To appear in the Proceedings of the ECC'97, Bruxelles, July 1997.

MODELING AND ANALYSIS OF TIMED PETRI NETS USING HEAPS OF PIECES

STÉPHANE GAUBERT† JEAN MAIRESSE‡

† INRIA, Domaine de Voluceau, B.P. 105, 78153 Le Chesnay Cedex, France.
email: Stephane.Gaubert@inria.fr, fax: +33 (0)1 39 63 57 86

‡ LIAFA, CNRS-Université Paris 7, 4 place Jussieu, 75252 Paris Cedex 05.
email: mairesse@litp.ibp.fr

Abstract

We show that safe timed Petri nets can be represented by special automata over the (max,+) semiring, which compute the height of heaps of pieces. This extends to the timed case the classical representation à la Mazurkievicz of the behavior of safe Petri nets by trace monoids and trace languages. For a subclass including all safe Free Choice Petri nets, we obtain reduced heap realizations using structural properties of the net (covering by safe state machine components). We illustrate the heap-based modeling by the typical case of safe jobshops. For a periodic schedule, we obtain a heap-based throughput formula, which is simpler to compute than its traditional timed event graph version, particularly if one is interested in the successive evaluation of a large number of possible schedules.

Keywords Timed Petri nets, automata with multiplicities, heaps of pieces, (max,+) semiring, scheduling.

I. Introduction

The purpose of this paper[1] is to prove the following result:

Timed safe Petri nets are special (max,+) automata, which compute the height of heaps of pieces.

Indeed, the firing times of the transitions of the net can be read on the upper contour of a Tetris-like heap of pieces, canonically associated to an (earliest) execution of the system.

This representation theorem extends to the timed case the representation à la Mazurkiewicz [18], [11] of the behavior of safe (untimed) Petri nets by *trace languages* and *trace monoids*. The isomorphism between trace monoids and *heap monoids* was noticed in Viennot [23]. The fact that the height of heaps of pieces is recognized by a (max,+) automaton, the result holding for general, polyomino-shaped, pieces was proved in Gaubert and Mairesse [14], and in a different form, by Brilman and Vincent [24], [6].

Heap representations are particularly well adapted to algebraic computations. As a typical illustration, we derive an heap-based performance evaluation method for safe jobshops. The assignment of the jobs on the machines is fixed but not the order on which the jobs are processed by the machines (the schedule). The classical modeling associates with each chosen schedule an event graph (i.e. a (max,+) linear system) whose size grows with the period of the schedule, see [16], [2]. On the other hand, the representation by heap model is independent of the (even non periodic) schedule which is considered. This is particularly interesting for the successive evaluation of a large number of schedules. For a periodic schedule, we propose a new heap-based algorithm to compute the throughput of the jobshop. The traditional (event graph based) method runs in *quadratic* time, with respect to the length of the schedule period. The execution time of the heap-based algorithm is of the same order, except that it is only *linear* in the length of this period.

It is worth noting that heaps of pieces are essentially an extension of the Gantt charts traditionally used in scheduling. Whereas conventional Gantt charts only display the resource (machine) occupation times, heaps of pieces contain the complete time information for both resources *and* jobs, which allows us to write dynamical equations.

Let us mention some related independent work of Hulgaard [17] on the subclass of safe Free Choice Petri net. Hulgaard did not put forward the automata or heap model, but he did introduce (for time analysis purposes) dater variables and dynamical equations similar to the ones used here.

The trace monoid interpretation is detailed in the long version of this paper[1].

II. Heap Models and (max,+) Automata

The following heap model generalises the heaps of pieces of Viennot [23]. Imagine an horizontal axis with a finite number of slots. A *piece* is a solid (possibly non connected) "block" occupying some of the slots, with staircase-shaped upper and lower contours, see Fig. 1. With an ordered sequence of pieces, we associate a *heap* by piling up the pieces, starting from an horizontal ground. This piling occurs in the intuitive[2] way: a piece is only subject to vertical translations and occupies the lowest possible position, provided it is above the ground and the pieces previously piled up. Let us propose a more formal definition.

DEFINITION II.1. A *heap model* is a 5-tuple $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, where:

- $\mathcal{T}$ is a finite set whose elements are called *pieces*.
- $\mathcal{R}$ is a finite set whose elements are called *slots*.
- $R : \mathcal{T} \to \mathcal{P}(\mathcal{R})$ gives the subset of slots occupied by a piece. We assume that each piece occupies at least one

[1] This paper has been abridged due to the lack of space. In particular it contains no proofs. The interested reader can obtain the extended version from the authors or from the Web pages: http://amadeus.inria.fr/gaubert, and http://www-litp.ibp.fr/˜mairesse/.

[2] It corresponds for example to the mechanism of the Tetris game.

slot: $\forall a \in \mathcal{T}, R(a) \neq \emptyset$.
• $l : \mathcal{T} \times \mathcal{R} \longrightarrow \mathbb{R} \cup \{-\infty\}$ gives the height of the lower contour of the piece at the different slots. $u : \mathcal{T} \times \mathcal{R} \longrightarrow \mathbb{R} \cup \{-\infty\}$ (with $u \geq l$) gives the height of the upper contour of the piece. By convention, $l(a,r) = u(a,r) = -\infty$ if $r \notin R(a)$ and $\min_{r \in R(a)} l(a,r) = 0$.

A piece a occupies a region of the $\mathcal{R} \times \mathbb{R}^+$ plane of the form $\{(r,y) \in R(a) \times \mathbb{R}^+ \mid \lambda + l(a,r) \leq y \leq \lambda + u(a,r)\}$, where $\lambda \in \mathbb{R}^+$ depends on the pieces already piled up, and $\lambda = 0$ if there is no other piece. We will interpret a length k word[3] $w = a_1, \dots, a_k \in \mathcal{T}^*$ as a heap, i.e. as a sequence of k pieces $a_1, \dots, a_k$ piled up in this logical order. We define the *upper contour* of the heap w as the $\mathcal{R}$-dimensional row vector $x_{\mathcal{H}}(w)$, where $x_{\mathcal{H}}(w)_r$ is the height of the heap on slot r. The horizontal ground assumption yields $x_{\mathcal{H}}(e) = (0, \cdots, 0)$. The *height* of the heap w is

$$y_{\mathcal{H}}(w) = \max_{r \in \mathcal{R}} x_{\mathcal{H}}(w)_r \ .$$

A useful interpretation of a heap model consists in viewing pieces as tasks and slots as resources. Each task a requires a subset of the resources (given by $R(a)$) during a certain amount of time ($u(a,r) - l(a,r)$ for a resource $r \in R(a)$). In the simplest case where $l(a,r) = 0, \forall r \in R(a)$, the execution of a task begins as soon as all the required resources, used by earlier tasks, become free. For more details along these lines, see [14], [6]. Borrowing the terminology of [2], [13], the maps $y_{\mathcal{H}}$ and $x_{\mathcal{H}}$ are called the *dater functions* of the heap model. The piling mechanism and the different notations are best understood graphically and on an example, see Fig. 1.

Example II.2. Let us consider the following heap model.
• $\mathcal{T} = \{a,b,c,d\}$, $\mathcal{R} = \{1,2,3,4\}$;
• $R(a) = \{1,2,3\}$, $R(b) = \{1,2\}$, $R(c) = \{2,4\}$, $R(d) = \{2,3,4\}$;
• $u(a,.) = [1,1,3,-\infty], l(a,.) = [0,0,0,-\infty]$,
$u(b,.) = [3,2,-\infty,-\infty], l(b,.) = [0,0,-\infty,-\infty]$,
$u(c,.) = [-\infty,2,-\infty,2], l(c,.) = [-\infty,0,-\infty,0]$,
$u(d,.) = [-\infty,1,3,1], l(d,.) = [-\infty,0,0,0]$.

We have represented, in Fig. 1, the heap associated with the word $w = abcd$. Piece c is an example of a non connected (but "rigid") piece. We can read directly on Fig. 1 the values $x_{\mathcal{H}}(w) = [4,6,8,6]$ and $y_{\mathcal{H}}(w) = 8$.

DEFINITION II.3. The (max,+) semiring $\mathbb{R}_{\max}$ is the set $\mathbb{R} \cup \{-\infty\}$, equipped with the operation $\max$, written additively (i.e. $a \oplus b = \max(a,b)$) and the usual sum, written multiplicatively (i.e. $a \otimes b = a + b$). In this semiring, $\mathbb{0} = -\infty$, $\mathbb{1} = 0$.

Note that $\mathbb{R}_{\max}$ is a dioid, i.e. a semiring with an idempotent addition ($a \oplus a = a$). We shall use throughout the paper the matrix and vector operations induced by the semiring structure. For matrices A, B of appropriate sizes, $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \max(A_{ij}, B_{ij})$, $(A \otimes B)_{ij} = \bigoplus_k A_{ik} \otimes B_{kj} = \max_k(A_{ik} + B_{kj})$, and for a scalar a, $(a \otimes A)_{ij} = a \otimes A_{ij} = a + A_{ij}$. We will omit the $\otimes$ sign, writing for instance AB instead of $A \otimes B$ as usual. Given a set $\mathcal{S}$, we denote by $\mathbb{1}_{\mathcal{S}}$ the $|\mathcal{S}|$-dimensional column vector whose entries are all equal to $\mathbb{1}$.

[3] We recall the following standard notations. Given a finite set (alphabet) $\mathcal{T}$, we denote by $\mathcal{T}^n$ the set of words of length n on $\mathcal{T}$. We denote by $\mathcal{T}^* = \cup_{n \in \mathbb{N}} \mathcal{T}^n$ the free monoid on $\mathcal{T}$, that is, the set of finite words equipped with concatenation. The unit (empty word) will be denoted e. The length of the word w will be denoted $|w|$. We shall write $|w|_a$ for the number of occurrences of a given letter a in w.

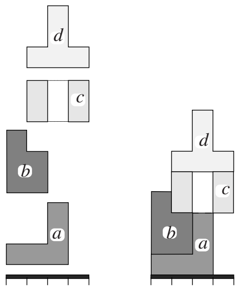


Fig. 1. Heap of pieces associated with the word $w = abcd$.

DEFINITION II.4. Given a finite alphabet $\mathcal{T}$, a (max,+) automaton[4] is a 4-tuple $\mathcal{A} = (Q, I, F, \mathcal{M})$, where

• Q is a finite set (of *states*);
• $I \in \mathbb{R}_{\max}^{1 \times Q}$ and $F \in \mathbb{R}_{\max}^{Q \times 1}$ are the *initial* and *final* vectors, respectively;
• $\mathcal{M}$ is a morphism $\mathcal{T}^* \to \mathbb{R}_{\max}^{Q \times Q}$.

The morphism $\mathcal{M}$ is uniquely specified by the finite family of $Q \times Q$ matrices, $\mathcal{M}(a), a \in \mathcal{T}$. Then, for a word $w = a_1 \dots a_n$, we have

$$\mathcal{M}(w) = \mathcal{M}(a_1 \dots a_n) = \mathcal{M}(a_1) \dots \mathcal{M}(a_n) \ , \qquad (1)$$

the matrix product being interpreted in the (max,+) semiring.

Let us define the vectors $x_{\mathcal{A}}(w) = I\mathcal{M}(w) \in \mathbb{R}_{\max}^{1 \times Q}$ and the scalars $y_{\mathcal{A}}(w) = I\mathcal{M}(w)F \in \mathbb{R}_{\max}$ associated with the (max,+) automaton $\mathcal{A}$. We say that $x_{\mathcal{A}}$ and $y_{\mathcal{A}}$ are *recognized*[5] by the automaton $\mathcal{A}$. We have

$$\begin{aligned} x_{\mathcal{A}}(e) &= I \ , \\ x_{\mathcal{A}}(wa) &= x_{\mathcal{A}}(w)\mathcal{M}(a) \ , \\ y_{\mathcal{A}}(w) &= x_{\mathcal{A}}(w)F \ . \end{aligned} \qquad (2)$$

Hence a (max,+) automaton may be seen as a (max,+) linear system whose dynamic is driven by letters. With a heap model $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, we associate the morphism $\mathcal{M} : \mathcal{T}^* \to \mathbb{R}_{\max}^{\mathcal{R} \times \mathcal{R}}$, given by

$$\mathcal{M}(a)_{sr} = \begin{cases} \mathbb{1} & \text{if } s = r, r \notin R(a), \\ u(a,r) - l(a,s) & \text{if } r \in R(a), s \in R(a), \\ \mathbb{0} & \text{otherwise.} \end{cases} \qquad (3)$$

THEOREM II.5. *Let $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$ be a heap model. The (max,+) automaton $(\mathcal{R}, \mathbb{1}_{\mathcal{R}}^t, \mathbb{1}_{\mathcal{R}}, \mathcal{M})$ recognizes the upper contour $x_{\mathcal{H}}$ and the height $y_{\mathcal{H}}$:*

$$\forall w \in \mathcal{T}^*, \qquad \begin{aligned} x_{\mathcal{H}}(w) &= \mathbb{1}_{\mathcal{R}}^t \mathcal{M}(w) \ , \\ y_{\mathcal{H}}(w) &= \mathbb{1}_{\mathcal{R}}^t \mathcal{M}(w) \mathbb{1}_{\mathcal{R}} \ . \end{aligned}$$

[4] This is the specialization to the (max,+) semiring of the classical notion of automaton with multiplicity [12], or recognizable series [22], [4].

[5] Classically, in automata theory, only the map $y_{\mathcal{A}}$ is said to be recognized. It is convenient here to extend the definition of recognizability.

A variant of this result was proved in [14], see also [6]. We will call $(\mathcal{R}, \mathbb{1}^t_{\mathcal{R}}, \mathbb{1}_{\mathcal{R}}, \mathcal{M})$ the heap automaton representing the heap model.

Example II.6. Let $\mathcal{T} = \{a, b, c, d\}$, $Q = \{1, 2, 3, 4\}$, $I = \mathbb{1}^t_Q$, $F = \mathbb{1}_Q$.

$$\mathcal{M}(a) = \begin{bmatrix} 1 & 1 & 3 & \\ 1 & 1 & 3 & \\ 1 & 1 & 3 & \\ & & & \mathbb{1} \end{bmatrix}, \quad \mathcal{M}(b) = \begin{bmatrix} 3 & 2 & & \\ 3 & 2 & & \\ & & \mathbb{1} & \\ & & & \mathbb{1} \end{bmatrix},$$

$$\mathcal{M}(c) = \begin{bmatrix} \mathbb{1} & & & \\ & 2 & & 2 \\ & & \mathbb{1} & \\ & 2 & & 2 \end{bmatrix}, \quad \mathcal{M}(d) = \begin{bmatrix} \mathbb{1} & & & \\ & 1 & 3 & 1 \\ & 1 & 3 & 1 \\ & 1 & 3 & 1 \end{bmatrix}.$$

(the $\mathbb{0}$ entries are omitted).

One verifies easily that the (max,+) automaton $(Q, I, F, \mathcal{M})$ represents the heap model given in Ex. II.2. We have

$$\begin{aligned} \mathcal{M}(abcd) &= \begin{bmatrix} 4 & 6 & 8 & 6 \\ 4 & 6 & 8 & 6 \\ 4 & 6 & 8 & 6 \\ \mathbb{0} & 3 & 5 & 3 \end{bmatrix}, \\ x_{\mathcal{A}}(abcd) &= I\mathcal{M}(abcd) = \begin{bmatrix} 4 & 6 & 8 & 6 \end{bmatrix}, y_{\mathcal{A}}(abcd) = 8 . \end{aligned}$$

This provides an algebraic confirmation of the values obtained graphically in Fig 1.

III. Untimed and Timed Petri Nets

A. Notation

The basic definitions of Petri nets are assumed to be known. For more details on Petri nets, see Murata [20], Desel and Esparza [10] or the proceedings [5]. We use the following notations: $(\mathcal{G}, M)$ a Petri net with initial marking M, $\mathcal{P}$ the set of places, $\mathcal{T}$ the set of transitions, $\mathcal{F}$ the set of arcs, ${}^\bullet x$, $x^\bullet$, the sets of predecessors and successors of a node x, $M' \xrightarrow{w} M''$ if the firing of $w \in \mathcal{T}^*$ transforms the marking M' into M'', $L \subset \mathcal{T}^*$ the ($\mathbf{P}$-type) language of the net (set of firing sequences starting from M). A net is *safe* if it is 1-bounded.

Example III.1. The Petri net represented in Fig. 2 is live and safe. Its language is $L = (ab \cup cd)^*(e \cup a \cup c)$. The interpretation of this net in terms of manufacturing systems will be detailed in § V below.

B. Timing

A timed Petri net (TPN) is a net with *firing times* associated with transitions and/or *holding times* associated with places. Several firing semantics have been considered in the literature. We restrict our attention to safe Petri nets and consider a semantic which coincides with the one of Ramchandani [21].

Let a be a transition with firing time τ_a and whose output places $p \in a^\bullet$ have holding times τ_p. We assume that transition a becomes enabled at instant t. A firing occurs in three steps.

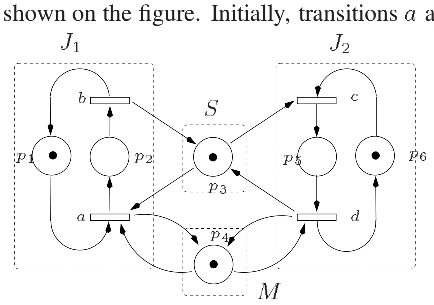
1. At instant t, the firing of a may be *initiated*. If initiated, it removes one token from each input place.
2. One token is added in each of the output place at instant $t + \tau_a$.
3. The token added in place $p \in a^\bullet$ can contribute to the enabling of the transitions in $p^\bullet$ after instant $t + \tau_a + \tau_p$.

Between t and $t + \tau_a$, the tokens can be considered as being 'frozen' in their original input place. The tokens and the transition a can not be involved in any other firing between t and $t + \tau_a$. A natural question to ask is what happens if transition a is not initiated at instant t. Then, either a never fires or it eventually gets disabled (because of the safeness property, this happens precisely when the token of one of the places in ${}^\bullet a$ participates in the firing of another transition). If transition a gets re-enabled later on, we are back to the original situation.

Let us investigate some other consequences of this semantic. First of all, if a transition fires, it does so "as soon as possible". We say that the Petri net operates with an *earliest firing rule*. Second, the decisions about which transitions are to fire is not based on time considerations. All logically feasible choices can be considered. This contrasts with several models studied in the literature. For example, in the so-called *race policy*, see for instance [1], a place with several output transitions allocates its token to the transition which is able to complete its firing first.

We denote by $(\mathcal{G}, M, \tau)$ a timed Petri net, where τ is a map $\mathcal{T} \cup \mathcal{P} \to \mathbb{R}^+$, providing the firing and holding times of transitions and places. By convention, the timed evolution of the Petri net starts at instant 0, in marking M, the holding times of the initial tokens being completed.

Example III.2. Let us consider the Petri net of Fig. 2, with holding and firing times shown on the figure. Initially, transitions a and c are enabled. If we choose to fire transition c, the firing will be initiated at time 0, completed at time 2, and transition d will become enabled at time 2.



Holding and Firing Times

a	b	c	d	p_1	p_2	p_3	p_4	p_5	p_6
1	2	2	1	1	0	0	2	0	0

Fig. 2. A two jobs — two resources manufacturing system

IV. Heap Representation for Safe Timed Petri Nets

A. Heap Representation Theorem

In this section, we state the main representation theorem of the paper: *firing times of safe timed Petri nets are recognized by heap automata*.

Let $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$ denote a safe timed Petri net, with set of firing sequences $L \subset \mathcal{T}^*$. The timed behavior of the net is defined as follows: for a firing sequence $w = a_1 \ldots a_k \in L$, we start at time 0, and fire the transitions $a_1, \ldots, a_k$ in this order, applying the earliest firing semantics described in §III-B above.

With each place p, we associate the real nonnegative numbers:

$z(w)_p$ = *instant at which the last token arrived in place p under the schedule w becomes available for the firing of downstream transitions.*

$z'(w)_p$ = *last instant of presence of a token in place* p*, under the schedule* w.

We set $z'(w)_p = z(w)_p = 0$, if no token was ever present in place p. We set

$$x_{\mathcal{G}}(w)_p = \begin{cases} z(w)_p & \text{if } M \xrightarrow{w} M', \text{ with } M'(p) = 1 \\ z'(w)_p & \text{if } M \xrightarrow{w} M', \text{ with } M'(p) = 0 \end{cases} \quad (4)$$

In words, this is the completion time of the last "event" at place p, under schedule w, an "event" being either the availability, or the departure of a token.

We call $x_{\mathcal{G}}$ the *dater vector* of the Petri net. We extend $x_{\mathcal{G}}$ to $\mathcal{T}^*$ by setting $x_{\mathcal{G}}(w)_p = \mathbb{0}$, for $w \notin L$.

The *makespan* or *execution time* of the firing sequence w is naturally defined by

$$y_{\mathcal{G}}(w) = \max_p x_{\mathcal{G}}(w)_p \quad . \quad (5)$$

In words, this is the completion time of the last "event" in the Petri net under schedule w.

THEOREM IV.1 (HEAP REPRESENTATION FOR SAFE TPN). *Let* $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$ *be a safe timed Petri net. Then, the heap model* $\mathcal{H} = (\mathcal{T}, \mathcal{P}, R, l, u)$*, with*

$$\begin{aligned} \forall a \in \mathcal{T}, \qquad R(a) &= a^\bullet \cup {}^\bullet a \ , \\ \forall a \in \mathcal{T}, \forall p \in a^\bullet, \qquad u(a,p) &= \tau_a + \tau_p \ , \\ \forall a \in \mathcal{T}, \forall p \in {}^\bullet a \setminus a^\bullet, \qquad u(a,p) &= 0 \ , \\ \forall a \in \mathcal{T}, \forall p \in R(a), \qquad l(a,p) &= 0 \ , \end{aligned}$$

is such that

$$\forall w \in L, \qquad x_{\mathcal{G}}(w) = x_{\mathcal{H}}(w), \ y_{\mathcal{G}}(w) = y_{\mathcal{H}}(w) \quad . \quad (6)$$

Equation (6) states that the dater vector of the net coincides with the upper contour of the associated heap.

Note that for $w \notin L$, the heap associated with w and the daters $x_{\mathcal{H}}(w)$ and $y_{\mathcal{H}}(w)$ are still defined. However, they have no meaning in terms of the Petri net.

Example IV.2. Let us illustrate the previous construction with the timed Petri net $(\mathcal{G}, M) = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M, \tau)$ represented in Fig. 2. The heap model associated with $(\mathcal{G}, M)$ is $\mathcal{H} = (\mathcal{T}, \mathcal{P}, R, l, u)$ with

$$\begin{aligned} R(a) &= {}^\bullet a \cup a^\bullet = \{p_1, p_2, p_3, p_4\}, R(b) = \{p_1, p_2, p_3\}, \\ R(c) &= \{p_3, p_5, p_6\}, R(d) = \{p_3, p_4, p_5, p_6\} \ . \\ u(a, .) &= [0, 1, 0, 3, \mathbb{0}, \mathbb{0}], u(b, .) = [3, 0, 2, \mathbb{0}, \mathbb{0}, \mathbb{0}], \\ u(c, .) &= [\mathbb{0}, \mathbb{0}, 0, \mathbb{0}, 2, 0], u(d, .) = [\mathbb{0}, \mathbb{0}, 1, 3, 0, 1] \ . \end{aligned}$$

We have represented the heap of pieces associated with the schedule $w = abcdabcd$ in Fig. 3. For the clarity of the figure, pieces with parts of zero width have been materialized by replacing zero by a 'small' but strictly positive height. The reader could check directly (by simulation of the Petri net) that the height of the heap, $y_{\mathcal{G}}(w) = 16$, attained at slot 4, corresponds to the last occurrence of an event in the system (the availability of the token in p_4, after the firing of the last occurrence of d in the schedule w).

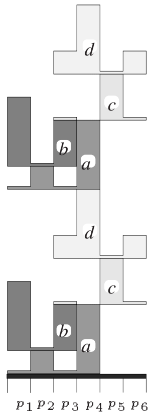


Fig. 3. Heap of pieces associated with the word $w = abcdabcd$.

B. The Minimal Realization Problem for Heaps of Pieces

The size of the heap representation in Theorem IV.1 is equal to the number of places of the Petri net, a possibly large number. This raises naturally the following *minimal realization* problem: *what is the minimal size of a heap representation of a given safe timed Petri net ?* As a partial answer to this probably difficult problem, we will show that simple (usually small) heap representations can be built from *structural invariants*, for a subclass of nets. To formalize this rigorously, we introduce the following definition.

DEFINITION IV.3 (HEAP REALIZATION). We say that a timed Petri net $(\mathcal{G}, M)$ with language L, has a *Heap Realization of size* k if there is a heap model $\mathcal{H}$ with k slots, such that

$$\forall w \in L, \quad y_{\mathcal{G}}(w) = y_{\mathcal{H}}(w) \quad . \quad (7)$$

That is, the execution time of the firing sequence w coincides with the height of the heap of pieces w. It is not required that $x_{\mathcal{H}} = x_{\mathcal{G}}$, which gives the potential for a smaller heap realization.

The following notions are classical, see [10, §5.1].

DEFINITION IV.4 (STATE MACHINE COVERING). A state machine is a Petri net such that each transition has exactly one predecessor and one successor. A *state machine component* of a Petri net $\mathcal{G}$ is a subnet $\mathcal{G}'$ of $\mathcal{G}$, that is a state machine, and satisfies

$${}^\bullet p \cup p^\bullet \subset \mathcal{G}' \ \text{ for every place } p \text{ of } \mathcal{G}'.$$

We say that $\mathcal{G}_1, \ldots, \mathcal{G}_k$ is a (cardinality k) *state-machine covering* of a Petri net $\mathcal{G}$, if

1. $\mathcal{G}_1, \ldots, \mathcal{G}_k$ are state-machine components of $\mathcal{G}$;
2. every arc of $\mathcal{G}$ belongs to at least one component $\mathcal{G}_i$, $1 \leq i \leq k$.

We say that the covering is *safe* (resp. *live*) if it is composed of safe (resp. live) state-machine components.

THEOREM IV.5 (REDUCED REALIZATION THEOREM). *A safe timed Petri net with a cardinality* k *safe state-machine covering admits a heap realization of size* k.

A heap realization of size k can be easily built by taking as resources the tokens of the state machine components. Formally, take the heap model $\mathcal{H} = (\mathcal{T}, \{1, \ldots, k\}, R, l, u)$, with

$$\begin{aligned} \forall a \in \mathcal{T},\ R(a) &= \{i \mid a \in \mathcal{G}_i\}\ , && (8)\\ \forall a \in \mathcal{T}, \forall i \in R(a),\ l(a,i) &= 0\ , && (9)\\ \forall a \in \mathcal{T}, \forall i \in R(a),\ u(a,i) &= \tau_a + \tau_{p(a,i)}\ , && (10) \end{aligned}$$

where $p(a,i)$ is the unique place such that the arc $(a, p(a,i))$ belongs to $\mathcal{G}_i$. A classical result (see for example [10, Th. 5.6]), which states that a live and safe free choice net admits a live and safe state machine covering, allows us to apply the reduced realization Theorem IV.5 to all live and safe free choice nets. In the special case of an event graph, state machine coverings are replaced by circuit coverings. The problem of finding a safe covering of minimal cardinality appears to be a difficult one.

Example IV.6. Let us illustrate Theorem IV.5. The timed Petri net of Fig. 2 admits a decomposition into 4 SM-components, $\mathcal{G}_i, i = 1, \ldots, 4$, with respective sets of places:

$$\mathcal{P}_1 = \{p_1, p_2\}, \mathcal{P}_2 = \{p_2, p_3, p_5\}, \mathcal{P}_3 = \{p_4\}, \mathcal{P}_4 = \{p_5, p_6\}\ .$$

Let us consider the associated heap model $\mathcal{H}$ as in Theorem IV.5. It is exactly the heap model defined in Ex. II.2 and Ex. II.6. The set of slots is $\mathcal{R} = \{1, 2, 3, 4\}$, slot i corresponding to the SM-component $\mathcal{G}_i$. The heap associated with the schedule $abcd$ was represented in Fig. 1. As a further illustration, the heaps associated with $abcdabcd$ and $abcdcdab$ are represented in Fig. 4,(1,2).

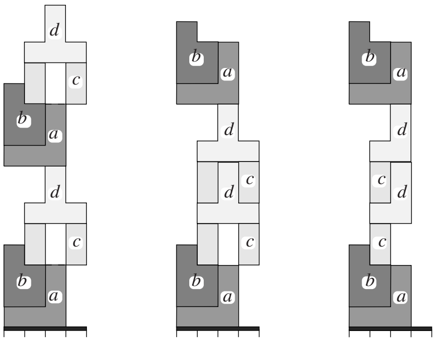


Fig. 4. (1,2): Heaps of pieces for the words $abcdabcd$ and $abcdcdab$. (3): Minimal Heap realization

It is interesting to note that the heap realization given above is not minimal. A smaller realization is shown on Fig. 4,(3). Note that this size 3 realization is not associated with a state machine covering of the net (here, the cardinality of a state machine covering is at least equal to 4).

V. An Application to the Modeling and Performance Analysis of Jobshops

The results presented above find a natural domain of application in scheduling theory.

A. An Elementary Example

The Petri net of Fig. 2 represents a manufacturing system processing two job types J_1, J_2, using two (heterogeneous) resources: one specialist S and one machine M. There are four elementary tasks a, b, c, d. The production sequence for job J_1 is ab which means that the elementary tasks a and b have to be performed in this order to complete one job J_1. The production sequence for job J_2 is cd. Let $w \in \mathcal{T}^*$. For each job type J_i, $i = 1, 2$, we set

$$\begin{aligned} |w|_{J_i} &= \text{number of type } J_i \text{ jobs completed}\\ &\quad\ \text{under the schedule } w. \end{aligned} \qquad (11)$$

Then, given an infinite schedule[6] $z = a_1 a_2 a_3 \ldots \in \mathcal{T}^\omega$, with $a_1, a_2, \ldots \in \mathcal{T}$, we may define the *asymptotic throughput* of the jobs of type J_i:

$$\lambda_i = \liminf_{n \to \infty} \frac{|a_1 \ldots a_n|_{J_i}}{\text{execution time of } a_1 \ldots a_n}\ . \qquad (12)$$

We are interested in finding the infinite schedules maximizing the throughput, under a production ratio constraint (e.g. one job J_1 for one job J_2, in the average). We restrain this problem, by requiring the schedules to be periodic. That is, one only considers periodic sequences of the form $v^\omega = vvvv\ldots$, where v is a finite production pattern satisfying the ratio constraint. In this case, as detailed below, the $\liminf$ in (12) becomes a limit (this will follow from the (max,+) linear representation, together with the (max,+) cyclicity theorem).

For instance, let us consider minimal length patterns with ratio $1/1$. There are two possible forms for such patterns: $abcd$ and $cdab$. Moreover, we note that the asymptotic performance is invariant by cyclic conjugacy of the pattern. That is, for all words u, v, $(uv)^\omega$ and $(vu)^\omega$ have the same asymptotic throughput, which follows from the identities $(uv)^\omega = u(vu)^\omega$, $(vu)^\omega = v(uv)^\omega$ (the two behaviors differ only by a finite number of tasks). Hence, there is only one behavior to consider: $L_1 = (abcd)^\omega$. One might of course consider longer patterns. E.g. periodic sequences whose pattern consists in the production of 2 jobs J_1 and 2 jobs J_2 are given by $L_2 = (abcdabcd)^\omega \cup (abcdcdab)^\omega$. We will not consider as such the schedule optimization problem (which is a difficult combinatorial one), but we will show how the heap-based modeling makes easier the *subproblem* of the performance evaluation of a given periodic schedule. This is best understood by comparison with the timed event graph modeling, that we next recall.

A.1 Illustrating the classical approach

For a given periodic schedule, one is able to build a timed event graph representing the system, and then to compute the periodic throughput of this timed event graph. Let us consider for example the schedule $(abcd)^\omega$. This functioning is represented by the timed event graph displayed in Fig. 5, which is obtained from the timed Petri net of Fig. 2 by replacing the resource places p_3 and p_4 by *circuits*, forcing the periodic sequence $abcdabcd\ldots$

Classically [2], the dynamics of a timed event graph can be represented by a $(\max,+)$-linear system: $x(n) = Ax(n-1)$, where $x(n) \in \mathbb{R}_{\max}^{\mathcal{T}}$ is a vector providing the dates of completion of the n-th firing of the transitions. The asymptotic throughput can be computed from this representation, using classical algorithms (Karp, see e.g. [8], [2]). For a schedule

[6] We denote by $\mathcal{T}^\omega$ the set of infinite words over the alphabet $\mathcal{T}$.

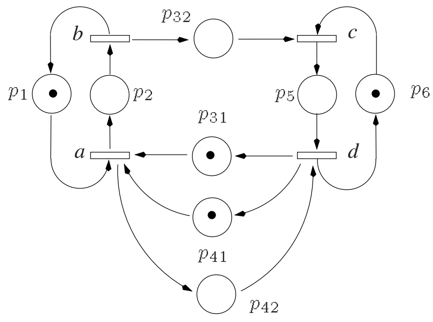


Fig. 5. Timed event graph for the schedule $(abcd)^\omega$

with a longer period, one would have to perform a similar analysis on a larger TEG (whose size grows linearly with the length of the pattern).

A.2 Illustrating the automata approach

We associate with the timed Petri net of Fig. 2 the reduced heap model and automaton given in Ex. IV.6 above.

As opposed to what was done in the classical approach, one considers a single Petri net (the original one, Fig. 2) and a single algebraic representation (the heap model). Only the order in which the products of the matrices $\mathcal{M}(u)$, $u \in \mathcal{T}$, are performed is modified from one schedule to the other.

In particular, for a periodic schedule $(v)^\omega$, the asymptotic throughput of job J_i is given by

$$\lambda_i = \frac{|v|_{J_i}}{\rho(\mathcal{M}(v))} \quad . \tag{13}$$

For instance, the matrix $\mathcal{M}(abcd)$ was given in Ex. II.6. Its eigenvalue is $\rho(\mathcal{M}(abcd)) = 8$, providing a throughput $\lambda_i = 1/8$. Similarly, one may compute the matrix $\mathcal{M}(abcdcdab)$ and obtain $\rho(\mathcal{M}(abcdcdab)) = 15$, yielding a throughput $\lambda_i = 2/15 > 1/8$. This improvement of the throughput can be visualized on the heaps of pieces of Fig. 4. More generally, one can easily check that the optimal periodic schedule having a period of length $4n$ and satisfying a 1/1 ratio constraint is v_n^ω with $v_n = ab(cd)^n(ab)^{n-1}$. It can be inferred from the heaps of Fig. 4 that the associated throughput is

$$\lambda_1 = \lambda_2 = \frac{n}{7n+1} \,,$$

so that λ_i increases to $1/7$ as $n \to +\infty$. This is of independent interest. It shows that we can always improve on the throughput by increasing the length of the pattern. Hence there exists no optimal schedule with a finite period (despite the fact that all durations are integer valued). An example of the same kind (but for a non-safe Petri net) was exhibited in Carlier & Chretienne [7], §VI-1.

The *Asymptotic Throughput Formula* (13) can be extended to general jobshop systems. It yields a much simpler way to compute the asymptotic throughput than the classical timed event graph-based method, sketched in § V-A.1. Evaluating (13) using sparse representations takes a time which is *linear* in the length of the pattern. The standard timed event graph method, implemented with the same degree of refinement, takes a time which is quadratic in the length of the pattern. This is detailed in the long version of this paper[1].

Remark V.1. It is essential to note that the heap representation coincides (up to a $90°$ rotation) with a version of the Gantt charts, where *both* the occupation of jobs and machines are represented. Traditional Gantt charts are exactly the restriction to the machine columns of heaps of pieces representations. As an illustration, we have represented below the traditional Gantt chart for the model treated at length in this paper (Ex. II.2, Ex. II.6, §V-A) and the schedule $abcdcdabab$. The added last two pieces ab seem to float in the air because one critical modelling variable is lacking (the smallest heap representation is of size three, see Fig. 4,(3)).

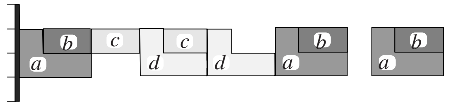


Fig. 6. (Conventional) Gantt charts are *not* (max,+) linear.

References

[1] M. Ajmone-Marsan, G. Balbo, A. Bobbio, G. Chiola, G. Conte, and A. Cumani. The effect of execution policies on the semantics and analysis of stochastic Petri nets. *IEEE Trans. on Software Engin.*, 15(7):832–846, 1989.

[2] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity*. John Wiley & Sons, New York, 1992.

[3] F. Baccelli, S. Foss, and B. Gaujal. Structural, temporal and stochastic properties of unbounded free-choice Petri nets. Technical Report 2411, INRIA, Sophia Antipolis, France, 1994. To appear in IEEE TAC.

[4] J. Berstel and C. Reutenauer. *Rational Series and their Languages*. Springer Verlag, 1988.

[5] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and Their Properties*, volume 254 of *Lect. Notes in Computer Sciences*. Springer-Verlag, 1987.

[6] M. Brilman and J.M. Vincent. Synchronisation by resources sharing : a performance analysis. Technical report, MAI-IMAG, Grenoble, France, 1995.

[7] J. Carlier and P Chretienne. Timed Petri net schedules. In *Advances in Petri Nets*, number 340 in LNCS, pages 62–84. Springer Verlag, 1988.

[8] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot. A linear system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing. *IEEE Trans. Automatic Control*, AC-30:210–220, 1985.

[9] G. Cohen, S. Gaubert, and J.P. Quadrat. Algebraic system analysis of timed Petri nets. In J. Gunawardena, editor, *Idempotency*. Cambridge University Press, 1997.

[10] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Comp. Sc.* Cambridge Univ. Press, 1995.

[11] V. Diekert and G. Rosenberg, editors. *The book of traces*. World Scientific Publ., 1995.

[12] S. Eilenberg. *Automata, languages and machines*, volume A. Academic Press, New York, 1974.

[13] S. Gaubert. Performance evaluation of (max,+) automata. *IEEE Trans. on Automatic Control*, 40(12), Dec 1995.

[14] S. Gaubert and J. Mairesse. Task resource models and (max,+) automata. In J. Gunawardena, editor, *Idempotency*. Cambridge University Press, 1997.

[15] M. Gondran and M. Minoux. *Graphes et algorithmes*. Eyrolles, Paris, 1979. Engl. transl. *Graphs and Algorithms*, Wiley, 1984.

[16] H. Hillion and J.M. Proth. Performance evaluation of job shop systems using timed event graphs. *IEEE Trans. Automatic Control*, 34(1):3–9, 1989.

[17] H. Hulgaard. *Timing Analysis and Verification of Timed Asynchronous Circuits*. PhD thesis, University of Washington, 1995.

[18] A. Mazurkiewicz. Concurrent program schemes and their interpretations. Research report DAIMI Rep. PB-78, Aarhus Univ., 1977.

[19] A. Mazurkiewicz. Trace theory. In *Petri Nets, Applications and Relationship to other Models of Concurrency*, number 255 in LNCS, pages 279–324. Springer Verlag, 1987.

[20] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[21] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets*. PhD thesis, MIT, Boston, 1974.

[22] A. Salomaa and M. Soittola. *Automata Theoretic Aspects of Formal Powers Series*. Springer Verlag, Berlin, 1978.

[23] G.X. Viennot. Heaps of pieces, I: Basic definitions and combinatorial lemmas. In Labelle and Leroux, editors, *Combinatoire Énumérative*, number 1234 in Lect. Notes in Math., pages 321–350. Springer, 1986.

[24] J.M. Vincent. Some ergodic results on stochastic iterative DEDS. Technical Report 4, Apache, IMAG, Grenoble, 1994. To appear in JDEDS.