable for TPNs, some sufficient conditions are given [21] to ensure the finiteness of the state class graph. Anyway,

the test of these conditions is based on the enumeration of all the reachable markings, and thus, seems difficult to extend in this context of the unfolding of a TPN where the global markings are not explicitly covered.

The concept of state equivalence is similar to the notion of cut-off in the branching process theory. For untimed Petri net [7], a *configuration* is defined as the set of events \mathcal{E}_i of a causal net \mathcal{C}_i , and $Mark(\mathcal{E}_i)$ represents the reachable marking corresponding to $Max(\mathcal{C}_i)$, that is the marking reached by producing all the events in \mathcal{E}_i . An event $e \in \mathcal{E}_i$ is a *cut-off event* if: $\exists e' \in \mathcal{E}_i, e' \prec e \wedge (Mark([e']) = Mark([e])) \wedge (\lambda(e') = \lambda(e))$. For an untimed Petri net, a cutoff event determines a local state already reached in the past: this local point of view is tolerable because the time does not restrict possible developments on concurrent events. There could exist many cutoff events in the same unfolding.

In a time Petri net, the cutoff notion needs to be linked to the global state of the scenarios because the local point of view could prevent to see how possible temporal restrictions on concurrent events could determine the future dynamics. During the unfolding of a causal net, concurrent events are produced in an arbitrary order without any consideration on the time, and it is only certain global states that are visited. In this context, a stop condition for a cut-off may concern a global state that is always visited whatever is the order adopted to produce the events. Only concurrent-free events generate such a state. In a causal net \mathcal{C}_i (with e_i the latest event produced as extension), e_i is a concurrent-free event if it gets no concurrent event in the past (\mathcal{E}_i) or in the future ($PE(\mathcal{E}_i)$), that is expressed by the following definition:

Definition 5.3. *An event e_i of a causal net \mathcal{C}_i is a concurrent-free event iff: $\forall e \in \mathcal{E}_i, e \in [e_i]$ and $\nexists e \in PE(\mathcal{E}_i), e \succ e_i$.*

Only concurrent-free events are candidate to be cutoff because the global states they generates are covered, whatever is the production order of the events \mathcal{E}_i . The first concurrent-free event is the fictitious event e_0 that generates the initial state. A *reset state* is the global state generates by a concurrent-free event: any event extensible from this state is a newly enabled transition.

A scenario will be stopped when the process of unfolding reaches a reset state already discovered (possibly in another previously computed scenario) with the same corresponding marking. The past state previously discovered is called a *return state*. The next event, from which the scenario is stopped by the *cutoff event*, and the event that generates a return state are called a *report event*. A cutoff event brings back to a return state where the same extensions of runs are possible from its report event. A quantitative analysis can be carried on from the report event on the computed prefix of the unfolding, and every event of a behavior exceeding the prefix can be reported to an event of this prefix.

For a safe TPN, a restriction may be stated to guarantee the finiteness and completeness of the unfolding thanks to the detection of an equivalent return state for any non-terminating run.

Assumption 51 *A finite number of events is produced between two consecutive reset states in any infinite run.*

This assumption is the relaxation of the assumption 41.

Theorem 5.2. *Under the assumption 51, a finite complete prefix exists and is computable for a TPN.*

Proof. 1. The system is safe and the set of transitions \mathcal{T} and places \mathcal{P} are finite. So, there is a finite number of reachable markings, and a finite number of enabled transitions per reachable marking. Consequently, there is a finite number of reset states.

2. An infinite run goes infinitely through at least one reset state since there are a finite number of reset states (item 1) and a finite number of states between two consecutive reset states (assumption ass:finite2). When a reset state is encountered for a second time, the unfolding of the scenario is stopped (cutoff detection). So, a finite prefix is computable for each infinite run.

3. The conflicts determine the nodes of derivation of multiple scenarios. There is a finite number of events per choice state. Consequently, since every scenario gets a finite prefix (item 2), there is a finite number of prefix of scenarios (or terminating scenarios with no cutoff).

4. The unfolding of a TPN results from the finite set of possible events computed in the scenarios. So, this unfolding is *finite* since there exists a finite number of scenarios (item 3). It is also *complete* since a cut-off expresses a return to a state from which the same reachable markings and dynamics are feasible.

The theorems and definitions resulting from the assumption 41 remain valid since an infinite run is captured by a finite prefix.

As it was aforementioned, the finiteness of the unfolding of a TPN is undecidable because it is based here on the detection of reachable markings. The fact that the reset states are a subset of the reachable marking makes this problem more difficult than for the state class graph construction.

In spite of the explosive nature of the state class graph, the finiteness of its construction may decide the finiteness of the unfolding. To ensure a finite construction of the unfolding, the solution proposed now and based on the state class graph keeps all the same an interest because allowing a partial order quantitative analysis unlike the method describes in [22]. The theorem given in the sequel is based on the following restricted class of TPNs: $\forall t \in \mathcal{T}, \exists p \in \mathcal{P} \mid p \in t^\bullet \wedge p \notin \bullet t$. So, every transition of the TPN gets an exclusively postcondition place.

Theorem 5.3. *If every elementary loop of the state class graph contains a firing t_x and the class C (resp. C') with the global marking m (resp. m') such that: $m' \xrightarrow{t_x} m \wedge \forall t_y \mid m \xrightarrow{t_y}, \bullet t_y \subseteq t_x^\bullet$, then the assumption 51 is always true.*

Proof. The elementary circuits are computable on the state class graph and the reset states are identified in the looped sequence of each circuit. The problem is the identification of the concurrent-free events on the edges between two state classes of the circuit.

The sufficient condition $m' \xrightarrow{t_x} m \wedge \forall t_y \mid m \xrightarrow{t_y}, \bullet t_y \subseteq t_x^\bullet$ makes the class C being a node from that any transition t_y is newly enabled and has t_x as the unique enabling event: there is no concurrent event following the firing t_x in a sequence.

It is also necessary to ensure that every predecessor event in a sequence is causal to t_x : there is no concurrent event preceding the firing t_x .

How to check this second implicit condition? Assume the existence of some event(s) temporally preceding the firing t_x but concurrent to t_x . Since every successor t_y of t_x gets the unique enabling cause t_x , it exists a predecessor firing t_z concurrent with t_x but without a successor event. After several loops on the circuit, the markings do not change. But the firing of t_z after many iterations of a circuit will strictly increase the marking in its next class because t_z gets a sink place (an exclusively postcondition place) in which the number of tokens increases indefinitely after each iteration without never be consumed by some firing. This is a contradiction, so the assumption of the existence of such a transition t_z is false.

Thus, t_x is a concurrent-free event and the class C corresponds to a reset state. Since the state class graph is finite, any infinite sequence follows the same (elementary) circuit a least two times, and then crosses a same reset state after a finite number of states or events. So, the assumption 51 is always checked.

6 Quantitative time analyses

The TPN unfolding contains all the temporally possible events (that is the events appearing in some scenario) of the underlying net. A complete prefix of the TPN unfolding covers the overall possible temporal behaviors of the system, and can help to operate some time quantitative analyses.

When the system (or subsystem) is always terminating, the full unfolding allows to directly obtain temporal informations about events. Thus, this work, that takes into account concurrency between events in conflict, is a generalization for an application on modular unfolding [12]; the time domain of an output event (terminating each scenario) is the time interval between an input event and this event, and is provided by the shortest path algorithm. In a general way, the possible delays between any two events can be obtained by the new method.

Another application is the checking the validness of a timing on a firing schedule (with absolute dates attached to the firings) or on a *time process* [23], even with non terminating systems, with no need to recompute

every time the corresponding causal net of the process as in [23], since this net is contained in the TPN unfolding. Given a firing schedule $(\sigma_n, \underline{\theta})$, the resulting causal net \mathcal{C}_n is identified progressively in the unfolding $TUnf$ by simulating the sequence $\sigma_n = t_1 \cdot \dots \cdot t_i \cdot \dots \cdot t_n$: from the net \mathcal{C}_{i-1} , the derivative net \mathcal{C}_i is obtained by finding the event e_i ($\lambda(e_i) = t_i$) in $PE(\mathcal{E}_{i-1})$ and by replacing in the B-cut $Max(\mathcal{C}_{i-1})$ the preconditions $\bullet e_i$ by the postconditions e_i^\bullet to obtain the new B-cut $Max(\mathcal{C}_i)$. Two inequations (*validness criteria* defined in [23]) are used to check that θ_i is a valid time: $\theta_i \geq Ed(e_i) + efd(t_i)$ and $\forall e_j \in PE(\mathcal{E}_{i-1}), Ed(e_j) + lfd(\lambda(e_j))$, where $Ed(e) = \max\{\theta_a \mid e_a \in \bullet\bullet e \cup \{e_0\}\}$ is the enabling date of the event e .

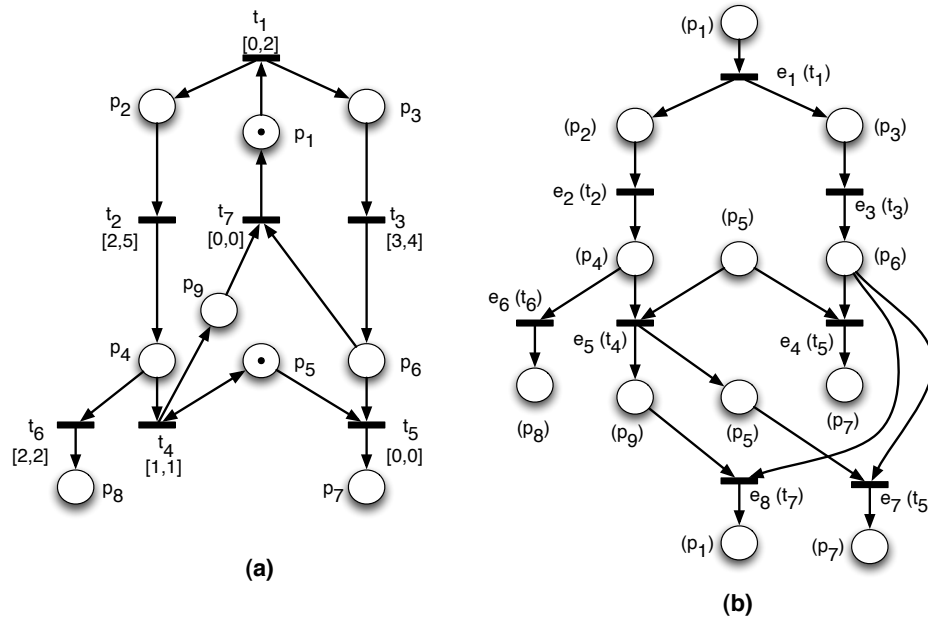


Fig. 4. Case study

The unfolding of the TPN figure 4(a) is made with three scenarios ³:

- Scenario 1: the set of events is $\{e_1, e_2, e_3, e_4, e_6\}$ with $e_4 \rightsquigarrow e_6$ (no cut-off);
- Scenario 2: the set of events is $\{e_1, e_2, e_3, e_5, e_7\}$ with $e_5 \rightsquigarrow e_7$ (no cut-off);
- Scenario 3: the set of events is $\{e_1, e_2, e_3, e_5, e_8\}$ with $e_5 \rightsquigarrow e_8$, and the cut-off is e_8 with the report event e_0 .

The events $\{e_1, \dots, e_8\}$ and their preconditions and postconditions make up $TUnf$ in figure 4(b).

For instance, if $\sigma_1 = t_1 \cdot t_2 \cdot t_3 \cdot t_4$ and $\underline{\theta}_1 = (0, 3, 3, 4)$, the causal net of the three first firings is made of the events $\{e_1, e_2, e_3\}$ with a valid timing; in this state, the last firing t_4 (event e_5) is invalid because the event e_4 also enabled necessarily occurs before at date 3. So, the firing schedule is unfeasible.

Even if a firing schedule or a time process exceeds the unfolding, the analysis could be carried on thanks to a report event. For instance, let be a firing schedule with $\sigma_2 = t_1 \cdot t_2 \cdot t_4 \cdot t_3 \cdot t_7 \cdot t_1 \cdot t_3 \cdot t_2 \cdot t_5 \cdot t_6$ and $\underline{\theta}_2 = (1, 3, 4, 5, 5, 7, 11, 11, 11, 13)$. When t_7 is fired, there is no more possible extensions in $TUnf$; since the prefix of sequence $t_1 \cdot t_2 \cdot t_4 \cdot t_3 \cdot t_7$ belongs to the third scenario that has the cutoff $e_8(t_7)$, the checking is continued from the conditions generated by the report event e_0 (here the initial state of $TUnf$). The suffix $t_1 \cdot t_3 \cdot t_2 \cdot t_5 \cdot t_6$ of σ_2 belongs to the first scenario that has no cutoff event. The timing of the sequence is valid,

³ For the sake of simplicity, an event of a scenario is denoted e instead of x .

and there is no more possible extension after firing t_6 . The existence of a non extensible scenario means that the system gets a deadlock state or a terminating state.

In [22], a method is presented to characterize one sequence in a state class graph by a timing profile. A third application here is the possibility to depict completely a TPN by a permanent set of quantitative time constraints characterizing all the possible runs of this net. It is based on the possibility to break down a scenario containing concurrent-free events into some parts for an autonomous quantitative time analysis.

Let be a process \mathcal{E} of TU_{nf} containing at least one concurrent-free event e_k : $\forall e \in \mathcal{E}, e \preceq e_k \vee e_k \preceq e$. Let be two events $e_i, e_j \in \mathcal{E}$ such as $e_i \prec e_k$ and $e_k \prec e_j$. If the time difference between e_i and e_k (resp. e_k and e_j) is denoted by the variable $D_{i,k}$ (resp. $D_{k,j}$), then the triangular relation $D_{i,j} = D_{i,k} + D_{k,j}$ is proven because a shortest path from e_i to e_j passes necessarily through e_k . So, e_k could be a reference event for e_j , instead of e_0 , to make the assessment of the time difference with any event e_i causal to e_k . Thus, the temporal assessment (see section 3) on any event e_j such as $e_k \prec e_j$ could have the reference e_k with no need to consider any event e_i coming before e_k .

Definition 6.4. *Let be two consecutive⁴ concurrent-free events e_1 and e_2 (with $e_1 \prec e_2$) in \mathcal{E} . The subset $\mathcal{E}' = \{e \in \mathcal{E} \mid e_1 \prec e \wedge e \preceq e_2\}$ is a minimal autonomous part of the process \mathcal{E} . The event $e_1 \notin \mathcal{E}'$ is the input event for \mathcal{E}' and the event $e_2 \in \mathcal{E}'$ is the output event for \mathcal{E}' .*

The set \mathcal{E}' is said *minimal* because it contains the only one concurrent-free event e_2 . It is said *autonomous* because a corresponding sub-scenario could be temporally assessed locally; the time difference with any outside event $e \in \mathcal{E} \setminus \mathcal{E}'$ may be established by the means of the input event e_1 when $e \prec e_1$ or the output event e_2 when $e_2 \prec e$. Notice that e_2 could be a cutoff event to map with one report event and e_1 could be the reference e_0 .

An autonomous subprocess \mathcal{E}' is said *terminating* when there is no possible future extension in the unfolding: if \mathcal{C}' is the corresponding causal net of \mathcal{E}' , then $PE(\mathcal{E}') = \emptyset$. So, a terminating minimal autonomous subprocess is a suffix of a terminating process broken down into its minimal autonomous subprocesses. The division of a scenario into some autonomous parts during the TPN unfolding may reduce the cost of temporal assessment because this evaluation is limited to the events of one autonomous part.

To illustrate, the (prefix of) scenarios of the net figure 4(a) gets three concurrent-free events: e_1, e_7 and e_8 . So, the minimal autonomous subprocesses are:

- $E_1 = \{e_1\}$ with the input event e_0 and the output event e_1 . E_1 is the prefix in each one of the previously given scenarios;
- $E_2 = \{e_2, e_3, e_4, e_6\}$ with the input event e_1 and no output event. E_2 is terminating and is the suffix of scenario 1;
- $E_3 = \{e_2, e_3, e_5, e_7\}$ with the input event e_1 and the output event e_7 . E_3 is terminating and is the suffix of scenario 2;
- $E_4 = \{e_2, e_3, e_5, e_8\}$ with the input event e_1 and the output event e_8 related to the report event e_0 . E_4 is the suffix of scenario 3.

Each one of these subprocesses corresponds to a sub-scenario characterized by a normal form of timing profile, acquired by the application of the shortest path algorithm on the events of this sub-scenario:

- for E_1 in the three scenarios: $0 \leq D_{0,1} \leq 2$;
- for E_2 with the interleaving $e_4 \rightsquigarrow e_6$ in scenario 1: $2 \leq D_{1,2} \leq 5 \wedge 3 \leq D_{1,3} \leq 4 \wedge 4 \leq D_{1,6} \leq 7 \wedge -2 \leq D_{2,3} \leq 2 \wedge 2 \leq D_{2,6} \leq 2 \wedge 0 \leq D_{5,6} \leq 4 \wedge D_{3,4} = 0$;
- for E_3 with the interleaving $e_5 \rightsquigarrow e_7$ in scenario 2: $(2 \leq D_{1,2} \leq 3 \wedge 3 \leq D_{1,3} \leq 4 \wedge 3 \leq D_{1,5} \leq 4 \wedge 1 \leq D_{2,3} \leq 2 \wedge D_{2,5} = 1 \wedge 0 \leq D_{5,3} \leq 1 \wedge D_{3,7} = 0) \vee (2 \leq D_{1,2} \leq 5 \wedge 3 \leq D_{1,3} \leq 4 \wedge 3 \leq D_{1,5} \leq 6 \wedge -2 \leq D_{2,3} \leq 1 \wedge D_{2,5} = 1 \wedge 0 \leq D_{5,3} \leq 3 \wedge D_{5,7} = 0)$;

⁴ e_1 and e_2 ($e_1 \prec e_2$ and $e_1, e_2 \in \mathcal{E}$) are *consecutive* concurrent-free events iff $\forall e \in \mathcal{E}$ such as $e_1 \prec e \prec e_2$, the event e is not concurrent-free.

- for E_4 with the interleaving $e_5 \rightsquigarrow e_8$ in scenario 3: $(2 \leq D_{1,2} \leq 3 \wedge 3 \leq D_{1,3} \leq 4 \wedge 3 \leq D_{1,5} \leq 4 \wedge 1 \leq D_{2,3} \leq 2 \wedge D_{2,5} = 1 \wedge 0 \leq D_{5,3} \leq 1 \wedge D_{3,8} = 0) \vee (2 \leq D_{1,2} \leq 5 \wedge 3 \leq D_{1,3} \leq 4 \wedge 3 \leq D_{1,5} \leq 6 \wedge -2 \leq D_{2,3} \leq 1 \wedge D_{2,5} = 1 \wedge 0 \leq D_{5,3} \leq 3 \wedge D_{5,8} = 0)$.

A timing profile could be extract from these constraints about any process of the TPN. The first step is to identify the minimal autonomous subprocesses composing this process by the simulation of $TUnf$. When a process exceeds $TUnf$, an event of the exceeding suffix may be reported to a mapping event e of $TUnf$, and the first event may be labeled by $e', e'', \dots e^n$, according to the number of instances of the equivalent events in the process. Then, the timing profile is the union of the normal forms of the constraints related to this subprocesses. For instance, let be a process $E = \{e_1, e_2, e_3, e_5, e_8, e'_1, e'_2, e'_3\}$. In order, the subprocesses are: $E_a = \{e_1\}$, $E_b = \{e_2, e_3, e_5, e_8 = e'_0\}$ and $E_c = \{e'_1\}$ and $E_d = \{e'_2, e'_3\}$. E_a , E_b and E_c is identified respectively with E_1 , E_4 and E_1 . The suffix E_d is included in each of the subprocesses E_2 , E_3 and E_4 but their timing profiles may not be considered just as there are. The corresponding timing profile must take into account the fact that the events in E_d preempts any concurrent in $PE(E) = PE(E_d)$. In this example, there is no concurrent for the events in E_d ; but in the general case, the temporal assessment must be made for a suffix not identifiable with some pre-computed subprocess, by taking into account the preemption of possible concurrent events in $PE(E)$. To conclude, the process E is characterized by the following timing profile obtained by the conjunction of the ones of the subprocesses:

$$(0 \leq D_{0,1} \leq 2) \wedge ((2 \leq D_{1,2} \leq 3 \wedge 3 \leq D_{1,3} \leq 4 \wedge 3 \leq D_{1,5} \leq 4 \wedge 1 \leq D_{2,3} \leq 2 \wedge D_{2,5} = 1 \wedge 0 \leq D_{5,3} \leq 1 \wedge D_{3,8} = 0) \vee (2 \leq D_{1,2} \leq 5 \wedge 3 \leq D_{1,3} \leq 4 \wedge 3 \leq D_{1,5} \leq 6 \wedge -2 \leq D_{2,3} \leq 1 \wedge D_{2,5} = 1 \wedge 0 \leq D_{5,3} \leq 3 \wedge D_{5,8} = 0)) \wedge (0 \leq D_{0',1'} \leq 2) \wedge (2 \leq D_{1',2'} \leq 5 \wedge 3 \leq D_{1',3'} \leq 4 \wedge -2 \leq D_{2',3'} \leq 2)$$

When one is interested to the time difference between two events in the process and not in the same sub-scenario, the triangular relation must be used by using the middle input or output events between these events. For instance, between e_2 and e'_2 , there is the references e_8 and e'_1 , so $D_{2,2'} = D_{2,8} + D_{0',1'} + D_{1',2'}$. The variable $D_{2,8} = D_{2,3}$ gets two possible domains $(1 \leq D_{2,8} \leq 2 \vee -2 \leq D_{2,8} \leq 1)$. Thus, $3 \leq D_{2,2'} \leq 9 \vee 3 \leq D_{2,2'} \leq 8$, that implies $D_{2,2'} \in [3, 9]$ (the occurrence periods of the event e_2 in an infinite run).

More complex analyses may be carry out from the timing profiles of a TPN unfolding, such as the possible delays between two states, knowing that the difference between the death date and the birth date of markings can be expressed as time difference between some corresponding events.

7 Conclusion

This work shows that the unfolding of a time Petri net requires some interleaving of conflicting events to assess the temporal possibility of the events of the underlying net unfolding. The generated complete finite prefix contains all the possible events of the system (modulo the dynamics from the return states). When the unfolding is finite, a prefix of scenario defines a permanent quantitative temporal relation between the events that form it. The interest is to get a complete and permanent database ($TUnf$ and the timing profiles about the subprocesses of the TPN unfolding) for an immediate temporal analysis (made in a polynomial time) like the checking of a time process or a firing schedule [23], the assessment of the time domain between two events or states, or the timing profile of a process. Contrary to the state class graph construction, the method is not based on the systematic interleaving of the concurrent events, that is the cause of the combinatory explosion when analyzing a complex system with a lot of parallelisms.

The restriction to have a point of synchronization (the concurrent-free events and the return states) after a finite number of actions is not unrealistic for a practical modeling: the modeler may render in the structure of the TPN the synchronization between concurrent events, and may not let the implicit orderings due to the timings settle the synchronizations.

This work imposes that the underlying net to be also safe. A perspective is to remove this restriction. An idea may be to reiterate (partially) the assessment on a scenario when, in the course of the unfolding, several

events in conflict get the same transition in the TPN. Indeed, such a conflict expresses a non-safeness of one preset place of these events, and the temporal reassessment will show that only one of these events is possible if the TPN is safe. Such an approach obviously implies more algorithmic complexity for the method.

References

1. Merlin, P.M.: A Study of the Recoverability of Computing Systems. PhD thesis, Department of Information and Computer Science, University of California (1974)
2. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Softw. Eng.* **17**(3) (1991) 259–273
3. Berthomieu, B., Vernadat, F.: State class constructions for branching analysis of time petri nets. In: TACAS. (2003) 442–457
4. Yoneda, T., Ryuba, H.: Ctl model checking of time petri nets using geometric regions. *IEICE Trans.* **E81-D**(3) (1998) 297–396
5. Sloan, R., Buy, U.: Reduction rules for time petri nets. *Acta Informatica* **33** (1996) pp 687–706
6. Bucci, G., Vicario, E.: Compositional validation of time-critical systems using communicating time petri nets. *IEEE Trans. Softw. Eng.* **21**(12) (1995) 969–992
7. Esparza, J., Römer, S., Vogler, W.: An improvement of mcmillan’s unfolding algorithm. *TACAS* **1**(1055) (1996)
8. McMillan, K.L.: Using unfolding to avoid the state explosion problem in the verification of asynchronous circuits. In: Proc. Fourth Workshop on Computer-Aided Verification. Volume 663., Montreal, Canada, Springer-Verlag (june 1992) pp 164–177
9. Engelfriet, J.: Branching processes of petri nets. *Acta Informatica* **28**(6) (june 1991) pp 575– 591
10. Semenov, A., Yakovlev, A.: Verification of asynchronous circuits using time petri net unfolding. In: DAC ’96: Proceedings of the 33rd annual conference on Design automation, New York, NY, USA, ACM (1996) 59–62
11. Chatain, T., Jard, C.: Complete finite prefixes of symbolic unfoldings of safe time petri nets. Springer-Verlag **4024** (2006) pp 125–145
12. Sogbohossou, M., Delfieu, D.: Temporal reduction in time petri net. In: IDT 2008, Monastir, Tunisia (December 2008)
13. Delfieu, D., Sogbohossou, M., Traounez, L.M., Revol, S.: Parameterized study of a time petri net. In: CITSA 2007, Orlando, Florida, USA (July 2007)
14. Merlin, P.M., Faber, D.J.: Recoverability of communication protocol - implications of theoretical study. *IEEE Transactions on Communications* **COM24** (September 1976) pp 1036–1043
15. Cerone, A., Maggiolo-Schettini, A.: Time-based expressivity of time petri nets for system specification. *Theoretical Computer Science* **216**(1-2) (1999) 1 – 53
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. The MIT Press and McGraw-Hill Book Company (2001)
17. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. *Artif. Intell.* **49**(1-3) (1991) 61–95
18. Pradin-Chézalviel, B., Valette, R., Künzle, L.A.: Scenario durations characterization of t-timed petri nets using linear logic”. In: PNPM’99, 8th International Workshop on Petri Nets and Performance Models, Zaragoza, Spain (1999) 208–217
19. Girard, J.Y.: Linear logic. *Theoretical Computer Science* **50** (1987) 1–102
20. Berthomieu, B., Menasche, M.: An enumerative approach for analyzing time petri nets. In: IFIP Congress. (1983) 41–46
21. Berthomieu, B., Diaz, M.: Modeling and verification of time dependent systems using time petri nets. *IEEE Trans. Software Eng.* **17**(3) (1991) 259–273
22. Vicario, E.: Static analysis and dynamic steering of time-dependent systems. *IEEE Trans. Softw. Eng.* **27**(8) (2001) 728–748
23. Aura, T., Lilius, J.: Time processes for time petri-nets. In: ICATPN. Volume 1248 of LNCS. (1997) 136–155