Modeling of timed Petri nets using deterministic (max,+) automata

Sébastien Lahaye * **Jan Komenda** ** **Jean-Louis Boimond** *

* *LUNAM Université, LISA, Angers, France.*
sebastien.lahaye@univ-angers.fr, jean-louis.boimond@univ-angers.fr.
** *Institute of Mathematics - Brno Branch, Czech Academy of Sciences, Czech Republic.* komenda@ipm.cz

Abstract: Automata with weights (multiplicities) in (max,+) algebra form a class of timed automata. Determinism is a crucial property for numerous results on (max,+) automata and, in particular, for applications to performance evaluation and control of a large class of timed discrete event systems. In this paper, we show how to build a deterministic (max,+) automaton equivalent to a live and safe timed Petri net in which any oriented path between any two transitions contains at most one "conflict place".

Keywords: Petri nets, (max,+) automata, modeling, determinization

1. INTRODUCTION AND MOTIVATIONS

An important class of timed discrete-event systems (TDES) can be captured by means of timed Petri nets (TPNs) with deterministic timing of places and/or transitions [16]. A corresponding class of timed automata with deterministic timing of transitions is known as (max,+) automata [5] that are automata with weights (multiplicities) in (max,+) algebra [1].

(Max,+) automata have been applied to performance evaluation [5,14,17,3], scheduling [8] and control [11,2] problems for a large class of TDES. Beyond the scope of TDES, there are important applications for image and speech processing [15], and more generally, weighted automata constitute a theoretical object extensively studied (see [4] for an overview).

The modeling power of (max,+) automata is studied in [7], where it is shown that the behavior of any safe [2] TPN can be expressed by a *heap model*. It has been shown that the height of heaps of pieces is recognized by a particular (max,+) automaton (especially useful for algebraic computations). The (max,+) automaton derived from the example of heap model is depicted [3] on the right-hand side of Figure 1.

Another approach in [12] proposes a direct transformation of any safe TPN into a (max,+) automaton. For example and as reminded in section 3, the (max,+) automaton of Figure 2 is obtained to represent the TPN of Figure 3.

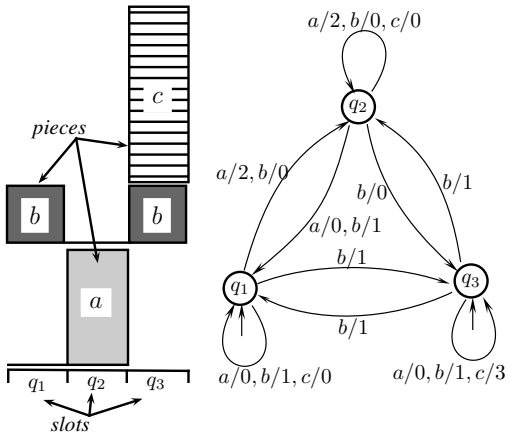

Fig. 1. Heap model and associated (max,+) automaton to represent TPN $\mathcal{G}$.

Observe that these (max,+) automata have larger languages [4] than the language of the TPN. For example, they recognize sequence $aaa\ldots$ whereas consecutive firings of transition a (without firing b) are impossible. As it is done classically in automata theory, it is then possible to adequately restrict the (max,+) automata languages by using the tensor product with the (Boolean) marking automaton recognizing the TPN language.

In addition, these (max,+) automata are both nondeterministic [5]. Except for trivial examples, this situation always occurs with the two approaches and this is a serious drawback. Determinism is indeed important for

[1] Set $\mathbb{Z} \cup \{-\infty\}$ endowed with the maximum playing the role of addition $\oplus$ and the conventional addition playing the role of multiplication $\otimes$ is an idempotent semiring, usually called (max,+) algebra, and denoted $\mathbb{Z}_{\max}$. We denote $e = 0$ and $\varepsilon = -\infty$ the neutral elements of $\oplus$ and $\otimes$.

[2] At most one token can be in a place at any time.

[3] The graphical representation of a (max,+) automaton is such that: nodes correspond to states, an arrow from a state to another with label a/n denotes a state transition requiring n units of time when event a occurs, an input arrow symbolizes an initial state.

[4] If A is a finite set (*alphabet*), the free monoid on A is defined as the set A^* of finite words with letters in A. A *word* $w \in A^*$ can be written as a sequence $w = a_1 a_2 \ldots a_n$ with a_1, a_2, ..., $a_n \in A$ and n a natural number (the zero-length word is denoted e in the following). *Formal languages* are subsets of the free monoid A^*.

[5] Several initial states and/or several transitions share the same event from certain states.

many results and, for example, it is a required property for (max,+) automata

- to do efficient computations in speech processing [15];
- to compute the optimal, as well as the average behavior, for TDES [5];
- to realize supervisors for TDES [11,2].

Unlike (Boolean) finite automata, it has been shown that nondeterministic (max,+) automata cannot always be determinized, that is transformed into equivalent deterministic (max,+) automata (see for example [13]).

In the continuity of [7], authors proposed in [6] a determinization procedure based on the *completion* of heap models [6]. Returning to the TPN from which the heap model is built, this means that a deterministic (max,+) automaton representation can be obtained if any two transitions share at least one input or output place (see Th.4 and the discussion in [6]). This condition is not satisfied for transitions a and c of the TPN in Fig. 3, and the procedure then fails for this example.

Another approach is to use a determinization procedure applying directly to (max,+) automata. This extension of the classical determinization algorithm of Boolean automata has been extensively studied (see e.g. [5,15,10,13,9]). In few words, a *normalization* operator is defined to quantify the gaps between the dates at which various states are reached at the end of a transition sequence. If a finite number of vectors of gaps is obtained for all the possible sequences, then it is possible to build a normalized automaton which is deterministic and which has the same behavior as the initial nondeterministic automaton. In the case of (max,+) automata derived to represent TPN in Fig. 3, the determinization procedure based on normalization fails. For (max,+) automata obtained to represent TPNs according to the approach recalled in Section 3, such an unfavourable circumstance arises, in particular, as soon as two transitions do not share any input and output place.

In this paper, we propose another way to build deterministic (max,+) automata describing safe TPNs. This new procedure jointly uses the reachability graph of the TPN to cover its language, and the (nondeterministic) (max,+) automaton derived from the TPN to compute the weights to be associated with transitions so that the timed behavior is properly described. As already notified this procedure cannot always succeed (remember that all the (max,+) automata cannot be determinized) but, interestingly, it terminates successfully for a wider class than the existing approaches mentioned above. Then a condition on the structure of the TPN is shown to be sufficient for termination of the procedure. More precisely, the main result can be expressed as: if the oriented path between any two transitions contains at most one "conflict place" (with more than one output transition), then the procedure builds a deterministic (max,+) automaton equivalent to the safe TPN. For example, this condition is satisfied by the TPN in Fig. 3, and the procedure builds the deterministic (max,+) automaton in Fig. 5.

The paper is organized as follows. In the following section basic concepts and notations are introduced. In section 3, the approach proposed in [12] to transform any safe TPN into a nondeterministic (max,+) automaton is briefly presented. This result is used in the procedure proposed in section 4. If it terminates, this procedure constructs a deterministic (max,+) automaton equivalent to the safe TPN. Then, a structural condition on TPNs is shown to be sufficient for the termination of the procedure. In section 5, concluding remarks with hints on future extensions are proposed.

[6] In [14], the case of heap models with two pieces is fully treated.

2. PRELIMINARIES

Several necessary concepts, results and notations about (max,+) algebra, (max,+) automata and Petri nets are introduced in this section (see the references [1], [5] and [16] for exhaustive presentations).

2.1 (Max,+) algebra and automata

The set of $n \times n$ matrices with coefficients in (max,+) algebra $\mathbb{Z}_{\max}$, endowed with the matrix addition and multiplication conventionally defined from $\oplus$ and $\otimes$, is also an idempotent semiring, denoted $\mathbb{Z}_{\max}^{n\times n}$. The zero element for the addition is the matrix denoted ε_n and exclusively composed of ε ($= -\infty$). We denote I_n the identity element of the multiplication, which is the matrix with e ($= 0$) on the diagonal and ε ($= -\infty$) elsewhere. As usual in conventional algebra, the multiplication symbol $\otimes$ will be often omitted in the following.

(Max,+) automata can be defined as follows.

Definition 2.1. A *(max,+) automaton* G is a quadruple (Q, A, α, μ) where

- Q and A are finite sets of states and of events;
- $\alpha \in \mathbb{Z}_{\max}^{1\times|Q|}$ is such that $\alpha_q = e$ or $\alpha_q = \varepsilon$, and q is an initial state if $\alpha_q = e$;
- $\mu : A^* \to \mathbb{Z}_{\max}^{|Q|\times|Q|}$ is a morphism specified by the family of matrices $\mu(a) \in \mathbb{Z}_{\max}^{|Q|\times|Q|}$, $a \in A$, and for a string $w = a_1 a_2 \dots a_n \in A^*$, we have $\mu(w) = \mu(a_1 a_2 \dots a_n) = \mu(a_1) \otimes \mu(a_2) \otimes \dots \otimes \mu(a_n)$. A coefficient $[\mu(a)]_{qq'} \neq \varepsilon$ means that, from state q, the occurrence of event a causes a state transition to state q'.

Equivalently G can be defined by the quadruple (Q, A, Q_i, t), in which Q_i denotes the set of *initial* states

$$Q_i \triangleq \{q \in Q : \alpha_q = e\},$$

and $t : Q \times A \times Q \to \mathbb{Z}_{\max}$ is the *transition function*

$$t(q, a, q') \triangleq [\mu(a)]_{qq'}.$$

With this def., all states can be thought of as final states (as for automata derived from heap models [7]).

A coefficient $[\mu(a)]_{qq'} \neq \varepsilon$ (equiv., $t(q, a, q') \neq \varepsilon$) means that, from state q, the occurrence of event a causes a state transition to state q', and value $[\mu(a)]_{qq'}$ is interpreted as the duration associated to event a (namely, the activation time of event a before it can occur). If $[\mu(a)]_{qq'} \neq \varepsilon$, then we denote (q, a, q') the *transition* in G. Let $m \geq 0$ and $\pi = (q_0, a_1, q_1)(q_1, a_2, q_2) \dots (q_{m-1}, a_m, q_m)$ be a sequence of transitions. We call π a *path* from q_0 to q_m. We denote

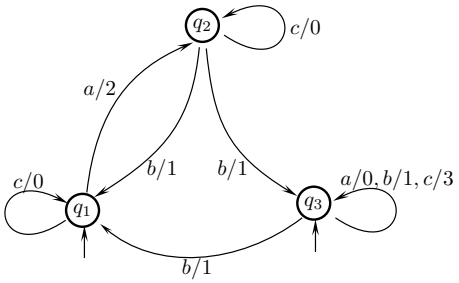

Fig. 2. (Max,+) automaton G.

$\sigma(\pi)$ the product $\otimes$ (i.e. the usual sum)of the weights on π, that is

$$\sigma(\pi) = \bigotimes_{i=1,\dots,m} t(q_{i-1}, a_i, q_i) = \bigotimes_{i=1,\dots,m} [\mu(a_i)]_{q_{i-1},q_i}.$$

Let $p, q \in Q$ and $w \in A^*$. We denote by $p \overset{w}{\rightsquigarrow} q$ the set of paths from p to q which are labeled by w. It can be shown that

$$[\mu(a_1 a_2 \dots a_m)]_{q_0 q_m} = \bigoplus_{\pi \in q_0 \overset{a_1 \dots a_m}{\rightsquigarrow} q_m} \sigma(\pi) \ . \tag{1}$$

This shows that $[\mu(a_1 a_2 \dots a_m)]_{q_0 q_m}$ corresponds to the maximum weight among the paths recognizing $a_1 a_2 \dots a_m$.

A (max,+) automaton is said to be *deterministic* if it has a unique initial state and from each state, no two state transitions share the same event (namely, if for all $a \in A$ each line of $\mu(a)$ contains at most one element not equal to ε).

For deterministic automata, $\forall q, q' \in Q$, $\forall w \in A^*, q \overset{w}{\rightsquigarrow} q'$ is the empty set or a singleton. Let us denote $\pi = (q_0, a_1, q_1)(q_1, a_2, q_2) \dots (q_{m-1}, a_m, q_m)$ the unique path recognizing $a_1 a_2 \dots a_m$ from q_0 to q_m, equation (1) is then reduced to

$$\begin{aligned} [\mu(a_1 a_2 \dots a_m)]_{q_0,q_m} &= \bigotimes_{i=1\dots m} [\mu(a_i)]_{q_{i-1},q_i}, \\ &= \bigotimes_{i=1\dots m} t(q_{i-1}, a_i, q_i). \end{aligned} \tag{2}$$

In order to describe the dynamic evolution of (max,+) automaton G, vector $x(w) \in \mathbb{Z}_{\max}^{1\times|Q|}$ for $w \in A^*$ is defined by

$$x(w) = \alpha\mu(w) \ . \tag{3}$$

An element $[x(w)]_q$ is interpreted as the date at which state q is reached at the conclusion of the sequence w starting from an initial state (with the convention that $[x(w)]_q = \varepsilon$ if state q is not reached from an initial state using the input sequence w). The language of a (max,+) automaton contains the words $w \in A^*$ such that there exists a state q with $[x(w)]_q \neq \varepsilon$. The elements of x are *generalized daters*, and we have

$$\begin{cases} x(\epsilon) = \alpha, \\ x(wa) = x(w)\mu(a). \end{cases} \tag{4}$$

Below, since all the states are considered as final, we denote y the sum $\oplus$ of all the coefficients of vector x, that is:

$$\forall w \in A^*, \ y(w) = \bigoplus_{q \in Q} [x(w)]_q \ . \tag{5}$$

2.2 Timed Petri nets

Definition 2.2. (Petri net). A *Petri net* (PN) $\mathcal{G}$ is a 4-tuple $(\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$, in which $\mathcal{P}$ is a finite set of places, $\mathcal{T}$ is a finite set of transitions, $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a relation between places and transitions, $M_0 : \mathcal{P} \to \mathbb{N}$ defines the *initial marking* of places (that is, the number of tokens they contain initially). [7]

For transition $a \in \mathcal{T}$, ${}^\bullet a$ (resp. $a^\bullet$) denotes the set of its input (resp. output) places. The marking evolves according to the following rules:

(1) Transition a is *enabled* at marking M if there exists at least one token in each input place $q \in {}^\bullet a$.
(2) An enabled transition a can fire. The firing of a transforms M into M' (written $M \overset{a}{\to} M'$) by removing one token from each input place $q \in {}^\bullet a$ and adding one token in each output place $q' \in a^\bullet$.

We say that a word $w = a_1 a_2 \dots a_n \in \mathcal{T}^*$ is a *firing sequence* starting from marking M_0 if there is a sequence of markings $M_1 M_2 \dots M_n$ such that transition a_i is enabled at M_{i-1} and its firing transforms M_{i-1} into M_i. We call *language* of the PN the set $\mathcal{L} \subset \mathcal{T}^*$ of firing sequences starting from initial marking M_0.

The *set of reachable markings* from initial marking M_0 of PN $\mathcal{G}$ will be denoted $\mathcal{R}(\mathcal{G}, M_0)$. A PN is said to be *safe* if for all reachable marking each place contains at most one token. If $\mathcal{G}$ is safe then set $\mathcal{R}(\mathcal{G}, M_0)$ is finite.

Let us denote $\mathbb{B}$ the subdioid of $\mathbb{Z}_{\max}$ defined on set $\{e, \varepsilon\}$. The *marking automaton* of a safe PN $\mathcal{G}$ is the deterministic boolean automaton $G_{\mathcal{R}} = (\mathcal{R}(\mathcal{G}, M_0), \mathcal{T}, \alpha_{\mathcal{R}}, \mu_{\mathcal{R}})$ defined by

- $\alpha_{\mathcal{R}} : \mathcal{R}(\mathcal{G}, M_0) \to \mathbb{B}$, $[\alpha_{\mathcal{R}}]_M = \begin{cases} e & \text{if } M = M_0, \\ \varepsilon & \text{otherwise,} \end{cases}$
- $\mu_{\mathcal{R}} : \mathcal{T} \to \mathbb{B}^{|\mathcal{R}(\mathcal{G},M_0)|\times|\mathcal{R}(\mathcal{G},M_0)|}$, $[\mu_{\mathcal{R}}(a)]_{MM'} = \begin{cases} e & \text{if } M \overset{a}{\to} M', \\ \varepsilon & \text{otherwise.} \end{cases}$

By construction, a word $w \in \mathcal{T}^*$ is a firing sequence of $\mathcal{G}$ *iff* it is accepted by its marking automaton. If a PN is safe and live, then it is strongly connected (that is, for every pair of nodes there exists an oriented path from one node to the other). In the following, we restrict our attention to such live and safe PNs.

We call *repetitive sequence* from $M \in \mathcal{R}(\mathcal{G}, M_0)$ a firing sequence $v = a_0 \dots a_n$ such that $M \overset{a_0}{\to} M_1 \dots \overset{a_n}{\to} M$.

We consider TPNs in which a finite firing duration τ_a is associated with each transition a: τ_a is the minimal time that must elapse, starting from the time at which a is enabled, until this transition can fire.

Example 1. A PN $\mathcal{G}$ is usually represented by a bipartite oriented graph as in Fig. 3. The two types of nodes are places in $\mathcal{P}$ and transitions in $\mathcal{T}$ represented respectively by circles and bars with the associated labels and durations. An element of $\mathcal{F}$ is displayed by an arrow from a place to a transition or from a transition to a place. The

[7] We restrict our attention to PNs which are *ordinary* and *free-labelled* (each transition is labeled by a single event and there are no two transitions with the same label).

initial marking is represented by $[M_0]_q$ tokens in place q. PN $\mathcal{G}$ is live and safe. Marking automaton $G_{\mathcal{R}}$ of $\mathcal{G}$ is also displayed on Fig. 3.

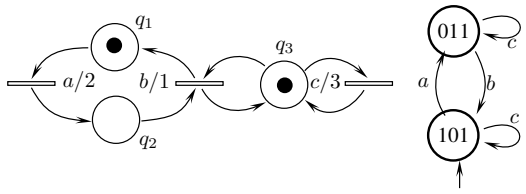

Fig. 3. TPN $\mathcal{G}$ and its marking automaton $G_{\mathcal{R}}$.

Several assumptions on the functioning of PNs are considered hereafter:

- a token from the initial marking is supposed to arrive in the TPN at time instant 0;
- if a place has several output transitions, then for each token in this place it needs to be decided which transition is to fire (that is, in case of a *conflict*). In the present work, all the logically feasible choices are considered for the decision. This corresponds to a so-called *preselection policy*, in contrast to models for which the decision is rather based on time considerations (e.g. using the *race policy*). Note that choices corresponding to the race policy are considered as particular cases with the policy adopted here;
- when a transition is to be fired, then it is fired as soon as possible.

We define $x_{\mathcal{G}}(w) \in \mathbb{Z}_{\max}^{1\times|\mathcal{P}|}$, the vector of variables associated with places $q \in \mathcal{P}$ and function of firing sequence $w \in \mathcal{T}^*$ by

$$[x_{\mathcal{G}}(w)]_q = \begin{cases} \text{instant at which the last token has arrived} \\ \quad \text{in } q \text{ after } w \text{ (assuming that it is still in } q), \\ \varepsilon \text{ if } q \text{ does not contain any token after } w. \end{cases} \tag{6}$$

Below, we denote $y_{\mathcal{G}}$ the sum $\oplus$ of coefficients of vector $x_{\mathcal{G}}$, that is:

$$\forall w \in \mathcal{T}^*,\ y_{\mathcal{G}}(w) = \bigoplus_{q\in\mathcal{P}} [x_{\mathcal{G}}(w)]_q\ . \tag{7}$$

3. DESCRIPTION OF SAFE TPN BY POSSIBLY NONDETERMINISTIC (MAX,+) AUTOMATA

In [7], authors have shown that the behavior of any safe TPN can be modeled by a (max,+) automaton. Their approach consists of two steps: a heap representation for such PNs is first proposed, then a (max,+) automaton is derived from the heap model. The following proposition given in [12] specifies how to directly derive a (max,+) automaton representing the timed behavior of a safe TPN.

Proposition 3.1. Let $\mathcal{G} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M_0)$ be a live and safe TPN. A (max,+) automaton $G = (Q, A, \alpha, \mu)$ is derived from $\mathcal{G}$ as follows:

$$Q = \mathcal{P},\ A = \mathcal{T},$$

$$\forall q \in Q,\ \alpha_q = \begin{cases} e & \text{if } [M_0]_q = 1, \\ \varepsilon & \text{otherwise,} \end{cases}$$

$\forall q, q' \in Q$, $\forall a \in A$,

$$[\mu(a)]_{qq'} = \begin{cases} \tau_a & \text{if } q \in {}^\bullet a \text{ and } q' \in a^\bullet, \\ e & \text{if } q = q' \text{ and } q \notin {}^\bullet a \cup a^\bullet, \\ \varepsilon & \text{otherwise.} \end{cases}$$

We have, for all $w \in \mathcal{L}$,

$$x(w) = x_{\mathcal{G}}(w)\ . \tag{8}$$

Example 2. (Max,+) automaton G displayed on Figure 2 is the one obtained by means of procedure described in Proposition 3.1 in order to represent safe TPN $\mathcal{G}$ in Figure 3.

Remark 3.2. Note that (max,+) automaton G defined in Proposition 3.1 is nondeterministic as soon as

- the corresponding TPN $\mathcal{G}$ has several places initially marked (then G has several initial states);
- the corresponding TPN $\mathcal{G}$ has a transition a with several output places, i.e. $a^\bullet > 1$ (then there exists at least a line in $\mu(a)$ which contains more than one non-zero element).

4. DESCRIPTION OF SAFE TPN BY DETERMINISTIC (MAX,+) AUTOMATA

To the best of our knowledge and as discussed in the introduction, results of the literature frequently fail to determinize (max,+) automata obtained to represent safe TPNs (in particular the automata displayed in figures 1 and 2). This observation has motivated the synthesis of a new procedure to build deterministic (max,+) automata describing live and safe TPNs. This procedure is introduced in the next subsection together with a sufficient structural condition on TPN for its termination.

4.1 New procedure to describe safe TPNs by deterministic (max,+) automata

A new procedure is proposed to build a deterministic (max,+) automaton $G' = (Q', A, q'_i, t')$ which is equivalent to a safe and live TPN $\mathcal{G}$. Models G' and $\mathcal{G}$ are said to be equivalent because $y'(w)$ defined by Eq. (5) for G' is equal to $y_{\mathcal{G}}(w)$ defined by Eq. (7) for $\mathcal{G}$ for every $w \in \mathcal{T}^*$. Note that:

- G' (unlike G considered in Prop. 3.1) does not have necessarily the same number of states as the number of places in $\mathcal{G}$. This is why we require the equality $y'(w) = y_{\mathcal{G}}(w)$ instead of $x(w) = x_{\mathcal{G}}(w)$ as in Eq. (8). The situation is identical in [7] where the height of heaps of pieces is shown to compute the completion date of a firing sequence in the TPN.
- G' and $\mathcal{G}$ have the same language, while the equality of vectors in Prop. 3.1 is only true for w belonging to language $\mathcal{L} \subseteq \mathcal{T}^*$ of $\mathcal{G}$.

Marking automaton $G_{\mathcal{R}}$ is used to cover $\mathcal{L}$ and possibly nondeterministic (max,+) automaton G defined in Prop. 3.1 is used to compute the weights to be associated with the transitions. In very few words, the procedure unfolds $G_{\mathcal{R}}$ and computes weights to be associated with transitions in G' so that y' represents the same timed behavior as TPN $\mathcal{G}$, that is $y'(w) = y_{\mathcal{G}}(w)$, $\forall w \in \mathcal{T}^*$. The states of G' are tuples composed of a reachable marking and a firing sequence leading to this marking. Compared to existing approaches discussed in the introduction, the key feature

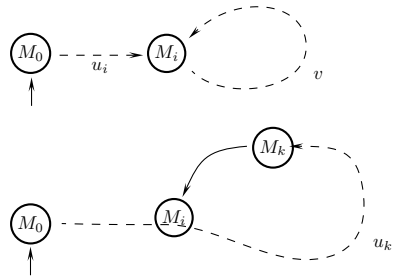

Fig. 4. Partial representation of unfolding of $G_{\mathcal{R}}$.

is that the procedure covers the language $\mathcal{L}$ of TPN $\mathcal{G}$ while building the corresponding (max,+) automaton G'.

Let us state some conventions used for the notations in the procedure proposed below:

- Figure 4 is a partial depiction of the marking automaton $G_{\mathcal{R}}$ which is unfolded by the procedure. In the following,v denotes by convention a repetitive sequence of transitions starting from and ending to state M_i. String u_i denotes a sequence with length $|u_i| = i$ from M_0 and leading to state M_i. Note that u_k for $k \geq i$ may have both u_i and v^n, with n a natural number, for subwords.
- As usual, $\mathrm{Tr}(A)$ denotes the trace of matrix A, i.e. the (max,+) sum $\oplus$ (maximum) of its diagonal elements.

Example 3. For PN $\mathcal{G}$ displayed in Figure 3, the proposed procedure builds deterministic (max,+) automaton G' depicted in Figure 5. In order to illustrate how the procedure operates to build G', let us detail some steps. Initial state q'_i is defined as the tuple $(M_0, u_0) = ((101), \mathrm{e})$. When RecursiveUnfolding is called with $((101), \mathrm{e})$ as argument and $Q' = \{((101), \mathrm{e})\}$, then possible iterations on line 9 are:

- a with $M' = (011)$: the condition on line 10 is false since $M' \notin Q'$ and so the tuple $((011), a)$ is added in Q'. State transition $t'(((101), \mathrm{e}), a, ((011), a)) = y(a) - y(\mathrm{e}) = 0 + 2$ is defined and RecursiveUnfolding is called with $((011), a)$ as argument and $Q' = \{((101), \mathrm{e}), ((011), a)\}$,
- c with $M' = (101)$: the condition on line 10 is true with $(M_0, u_0) = ((101), \mathrm{e})$, but the condition on line 11 is false since $y(ca) - y(c) = (3 - 3) \neq y(a) - y(\mathrm{e})(= 2)$. So tuple $((101), c)$ is added in Q', $t'(((101), \mathrm{e}), c, ((101), c)) = y(c) - y(\mathrm{e}) = 3$ and RecursiveUnfolding is called with $((101), c)$ as argument and $Q' = \{((101), \mathrm{e}), ((101), c)\}$.

When RecursiveUnfolding is called with $((011), a)$ as argument and $Q' = \{((101), \mathrm{e}), ((011), a)\}$, then possible iterations on line 9 are:

- b with $M' = (101)$: the condition on line 10 is true with $(M_0, u_0) = ((101), \mathrm{e})$, the condition on line 11 is true since $^y(aba) - y(ab)(= 5-3) = y(a) - y(\mathrm{e})(= 2-0)$ and $y(abc) - y(ab)(= 6 - 3) = y(c) - y(\mathrm{e})(= 3 - 0)$, and the condition on line 12 is true since

$$\mu(ab) = \begin{pmatrix} 3 & \varepsilon & 3 \\ \varepsilon & \varepsilon & \varepsilon \\ 1 & \varepsilon & 1 \end{pmatrix}, \quad \mathrm{Tr}(\mu(ab)) = 3,$$

Procedure building deterministic (max,+) automaton $G' = (Q', A, q'_i, t')$ equivalent to $\mathcal{G}$

1: Build $G_{\mathcal{R}} = (\mathcal{R}(\mathcal{G}, M_0), \mathcal{T}, \alpha_{\mathcal{R}}, \mu_{\mathcal{R}})$ the marking automaton of $\mathcal{G}$
2: Build (max,+) automaton $G = (Q, A, \alpha, \mu)$ from $\mathcal{G}$ using Prop. 3.1
3: **if** G is nondeterministic **then**
4: $q'_i \leftarrow (M_0, \mathrm{e})$ ▷ (M_0, u_0) with $u_0 = \mathrm{e}$ is the initial state
5: $Q' \leftarrow q'_i$ ▷ adds the initial state in set Q'
6: RecursiveUnfolding(M_0, e) ▷ Calls the recursive procedure
7: **end if**

8: **procedure** RecursiveUnfolding(M_k, u_k)
9: **for all** $a \in \mathcal{T}, M' \in \mathcal{R}(\mathcal{G}, M_0)$ s.t. $[\mu_{\mathcal{R}}(a)]_{M_k M'} \neq \varepsilon$ **do**
10: **if** $\exists (M_i, u_i) \in Q'$ such that $M' = M_i$,
11: $y(u_k ab) - y(u_k a) = y(u_i b) - y(u_i)$ for all $b \in \mathcal{T}$,
12: $y(u_k a) = \mathrm{Tr}(\mu(v)) + y(u_i)$ with $v \in A^*$ s.t. $u_k a = u_i v$ **then**
▷ State $(M', u_k a)$ is equivalent to (M_i, u_i) in Q'
▷ loops on state (M_i, u_i) and stops unfolding
13: $t'((M_k, u_k), a, (M_i, u_i)) \leftarrow y(u_k a) - y(u_k)$
14: **else**
▷ state $(M', u_k a)$ is not equivalent to a state in Q'
▷ adds the state in Q'
15: $Q' \leftarrow Q' \bigcup \{(M', u_k a)\}$
▷ defines the state transition
16: $t'((M_k, u_k), a, (M', u_k a)) \leftarrow y(u_k a) - y(u)$
▷ calls the procedure to keep on unfolding
17: RecursiveUnfolding$(M', u_k a)$
18: **end if**
19: **end for**
20: **end procedure**

and $y(ab)(= 3) = \mathrm{Tr}(\mu(ab)) + y(\mathrm{e})(= 3 + 0)$. Then $t'(((011), a), b, ((101), \mathrm{e})) = y(ab) - y(a) = 1$ and this instance of RecursiveUnfolding ends.
- c with $M' = (011)$: the condition on line 10 is true with $(M_1, u_1) = ((011), a)$, but the condition on line 11 is false since $y(acc) - y(ac)(= 6 - 3) \neq y(ac) - y(a)(= 3 - 2)$. Tuple $((011), ac)$ is added in Q', $t'(((011), a), c, ((011), ac)) = y(ac) - y(a) = 3 - 2$ and RecursiveUnfolding is called with $((011), ac)$ as argument and $Q' = \{((101), \mathrm{e}), ((101), c), ((011), ac)\}$.

4.2 Sufficient condition for the termination of the procedure

Unfortunately, the above procedure does not always terminate because the number of states in G' may be infinite (no surprise here since it is well-known that not all the (max,+) automata can be determinized, see e.g. [13]).

In this subsection, it is shown that the following property for $\mathcal{G}$ is sufficient for the termination of the procedure.

Property 4.1. For every pair of transitions (q, q'), there exists an oriented path from q to q' which contains at most one place in set $\mathcal{P}_s$ defined by

$$\mathcal{P}_s \triangleq \{q \in \mathcal{P} | q^\bullet > 1\}.$$

In other words, this class of live and safe TPNs is identified to be equivalently modeled by deterministic (max,+)

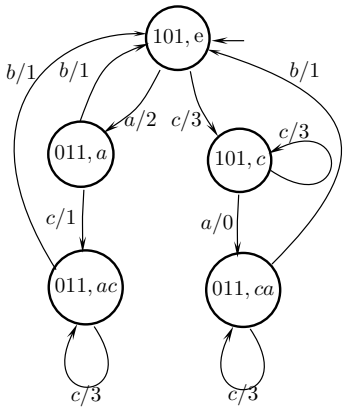

Fig. 5. Deterministic (max,+) automaton G' equivalent to safe TPN $\mathcal{G}$ represented in Figure 3.

automata. Let us emphasize that our contribution extends existing results on this topic as pointed out in Remark 4.6 below. In particular, for TPN $\mathcal{G}$ displayed in Fig. 3 the procedures from the literature fail to build a deterministic (max,+) automaton representation (see the discussion in the introduction), but it satisfies Property 4.1 and the proposed procedure then succeeds to build G' which is deterministic and equivalent to $\mathcal{G}$ (see Ex. 3 and Prop. 4.5).

Before stating the main result in Proposition 4.5, we give several properties for TPN $\mathcal{G}$ and (max,+) automaton G obtained using Proposition 3.1. Note that we denote x and y (instead of $x_{\mathcal{G}}$ and $y_{\mathcal{G}}$) the daters associated with $\mathcal{G}$ since they are equal to the daters associated to G according to 3.1.

We say that two circuits in a Petri net $\mathcal{G}$ denoted by $w_1 \in \mathcal{T}^*$, $w_2 \in \mathcal{T}^*$ are *disjoint* if they don't have any common place and transition, and if $\forall a \in w_1$, $\forall b \in w_2$, $({}^\bullet a \cup a^\bullet) \cap ({}^\bullet b \cup b^\bullet) = \emptyset$.

Lemma 4.2. Let $\mathcal{G}$ be a live and safe PN. If $\mathcal{G}$ satisfies Property 4.1, then there does not exist two disjoint circuits denoted by $w_1 \in \mathcal{T}^*$, $w_2 \in \mathcal{T}^*$ such that $w_1^n w_2^m$ with $n > 1$ and $m > 1$ is a possible firing sequence.

Lemma 4.3. Let $\mathcal{G}$ be a live and safe PN. Let $w_1 \in \mathcal{T}^*$, $w_2 \in \mathcal{T}^*$ denoting two circuits in $\mathcal{G}$ having common transition(s) and/or place(s), or such that $\exists a \in w_1$, $\exists b \in w_2$ with $({}^\bullet a \cup a^\bullet) \cap ({}^\bullet b \cup b^\bullet) \neq \emptyset$. If $\mathcal{G}$ satisfies Property 4.1, then for all $v \in \mathcal{T}^*$ a repetitive sequence [8] on places in w_1 and w_2 there exists a marked place p in w_1 and/or w_2 such that $\forall u_i \in \mathcal{T}^i$,

$$[x(u_i)]_p \geq [x(u_i)]_q \text{ and } [x(u_i v)]_p \geq [x(u_i v)]_q$$

for all q a place in w_1 and/or w_2.

Lemma 4.4. If $\mathcal{G}$ satisfies Property 4.1, then there exists a (uniform) bound $k \in \mathbb{N}$ such that $\forall u_k \in \mathcal{T}^*$, $k \geq 1$, there exists a finite length repetitive sequence $v^j \in \mathcal{T}^*$, $j \in \mathbb{N}$ (circuit in the marking automaton of $\mathcal{G}$) such that there is a place $p \in \mathcal{P}$ that is marked after both u_i and $u_i v^j$, with

$$y(u_i) = [x(u_i)]_p \text{ and } y(u_i v^j) = [x(u_i v^j)]_p. \quad (9)$$

Proposition 4.5. If $\mathcal{G}$ satisfies Property 4.1, then the procedure of Subsec. 4.1 terminates and builds a deterministic (max,+) automaton G' which is equivalent to TPN $\mathcal{G}$.

[8] The marking of places in w_1 and w_2 is the same before and after firing sequence v.

Remark 4.6. Note that the special class of Petri nets for which there exists a pair of transitions (q, q') such that any oriented path from q to q' contains more than one "conflict" place (i.e. in $\mathcal{P}_s$) is much weaker than the requirement that there exist transitions which do not share any input and output place.

5. CONCLUSION

The main result in this paper is the constructive proof that a safe TPN can be equivalently represented by a deterministic (max,+) automaton provided that the oriented path between any two transitions contains at most one "conflict place". The complexity of our determinization procedure should be addressed in future work. It is also our plan to transpose this result into a determinization procedure applied directly to (max,+) automata.

REFERENCES

[1] F. Baccelli, G. Cohen, G.-J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. Wiley, 1992.

[2] E. Badouel, A. Bouillard, P. Darondeau, and J. Komenda. Residuation of tropical series: rationality issues. In *CDC-ECC'11*, pages 3855–3861, 2011.

[3] R. Boukra, S. Lahaye, and J.-L. Boimond. New representations for (max,+)-automata with applications to the performance evaluation of discrete event systems. In *WODES'12*.

[4] M. Droste, W. Kuich, and H. Vogler (Eds.). *Handbook of Weighted Automata*. Springer, 2009.

[5] S. Gaubert. Performance Evaluation of (max,+) Automata. *IEEE TAC*, 40(12):2014–2025, 1995.

[6] S. Gaubert and J. Mairesse. Asymptotic analysis of heaps of pieces and application to timed Petri nets. In *PNPM'99*, pages 158 – 169, 1999.

[7] S. Gaubert and J. Mairesse. Modeling and analysis of timed Petri nets using heaps of pieces. *IEEE TAC*, 44(4):683–698, 1999.

[8] L. Houssin. Cyclic jobshop problem and (max,plus) algebra. In *18th IFAC WC*, Milan, Italy, 2011.

[9] D. Kirsten. A burnside approach to the termination of Mohri's algorithm for polynomially ambiguous min-plus-automata. *RAIRO*, 42(3):553–581, 2008.

[10] I. Klimann, S. Lombardy, J. Mairesse, and C. Prieur. Deciding unambiguity and sequentiality from a finitely ambiguous max-plus automaton. *TCS*, 2004.

[11] J. Komenda, S. Lahaye, and J.-L. Boimond. Supervisory Control of (max,+) Automata: A Behavioral Approach. *Disc. Event Dyn. Syst.*, 19(4):525–549, 2009.

[12] S. Lahaye, J. Komenda, and J.-L. Boimond. Compositions of (max,+) automata. *Disc. Event Dyn. Syst.* To appear.

[13] S. Lombardy and J. Sakarovitch. Sequential ? *Theoretical Computer Science*, 359(1-2):224–244, 2006.

[14] J. Mairesse and L. Vuillon. Asymptotic behavior in a heap model with two pieces. *Theoretical Computer Science*, 270(12):525 – 560, 2002.

[15] M. Mohri. Weighted automata algorithms. In *Handbook of weighted automata*. Springer, 2011.

[16] C. Ramchandani. *Analysis of asynchronous concurrent syst. by timed PN*. Ph.d. thesis, M.I.T., 1973.

[17] Rong Su and Gerhard J. Woeginger. String execution time for finite languages: Max is easy, min is hard. *Automatica*, 47(10):2326–2329, 2011.