Modeling and Analysis of Timed Petri Nets using Heaps of Pieces

Stéphane Gaubert, Jean Mairesse

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 Mazurkiewicz 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

UNTIMED and timed Petri nets have been actively studied for a long time, in particular as a modeling and analysis tool for Discrete Event Systems. Two different kinds of algebraic objects have been introduced for untimed and timed Petri nets respectively.

(1) The *untimed* behaviors of Petri nets can be represented by *languages* (sets of possible firing sequences). In the case of a net with bounded markings, the language of the net is recognized by a finite automaton, the reachability graph. (2) The *timed* behavior of a net can be represented by dater functions which provide the occurrence time of all the possible events in the system. The most satisfactory results are relative to the subclass of timed Event Graphs, which can be modeled by finite dimensional recurrent linear systems over the (max,+) semiring, see [2], [12].

There exists a striking connection between the two cases: both (max,+) linear systems and conventional automata are specializations of (max,+) automata, i.e. *automata with multiplicities* [20] over the (max,+) semiring. This observation leads to the natural question, which was left unsolved in [21]:

What is the modeling power of (max,+) automata in terms of timed Petri nets?

The purpose of this paper is to propose the following answer:

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

This work was partially supported by the European Community Framework IV programme through the research network ALAPEDES ("The ALgebraic Approach to Performance Evaluation of Discrete Event Systems")

S. Gaubert is with INRIA, Domaine de Voluceau, B.P. 105, 78153 Le Chesnay Cedex, France. E-mail: Stephane.Gaubert@inria.fr

J. Mairesse is with CNRS, LIAFA, Université Paris 7, Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France. E-mail: mairesse@liafa.jussieu.fr

As a by-product, we will obtain new automata-based performance evaluation algorithms.

This representation theorem is best understood by comparison with the following existing approaches.

1. *Trace languages.* In the landmark paper [31], Mazurkiewicz observed that *trace monoids* (that is, free partially commutative monoids) and their subsets (trace languages) are a natural model of the logical behavior of safe Petri nets. Trace monoids, in which certain letters commute, and others do not, allow one to identify the different sequential representations of the same concurrent events.

2. *Heaps of pieces.* In [40], Viennot observed that trace monoids are isomorphic to *heap monoids*, that is, monoids in which the generators are *pieces* (in the nearly usual understanding: these pieces are solid rectangular-shaped blocks), and where the concatenation consists in piling up one *heap* above the other. This yields a very intuitive graphical representation of trace monoids.

3. *(Max,+) linear representations.* A next step was the observation that the height of heaps of pieces is recognized by a *heap automaton*, a special type of (max,+) automaton, the result holding for general, polyomino-shaped, pieces. This was proved in Gaubert and Mairesse [22], and in a different form, by Brilman and Vincent [41], [7].

The heap representation theorem for safe timed Petri nets that we give can be seen as a synthesis of these three results. The essential idea is that considering general pieces in heap automata enables to model time through the height of a piece.

More precisely, one of the main results of the paper goes as follows. Let $\mathcal{G}$ be a safe timed Petri net, with set of transitions $\mathcal{T}$ and set of firing sequences $L \subset \mathcal{T}^*$. Then the map $y_{\mathcal{G}} : L \to \mathbb{R}^+$, which associates to a firing sequence the date of completion of the last event in the net, is recognized by a heap automaton (less formally, a piece is associated to each letter in $\mathcal{T}$ and $y_{\mathcal{G}}(w)$ is the height of the heap obtained by piling up the pieces corresponding to the letters of w).

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). With each periodic schedule, the classical modeling associates an event graph (i.e. a (max,+) linear system) whose size grows with the period of the schedule, see [12], [27], [2]. On the other hand, the representation by heap model is independent of the (even non periodic) schedule which is considered. This is par-

ticularly 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, which is simpler than the refined variants of the traditional (event graph based) method.

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 the related independent work of Hulgaard [28], who studied the subclass of safe Free Choice Petri nets. He did not put forward the automata or heap models, but he did introduce (for time analysis purposes) dater variables and dynamical equations similar to the ones used here, see the discussion in Remark IV.8 below.

A very different algebraic approach is that of Baccelli, Foss and Gaujal [3]; Cohen, Gaubert and Quadrat [13], [14]; and Libeaut and Loiseau, see [29, Chap. 2]. Essentially, the counter function (vector of numbers of firings, as a function of time) of general (non-necessarily safe) Free Choice or Fluid Petri nets satisfies some combination of implicit (min,+) and (+,×)-linear dynamical equations, which become explicit under certain assumptions on the timing or routing policy. In a certain sense, this approach, which computes the number of firings (logical time) as a function of the physical time, is dual, or inverse, to the automata approach proposed here, which computes the dates (physical times) as a function of the schedule (logical time). Surprisingly, these two points of view lead to completely different technical developments.

The paper is organized as follows. In § II, we introduce heap models and (max,+) automata. In § III, we recall basic facts about Petri nets and their representation by trace monoids. The main result, "safe timed Petri nets can be represented by heaps of pieces", is proved in § IV-A.

In this representation, the size of the heap model is equal to the number of places in the Petri net. In § IV-B, we show the existence of a much smaller heap representation for the subclass of nets which can be covered by safe State Machine Components. It includes in particular all safe Free Choice Petri nets.

In § V, we apply these results to the typical example of jobshops, proposing a new performance evaluation algorithm. In § VI, we discuss at a more algebraic level the models used in the paper and their interplay.

To conclude, let us mention related general references. The reader is referred to [2], [15], [42], [30], [25], [23], for an account of the theory of the (max,+) semiring. The theory of automata with multiplicities is dealt with in [4], [20], [37]. The (max,+) automata used in this paper are generalizations of "cost automata", or automata with multiplicities over the *tropical* semiring $(\mathbb{N} \cup \{+\infty\}, \min, +)$. They have been widely studied for their connections with classical decidability problems in language theory (see [35], [38], and the references therein). A Discrete Event Systems oriented presentation can be found in [21]. General accounts of Petri net theory can be found in [5], [33], [16] or in the proceedings [6].

II. Heap Models and (max,+) Automata

The following heap model generalizes the heaps of pieces of Viennot [40].

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[1] 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*.
- $R : \mathcal{T} \to \mathfrak{P}(\mathcal{R})$ gives the subset of slots occupied by a piece. We assume that each piece occupies at least one 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[2] $w = a_1, \ldots, a_k \in \mathcal{T}^*$ as a heap, i.e. as a sequence of k pieces $a_1, \ldots, a_k$ 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 $x_{\mathcal{H}}(e) = (0, \cdots, 0)$ (recall that e denotes the empty word). 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 [22], [7].

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

[2] 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 by e. The length of the word w will be denoted by $|w|$. We shall write $|w|_a$ for the number of occurrences of a given letter a in w.

Borrowing the terminology of [2], 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.

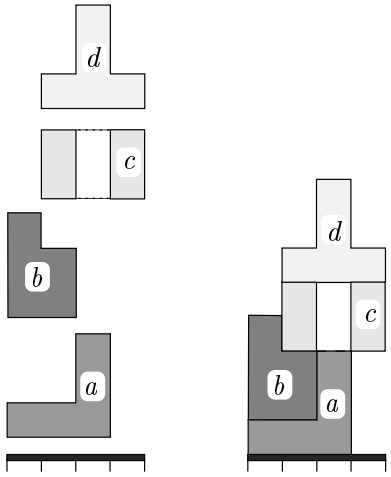


Fig. 1. Heap of pieces associated with the word $w = abcd$.

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[3] $\mathbb{R}_{\max}$ 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 $\mathbb{1}$.

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

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

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

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

Hence a (max,+) automaton may be seen as a (max,+) linear system whose dynamics is driven by letters.

With a heap model $\mathcal{H} = (\mathcal{T}, \mathcal{R}, R, l, u)$, we associate the morphism $\mathcal{M} : \mathcal{T}^* \to \mathbb{R}_{\max}^{\mathcal{R} \times \mathcal{R}}$, defined by

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

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 $y_{\mathcal{H}}$:

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

A variant of this result was proved in [22], see also [7]. We say that $(\mathcal{R}, \mathbb{1}_{\mathcal{R}}^t, \mathbb{1}_{\mathcal{R}}, \mathcal{M})$ is the heap automaton associated with the heap model. For the sake of completeness, we give an abridged version of the argument.

Proof: The following dynamical equation should be clear from the physical description of the system:

$$x_{\mathcal{H}}(wa)_r = \begin{cases} x_{\mathcal{H}}(w)_r \text{ if } r \notin R(a) \\ \max_{s \in R(a)} (\, x_{\mathcal{H}}(w)_s + u(a, r) - l(a, s) \,) \\ \qquad \text{if } r \in R(a). \end{cases} \tag{2}$$

For example, let us consider the case of a piece a with an horizontal base ($l(a, s) = 0, s \in R(a)$). When adding piece a to a heap w, one has first to compute the height of the base of the piece which is equal to $\max_{s \in R(a)} x_{\mathcal{H}}(w)_s$. Then

[3] A set K equipped with two operations $\oplus$ and $\otimes$ is a semiring if $\oplus$ is associative and commutative, $\otimes$ is associative and distributive with respect to $\oplus$, there is a zero element $\mathbb{0}$ ($a \oplus \mathbb{0} = a, a \otimes \mathbb{0} = \mathbb{0} \otimes a = \mathbb{0}$) and a unit element $\mathbb{1}$ ($a \otimes \mathbb{1} = \mathbb{1} \otimes a = a$).

[4] This is the specialization to the (max,+) semiring of the classical notion of automaton with multiplicity [20], or recognizable series [37], [4]. For more details along these lines, see § VI.

[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.

the height of the heap wa on slot $r \in R(a)$ is obtained as $\max_{s\in R(a)} x_{\mathcal{H}}(w)_s + u(a,r)$. Clearly, we have:

$$\begin{array}{lll} \forall r \in \mathcal{R}, \quad x_{\mathcal{H}}(e)_r & = & \mathbb{1} \ , \\ \qquad\qquad y_{\mathcal{H}}(w) & = & \max_r x_{\mathcal{H}}(w)_r = x_{\mathcal{H}}(w)\mathbb{1}_{\mathcal{R}} \ . \end{array} \tag{3}$$

We identify in (2) and (3) a dynamics of the form (1). ■
We mention for further use the following elementary commutation property, which holds for all $a, b \in \mathcal{T}$:

$$R(a) \cap R(b) = \emptyset \implies \mathcal{M}(a)\mathcal{M}(b) = \mathcal{M}(b)\mathcal{M}(a) \ . \tag{4}$$

An alternative "dual" automaton representing the heap model, obtained by associating dater variables to pieces, instead of slots, was given in [22].

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

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

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

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

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

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

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

Remark II.7: Heap automata, as introduced in Theorem II.5, are (max,+) automata of a specific form. The morphism $\mathcal{M}$ of a heap automaton verifies:

$$\mathcal{M}(a) = I \oplus [\tilde{l}(a,.)]^t u(a,.) \ ,$$

where I is the identity matrix defined by $I_{ii} = \mathbb{1}, I_{ij} = \mathbb{0}, i \neq j$, where $\tilde{l}(a,i) = -l(a,i)$ if $l(a,i) \neq \mathbb{0}$ and $\tilde{l}(a,i) = l(a,i) = \mathbb{0}$ otherwise, and where $\tilde{l}(a,.), u(a,.)$ are viewed as row vectors.

III. Untimed and Timed Petri Nets

A. Definitions

We next recall some classical facts about untimed and timed Petri nets.

Definition III.1: A Petri net (PN) is a 4-tuple $\mathcal{G} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$, where :

- $\mathcal{P}$ is a finite set, whose elements are called *places*.
- $\mathcal{T}$ is a finite set, whose elements are called *transitions*.
- $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$ is a relation between places and transitions.
- M is a map $\mathcal{P} \to \mathbb{N}$. The integer $M(p)$ is called the *initial marking* of place p.

We will use the term Petri net to denote both the unmarked and the marked net. We will sometime denote the marked net by $(\mathcal{G}, M)$ instead of $\mathcal{G}$ to insist on the value of the initial marking.

A Petri net is traditionally represented as a bipartite directed graph. There are two different kinds of nodes, places $p \in \mathcal{P}$, (represented by circles) and transitions $a \in \mathcal{T}$ (represented by rectangles). An element of $\mathcal{F}$ is an arc from a place to a transition or from a transition to a place. It is therefore natural to speak of "input places", "output transitions" and so on. We use the notations ${}^\bullet p, p^\bullet$ (resp. ${}^\bullet a, a^\bullet$) for the set of input and output transitions of place p (resp. input and output places of transition a). The marking $M(p)$ is displayed by drawing $M(p)$ tokens in place p.

We will use the classical notions of (oriented) *path*, *circuit*, *connectedness* and *strong-connectedness* of graph theory. An example of a strongly connected Petri net is provided in Fig. 2.

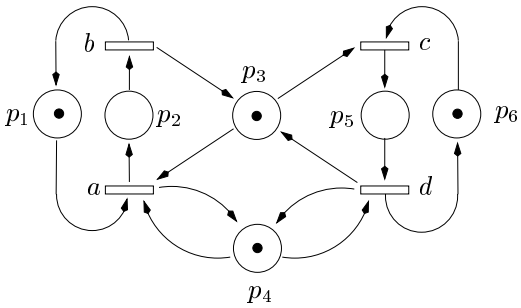


Fig. 2. Strongly connected safe Petri net

A Petri net is a dynamic object. The underlying structure $(\mathcal{P}, \mathcal{T}, \mathcal{F})$ is never modified, but the marking M evolves according to the following *firing rule*.

1. Transition a is *enabled* at M if there is at least one token in each of its input places.
2. An enabled transition a can *fire*. The firing of a transforms M into M' (written $M \xrightarrow{a} M'$) by removing one token from each of the input places and adding one token in each of the output places of a.

We say that a word $w = a_1 a_2 \cdots a_n \in \mathcal{T}^*$ is a *firing sequence* starting from marking M' if there is a sequence of markings $M' = M_0, M_1, \ldots, M_n = M''$ such that transition a_i is enabled at M_{i-1} and $M_{i-1} \xrightarrow{a_i} M_i$. We abbreviate this by $M' \xrightarrow{w} M''$. A marking M'' is *reachable* from a marking M' if there is a firing sequence $w \in \mathcal{T}^*$ such that $M' \xrightarrow{w} M''$. We denote by $\mathsf{R}(M')$ the set of markings reachable from M'.

We call *language*[6] of the Petri net $(\mathcal{G}, M)$ the set $L \subset \mathcal{T}^*$ of firing sequences starting from M.

[6]More properly, the **P**-type free labeled language, according to the terminology of Peterson [34]. Various kinds of Petri net languages have been defined (according to various kinds of acceptance conditions

Definition III.2: A Petri net $(\mathcal{G}, M)$ with language L is:
- *live* if $\forall w \in L, \forall t \in \mathcal{T}$, $\exists u \in \mathcal{T}^*$, such that $wut \in L$, i.e. if whatever the past firings ($= w$) are, it is possible to find a firing sequence from the current state, containing transition t;
- *bounded* if the set $\mathsf{R}(M)$ is finite. Equivalently, if $\exists k$ such that $\forall M' \in \mathsf{R}(M), \forall p \in \mathcal{P},\ M'(p) \leq k$;
- *safe* (or 1-bounded) if a place will not hold more than one token: $\forall M' \in \mathsf{R}(M), \forall p \in \mathcal{P},\ M'(p) \leq 1$.

Example III.3: The Petri net represented in Fig. 2 is live and safe. Its language is $L = (ab \cup cd)^*(e \cup a \cup c)$.

Let us recall some classical subclasses of Petri nets.
1. A *circuit* is a PN such that $|{}^\bullet t| = |t^\bullet| = |{}^\bullet p| = |p^\bullet| = 1$ for all $t \in \mathcal{T}, p \in \mathcal{P}$.
2. A *State Machine* (SM) is a PN such that $|{}^\bullet t| = |t^\bullet| = 1$ for all $t \in \mathcal{T}$.
3. An *Event Graph* (EG) is a PN such that $|{}^\bullet p| = |p^\bullet| = 1$ for all $p \in \mathcal{P}$.
4. A *Free Choice net* (FC) is a PN such that $p^\bullet \cap q^\bullet \neq \emptyset \Rightarrow p^\bullet = q^\bullet$ for all $p, q \in \mathcal{P}$.

SM are also known as *S-systems* and EG as *marked graphs*, *decision-free Petri nets* or *T-systems*. Our definition of FC corresponds to what is often called *extended Free Choice nets* in the literature. It follows from the definitions that $EG \subset FC$ and $SM \subset FC$.

In Fig. 3, we illustrate the basic notions of concurrency, choice (or decision) and synchronization. In case (I), transitions a and b are concurrently enabled, i.e. they can fire independently. In case (II), we say that there is a choice between transitions a and b or that a and b are in conflict. In case (III), there is a synchronization at transition a. Among Petri nets, SM allow choice but not synchronization whereas EG allow synchronization but not choice. FC are the natural generalizations of both SM and EG.

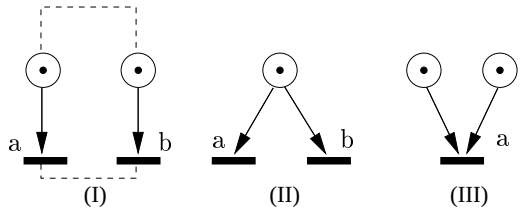


Fig. 3. (I) Concurrency. (II) Choice. (III) Synchronization.

B. Execution semantics and traces

Let us consider a Petri net where n transitions, a_1 to a_n, are concurrently enabled, see Fig. 3,(I). We assume that all of them have to fire before any new transition becomes enabled. Then, the same behavior, i.e. the firing of the n transitions, can be described by any of the following $n!$ firing sequences: $a_{\sigma(1)}, \cdots, a_{\sigma(n)}$ where σ is a permutation of $\{1, \ldots, n\}$.

This simple example shows that firing sequences, which provide a sequential description of the behavior, are not really adapted in the presence of concurrency.

on final markings and the different labeling functions that one may consider).

The problem of modeling concurrency in a more efficient way has long been considered. A classical approach, proposed by Mazurkiewicz [31], [32], uses the notion of trace monoid. For a general and recent reference on traces, see [18].

Definition III.4: Let $\mathcal{T}$ be an alphabet equipped with a reflexive symmetric relation called *dependence relation* and denoted by $\mathcal{D}$. We denote by $\mathcal{I}$ the complement of $\mathcal{D}$, called *independence* relation. The *trace monoid* $\mathcal{T}^*/_\sim$ is the quotient of the free monoid $\mathcal{T}^*$ by the least congruence $\sim$ containing the relations $ab \sim ba$, $\forall (a,b) \in \mathcal{I}$. The elements of $\mathcal{T}^*/_\sim$ will be called *traces*.
Two words are representatives of the same trace if they can be obtained one from the other by repeatedly interchanging adjacent independent letters. Indeed, one can easily show that the reflexive and transitive closure of the relation $uabv \equiv ubav$, $\forall (a,b) \in \mathcal{I}, u, v \in \mathcal{T}^*$ is compatible with the monoid structure of $\mathcal{T}^*$. Hence, this reflexive and transitive closure is precisely the trace relation $\sim$. See [18, Chap. 1,§ 1.3] for details.

Given a Petri net $(\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$, we define the following independence relation:

$$\mathcal{I} = \{(a,b) \in \mathcal{T}^2 \mid ({}^\bullet a \cup a^\bullet) \cap ({}^\bullet b \cup b^\bullet) = \emptyset\}\ , \qquad (5)$$

and the associated trace monoid $\mathcal{T}^*/_\sim$. Two transitions are independent iff they do not share input or output places.

For safe Petri nets[7], the congruence $\sim$ generated by $\mathcal{I}$ identifies firing sequences which differ only by the sequential ordering of concurrent events. In particular, if $w_1 \sim w_2$ then $w_1 \in L \Leftrightarrow w_2 \in L$ (where L is the language of the Petri net). A trace whose representatives are firing sequences is called a *firing trace*. The set of firing traces is denoted by $(L/_\sim)$.

It is very convenient to visualize trace monoids using heap models, as it was originally proposed by Viennot in [40]. Let us detail this for the trace monoid $\mathcal{T}^*/_\sim$ associated with a safe Petri net $(\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$. Consider the heap model $\mathcal{H}(\mathcal{T}^*/_\sim)$ with:
- set of pieces $\mathcal{T}$;
- set of slots $\mathcal{P}$;
- $R(a) = {}^\bullet a \cup a^\bullet$, $a \in \mathcal{T}$;
- $l(a,r) = 0, u(a,r) = 1, r \in R(a)$.

Two words $w_1, w_2 \in \mathcal{T}^*$ are equivalent ($w_1 \sim w_2$) if and only if they provide the same heap. In the heap associated with $w \in L$, the pieces associated with one level (i.e. the pieces occupying a common vertical position) correspond to the events (i.e. the transition firings) occurring concurrently.

Example III.5: Let us consider the Petri net of Fig. 2. We have represented in Fig. 4 the heap of pieces associated with the firing sequence *abcdabcd*. Note that there is no concurrency at all in this Petri net. It is purely sequential and the independence relation $\mathcal{I}$ defined in (5) is empty.

[7] In a non-safe Petri net, two transitions a, b such that ${}^\bullet a \cap {}^\bullet b \neq \emptyset$ can be concurrently enabled if their common input places hold more than one token. It is possible to introduce another dependence relation to take care of this, see Diekert [17]. Such refinements are not needed here as we consider only safe Petri nets.

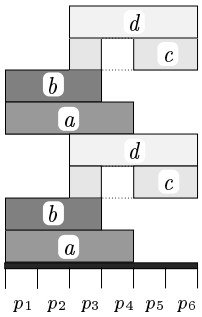


Fig. 4. Heap of pieces for the word $w = abcdabcd$.

C. 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 [36] for this subclass.

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. First, a may never fire. Second, in order to fire, transition a needs to get disabled in a first time (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), then re-enabled later on.

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 on 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.6: Let us consider the Petri net of Fig. 2. We associate with its transitions, the following firing times:

$$\tau_a = 1, \tau_b = 2, \tau_c = 2, \tau_d = 1 \, .$$

We associate with the places the holding times:

$$\tau_1 = 1, \tau_2 = 0, \tau_3 = 0, \tau_4 = 2, \tau_5 = 0, \tau_6 = 0 \, .$$

Let us assume that the initial marking is the one shown in Fig. 2. 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.

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 or *schedule* $w = a_1 \dots a_k \in L$, we start at time 0, and fire the transitions $a_1, \dots, a_k$ in this order, applying the earliest firing semantic described in § III-C above.

With each place p, and for $w \in L$, 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

$$x_{\mathcal{G}}(w)_p = \begin{cases} z(w)_p & \text{if } M \xrightarrow{w} M', \text{ with } M'(p) = 1 \\ z'(w)_p & \text{if } M \xrightarrow{w} M', \text{ with } M'(p) = 0 \end{cases} \tag{6}$$

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 function* of the Petri net. The *makespan* or *execution time* of the firing sequence w is naturally defined by

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

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 with language L. Then, the heap model $\mathcal{H} = (\mathcal{T}, \mathcal{P}, R, l, u)$, with

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

is such that

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

Equation (7) states that the dater vector of the net coincides with the upper contour of the associated heap.

Let us consider the heap model obtained from $\mathcal{H}$ by replacing u and l by $\hat{u}(a) = 1, a \in R(a)$ and $\hat{l}(a) = 0, a \in R(a)$, respectively. This is precisely the heap model associated with the trace monoid $\mathcal{T}^*/_\sim$ of the Petri net, i.e. $\mathcal{H}(\mathcal{T}^*/_\sim)$, see § III-B. The pieces of the heap model $\mathcal{H}$ are obtained by a deformation of the pieces of the heap model $\mathcal{H}(\mathcal{T}^*/_\sim)$, incorporating the timing information.

Proof of Theorem IV.1: Let us consider $w \in L, a \in \mathcal{T}$, such that $wa \in L$. We have,

$$x_{\mathcal{G}}(wa)_p = \begin{cases} x_{\mathcal{G}}(w)_p & \text{if } p \notin a^\bullet \cup {}^\bullet a \\ \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'} + \tau_a + \tau_p & \text{if } p \in a^\bullet \\ \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'} & \text{if } p \in {}^\bullet a \setminus a^\bullet. \end{cases} \tag{8}$$

Indeed, the firing of transition a after w is initiated at instant

$$T = \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'}\ ,$$

from which (8) follows. The dynamics (8) would coincide verbatim with the one of the heap model $\mathcal{H}$ given above (see (2)), if the term $\max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'}$ was replaced by $\max_{p' \in {}^\bullet a \cup a^\bullet} x_{\mathcal{G}}(w)_{p'}$. Hence, it remains to check that

$$\max_{p' \in {}^\bullet a \cup a^\bullet} x_{\mathcal{G}}(w)_{p'} = \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'}\ ,$$

or equivalently

$$\forall p'' \in a^\bullet \setminus {}^\bullet a, \qquad x_{\mathcal{G}}(w)_{p''} \leq \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'}\ .$$

Since $p'' \in a^\bullet \setminus {}^\bullet a$, the firing of a at time T adds one token to the marking of p''. Since the net is safe, there is at most one token in each place, for any logically admissible execution of the system. We conclude that the exit time of the last token in place p'' under the firing sequence w must be strictly less than T. Using the defining relation (6), $x_{\mathcal{G}}(w)_{p''}$ is equal to the last instant of presence of a token in p'', under w. We conclude that $x_{\mathcal{G}}(w)_{p''} \leq T = \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'}$. ■

Due to the commutation in (4), the heap associated with $w \in \mathcal{T}^*$, and a fortiori $x_{\mathcal{H}}(w)$ and $y_{\mathcal{H}}(w)$, depend only on the equivalence class of w in $\mathcal{T}^*/\sim$. Although this heap, $x_{\mathcal{H}}(w)$, and $y_{\mathcal{H}}(w)$ are defined for all $w \in \mathcal{T}^*$, they have no meaning in terms of the Petri net if $w \notin L$.

Example IV.2: Let us illustrate the previous construction with the Petri net $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M)$ represented in Fig. 2 and the numerical values of Ex. III.6.

The heap model associated with $\mathcal{G}$ 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. 5. 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. This heap is a deformation of the one of Fig. 4. 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).

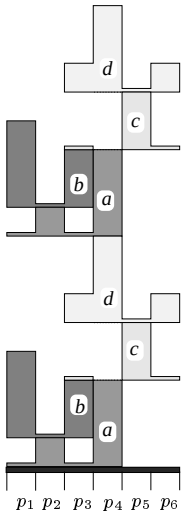


Fig. 5. Heap of pieces associated with the word $w = abcdabcd$.

B. The minimal realization problem

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

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

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 [16, §5.1].

Definition IV.4 (State Machine Covering) 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.
It follows from this definition that a state machine component is uniquely defined by its set of places. Note that a safe (resp. live) SM is a SM with at most (resp. at least) one token.

Theorem IV.5 (Reduced Realization Theorem) Let $\mathcal{G} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M, \tau)$ be a safe timed Petri net with language L and having a safe state machine covering $(\mathcal{G}_1, \ldots, \mathcal{G}_k)$. Then, $\mathcal{G}$ admits a heap realization of size k given by the heap model $\mathcal{H} = (\mathcal{T}, \{1, \ldots, k\}, R, l, u)$, where

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

where $p(a,i)$ is the unique place such that $(a, p(a,i)) \in \mathcal{G}_i$.

Proof: We first prove the result when the state-machine components are not only safe but live. Then, there is exactly one token at any time in each state-machine component and we may speak unambiguously of *the token* in $\mathcal{G}_i$. With this token, we associate a dater function $\tilde{z}_i$. If w is a firing sequence, we denote by $\tilde{z}(w)_i$ the time at which token i becomes available in its current place $p \in \mathcal{G}_i$, after the firing of the last transition $a \in w$ such that $p \in a^\bullet$ (it is the "completion time" of schedule w for the token i). It is clear that $\max_i \tilde{z}(w)_i = y_{\mathcal{G}}(w)$. Hence, it is enough to prove that $\tilde{z}(w) = x_{\mathcal{G}}(w)$, for $w \in L$. We set $\tilde{z}(e)_i = 0$ (recall that e denotes the empty word). Clearly, $\tilde{z}(e) = x_{\mathcal{H}}(e)$. Let us consider $w \in L, a \in \mathcal{T}$ such that $wa \in L$. We have

$$\tilde{z}(wa)_i = \begin{cases} \tilde{z}(w)_i & \text{if } i \notin R(a) \\ \max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'} + \tau_a + \tau_p & \text{if } i \in R(a) \end{cases} . \tag{9}$$

We recall that $p(a,i)$ is the unique place such that $(a, p(a,i)) \in \mathcal{G}_i$. For all places $p' \in {}^\bullet a$, there exists at least one state machine component $\mathcal{G}_j$ such that $(p', a) \in \mathcal{G}_j$. Thus,

$$\max_{p' \in {}^\bullet a} x_{\mathcal{G}}(w)_{p'} = \max_{j:\ \exists p'', (p'', a) \in \mathcal{G}_j} \tilde{z}_j(w)\ . \tag{10}$$

Arguing as at the end of the proof of Theorem IV.1 (using the safe character of the net), we get that

$$\begin{aligned} \max_{j: \exists p'', (p'', a) \in \mathcal{G}_j} \tilde{z}(w)_j &= \max_{j:\ \exists p'', (p'', a) \in \mathcal{G}_j \text{ or } (a, p'') \in \mathcal{G}_j} \tilde{z}(w) \\ &= \max_{j \in R(a)} \tilde{z}(w)_j\ . \end{aligned} \tag{11}$$

Substituting (10) in (9), and using (11), we get that $\tilde{z}$ satisfies precisely the dynamical equations (2) of the Heap model $\mathcal{H}$. This concludes the proof of the theorem, when the state-machine covering is live.

For a general safe but not-live covering, there is either 0 or 1 token in each state-machine component. All the transitions within unmarked state-machine components will never fire. It is now immediate to adapt the above argument, setting $\tilde{z}(w)_i = -\infty$, for any unmarked state-machine component $\mathcal{G}_i$. ■

Example IV.6: Let us illustrate Theorem IV.5. We consider the Petri net $\mathcal{G} = (\mathcal{T}, \mathcal{P}, \mathcal{F}, M)$ of Fig. 2, with the timings defined in Ex. III.6.

This net 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. 6,(1,2).

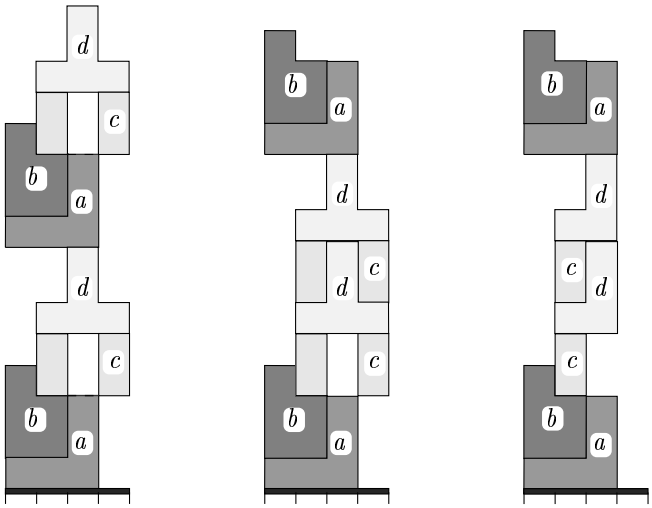


Fig. 6. (1,2): Heaps of pieces for the words *abcdabcd* and *abcdcdab*. (3): Minimal Heap realization

It is interesting to note that the heap realization given above is not minimal. A smaller realization is shown on Fig. 6,(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). This shows that Theorem IV.5 provides only a partial answer to the minimal realization problem.

The next classical result (see for example [16, Th. 5.6]), shows that the reduced realization of Theorem IV.5 can be applied to all live and safe Free Choice nets.

Proposition IV.7: A live and safe Free Choice net admits a live and safe state machine covering.

In the case of an event graph, the same result applies when replacing state machine coverings by circuit coverings. The problem of finding the covering of minimal cardinality in Prop. IV.7 (or, in general, for Petri nets admitting such coverings) appears to be a difficult one.

Remark IV.8: In his thesis, Hulgaard proposed a similar approach for safe FC [28, Chap. 7]. He defined an analogue of the vector $x_{\mathcal{G}}(w)$ and of the matrices $\mathcal{M}(a)$, and derived dynamical equations similar to (9). The main difference is that he uses rectangular matrices which depend not only on the transition to fire but also on the current marking.

C. Heap realization and $\mathcal{P}$-semiflow

There is a close connection between the size of the heap realization in Theorem IV.5 and a classical invariant, the $\mathcal{P}$-semiflow. A $\mathcal{P}$-semiflow is a column vector $x \in \mathbb{N}^{\mathcal{P}}$ such that $\sum_{p\in{}^\bullet a} x(p) = \sum_{p\in a^\bullet} x(p)$, $\forall a \in \mathcal{T}$. That is, the weighted marking (usual algebra) $\sum_p x(p)M(p)$ of the places is invariant by the firing of a transition.

Let $\mathcal{G}$ be a safe timed Petri net admitting a SM covering $\{\mathcal{G}_1, \ldots, \mathcal{G}_k\}$. We consider the strictly positive vector $x = x_{\mathcal{G}_1} + \cdots + x_{\mathcal{G}_k}$, where $x_{\mathcal{G}_i} \in \mathbb{R}^{\mathcal{P}}_{\max}$ is the characteristic vector of $\mathcal{G}_i$, defined by:

$$x_{\mathcal{G}_i}(p) = \begin{cases} 1 & \text{if } p \in \mathcal{G}_i \\ 0 & \text{otherwise} \end{cases} .$$

The vector x is a $\mathcal{P}$-semiflow. The invariant $\sum_p x(p)M(p)$ is equal to k times the size of the heap realization of $\mathcal{G}$. However, it is not true that each $\mathcal{P}$-semiflow of a Petri net can be represented as the sum $x_{\mathcal{G}_1} + \cdots + x_{\mathcal{G}_l}$ of the characteristic vectors of a SM covering. A fortiori, the problem of finding a safe SM covering of minimal cardinality can not be reduced to the classical problem of finding a minimal $\mathcal{P}$-semiflow.

We define the *expansion* $\tilde{\mathcal{G}}$ of the Petri net $\mathcal{G}$ with respect to x as follows:

Each place $p \in \mathcal{P}$ such that $x(p) > 1$ is replaced by $x(p)$ places. Each of these $x(p)$ places have the same input and output transitions as the original place p. They also have the same number of tokens and the same holding time as p.

The firing sequences and the temporal behaviors of $\tilde{\mathcal{G}}$ and $\mathcal{G}$ are exactly the same (given a firing sequence w, the firing instants of the transitions are the same). Furthermore, the graph $\tilde{\mathcal{G}}$ admits a SM partition[8] $(\tilde{\mathcal{G}}_1, \ldots, \tilde{\mathcal{G}}_k)$. This is best understood with an example.

Example IV.9: Let us consider the Petri net of Fig. 2. A state machine covering of this Petri net was provided in Example IV.6. The associated characteristic vector is $x = (1, 2, 1, 1, 2, 1)$. Hence, we have to duplicate p_2 and p_5. We have represented the expanded Petri net with its SM-partition in Fig. 7.

The reduced heap automaton $\mathcal{H}$ associated with $\tilde{\mathcal{G}}$ is the same as the one associated with $\mathcal{G}$. Now, the number of tokens of $\tilde{\mathcal{G}}$ is constant and there is a simple interpretation for $(x_{\mathcal{H}})_i$ (or $(x_{\tilde{\mathcal{H}}})_i$): it is the dater function of the token of $\tilde{\mathcal{G}}_i$.

[8]Same definition as a SM covering except that each arc belongs to exactly one SM component.

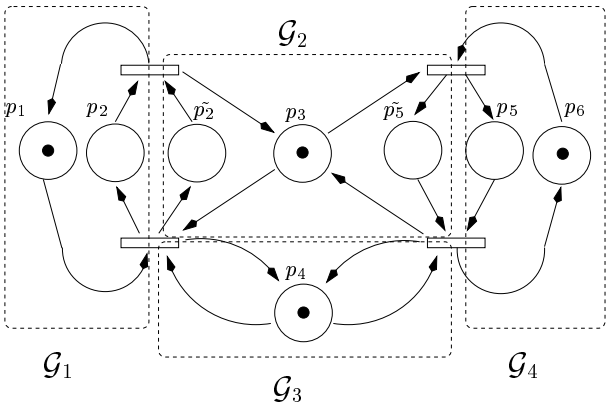


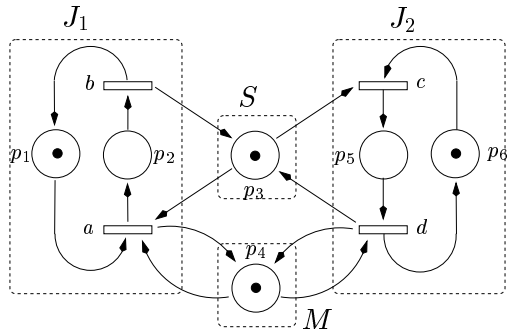
Fig. 7. Expansion of the Petri net of Fig. 2.

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 good introduction to the subject is provided by the books [8], [10]. We first show how heap representations can be used to design performance evaluation methods. We explain informally the method on a small manufacturing model, and we compare it with the classical approach. Then, we consider the general subclass of safe jobshops. We describe the classical performance evaluation algorithm and a new heap automata based one, and we derive complexity bounds for both of them.

A. An Elementary example

Let us consider the Petri net of Fig. 8, that the reader certainly recognizes as being the one discussed extensively above.



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. 8. A two jobs — two resources manufacturing system

This Petri net can be interpreted as 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

$$|w|_{J_i} = \text{number of type } J_i \text{ jobs completed under the schedule } w. \quad (12)$$

Then, given an infinite schedule[9] $z = a_1 a_2 a_3 \ldots \in \mathcal{T}^\omega$, with $a_1, a_2, \ldots \in \mathcal{T}$, we define the *asymptotic throughput* of the jobs of type J_i:

$$\lambda_i = \liminf_{n\to\infty} \frac{|a_1 \ldots a_n|_{J_i}}{\text{execution time of } a_1 \ldots a_n} \ . \quad (13)$$

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 lim inf in (13) 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:

$$L_2 = (abcdabcd)^\omega \cup (abcdcdab)^\omega \ .$$

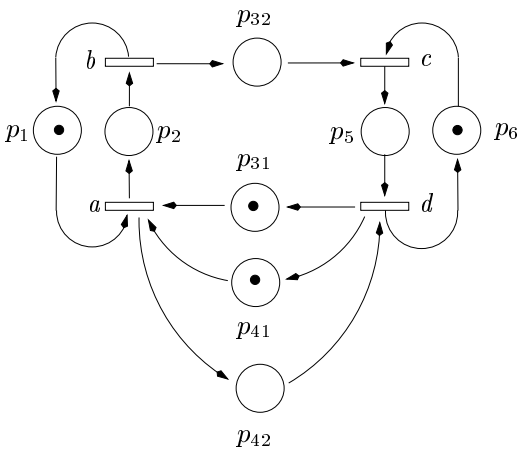
We will not consider as such the schedule optimization problem (which is a difficult combinatorial one), but we will show how the heap-based modeling makes easier the *subproblem* of the performance evaluation of a given periodic schedule. This is best understood by comparison with the timed Event Graph 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. 9, 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 $abcdabcd\ldots$

Let $x(n) \in \mathbb{R}_{\max}^{\mathcal{T}}$ be the vector providing the dates of the completion of the n-th firing of the transitions. The vector

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



Holding and Firing Times:

a	b	c	d	p_1	p_2	p_{31}	p_{32}	p_{41}	p_{42}	p_5	p_6
1	2	2	1	1	0	0	0	2	2	0	0

Fig. 9. Timed Event Graph for the schedule $(abcd)^\omega$

$x(n)$ evolves according to a (max,+) linear dynamic:

$$\begin{aligned} x_a(n) &= \tau_a(\tau_1 x_b(n-1) \oplus (\tau_{31} \oplus \tau_{41}) x_d(n-1)) \\ x_b(n) &= \tau_b \tau_2 x_a(n) \\ x_c(n) &= \tau_c(\tau_{32} x_b(n) \oplus \tau_6 x_d(n-1)) \\ x_d(n) &= \tau_d(\tau_5 x_c(n) \oplus \tau_{42} x_a(n)) \ . \end{aligned}$$

Setting $\xi(n) = [x_b(n), x_d(n)]^t$, eliminating x_a and x_c, and taking the numerical values of Fig. 9, we obtain the subsystem $\xi(n) = \overline{A}\xi(n-1)$, with

$$\overline{A} = \begin{bmatrix} 4 & 5 \\ 7 & 8 \end{bmatrix} \ , \quad \rho(\overline{A}) = 8 \ ,$$

where $\rho(\overline{A})$ denotes the (unique) (max,+) eigenvalue of the (irreducible) matrix $\overline{A}$ (see [2], [12], particularly the introductive section §1.3 of [2], for details on the (max,+) spectral theory and its applications to discrete event systems). It follows from the cyclicity theorem in [2] that for $i = 1, 2$, and $u \in \mathcal{T}$, the asymptotic throughputs are

$$\lambda_i = \lim_{n\to\infty} \frac{n}{x(n)_u} = \frac{1}{\rho(\overline{A})} = \frac{1}{8} \ . \quad (14)$$

For a schedule with a longer period, one would have to perform a similar analysis on a larger timed EG. For instance, the timed EG corresponding to the schedule $w = (abcdcdab)^\omega$ is shown on Fig. 10.

A.2 Illustrating the automata approach

We associate with the timed Petri net of Fig. 8 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. 8) 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.

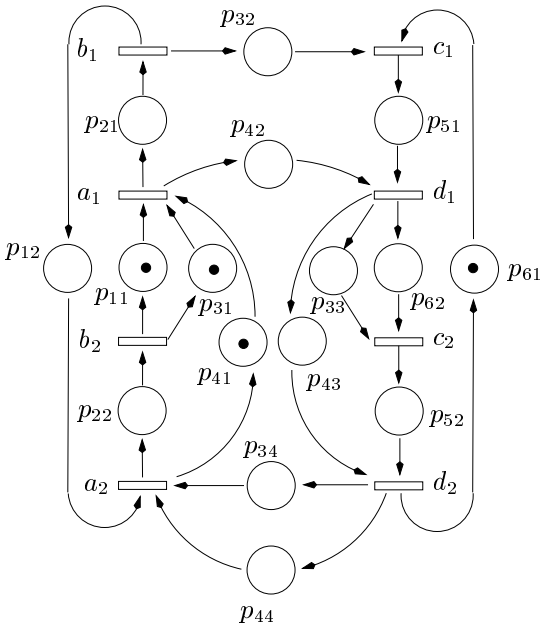


Fig. 10. Timed Event Graph for the schedule $(abcdcdab)^\omega$

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

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

For instance, the matrix $\mathcal{M}(abcd)$ was given in Ex. II.6. Its eigenvalue is $\rho(\mathcal{M}(abcd)) = 8$, providing a throughput $\lambda_i = 1/8$, which confirms the value obtained in Eqn (14).

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. 6.

These computations could be performed equivalently, and more simply, with the three dimensional matrices corresponding to the minimal realization, see Fig. 6,(3).

More generally, one can easily check that the optimal periodic schedule of period $4n$ 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. 6 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 and Chretienne [9, §VI-1].

We next turn our attention to the general class of jobshops.

B. Performance evaluation of safe jobshops

Definition V.1: A jobshop is specified by:

- a finite set $\mathcal{R}$ of resources (machines);
- a finite set $\mathcal{T}$ of elementary tasks;
- for each task $a \in \mathcal{T}$, a duration $\tau(a)$ and a single machine $R(a) \in \mathcal{R}$ on which a is to be executed;
- a finite set $\mathcal{J} \subset \mathcal{T}^*$ of production sequences or *jobs*. Each job $J = a_1 \dots a_k \in \mathcal{J}$ is composed of a finite number of tasks $a_1 \dots a_k$, to be executed in this order. We require[10] that a task $a \in \mathcal{T}$ belongs to a unique job $J(a) \in \mathcal{J}$.

We say that a unit of *job* J is produced each time the production sequence J is completed.

This model is equivalent to the one of [27]. We will make a restriction by assuming that the work in process for each job is equal to one, that is, at most one unit of each job is processed simultaneously. We call such jobshops *safe*, this assumption being equivalent to the safeness of the natural Petri net representation of the system, shown in Fig. 11.

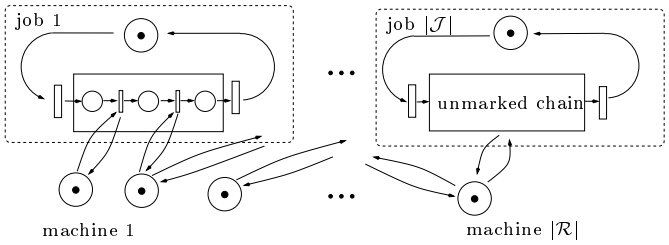


Fig. 11. Generic safe Petri net associated with a safe jobshop

We also assume that the jobshop, or equivalently the Petri net, is connected. It means that there is no proper subset of jobs sharing no resources with some other jobs. The extension to the non-connected case is straightforward.

The Petri net admits a natural covering by live and safe state machine components, each job and machine corresponding to a component. Hence, jobshops admit reduced heap realizations (see Theorem IV.5):

$$\begin{aligned} &\mathcal{H} = (\mathcal{T}, \mathcal{R}', R', l, u),\ \mathcal{R}' = \mathcal{R} \cup \mathcal{J}, \\ &R'(a) = R(a) \cup J(a), \\ &l(a,r) = 0, u(a,r) = \tau(a),\ \forall r \in R(a)\ . \end{aligned} \tag{15}$$

We will be interested in periodic schedules of the form $v^\omega \in \mathcal{T}^\omega$, where v belongs to L, the language of the Petri net, and is a pattern corresponding to the exact completion of several jobs (i.e. without leaving some production sequence unfinished) and meeting a fixed ratio constraint.

In line with (12), for each production sequence J, we will denote by $|v|_J$ the number of units of J completed under v. The asymptotic throughput of job J is defined as in (13), replacing J_i by J. The following result is an immediate consequence of the heap representation theorem, together with the cyclicity theorem for powers of matrices [2, § 3.7.5, Th. 3.112].

Theorem V.2 (Throughput Formula) For a safe jobshop with heap realization (15) and associated matrix representation $\mathcal{M}$, the asymptotic throughput of job J is given by

$$\lambda_J = |v|_J \times \rho(\mathcal{M}(v))^{-1} \ ,$$

[10] We may always assume this. If the same physical task occurs in two jobs, we have to represent it by two distinct letters.

where $\rho(\mathcal{M}(v))$ is the (max,+) eigenvalue of $\mathcal{M}(v)$.

As the jobshop is assumed to be connected, the matrix $\mathcal{M}(v)$ is irreducible, hence it has a unique eigenvalue, see [2]. As a byproduct of this theorem, we obtain an algorithm to compute λ_J.

Algorithm V.3 (Automata-based)

Input: a jobshop, a pattern $v \in L$.

1. Build the Heap model (15), and its associated matrices $\mathcal{M}(a)$, $a \in \mathcal{T}$.
2. Compute the product of matrices $\mathcal{M}(v)$. *Complexity*[11]*:* $O(|v|(|\mathcal{J}|+|\mathcal{R}|))$.
3. Compute the eigenvalue of $\mathcal{M}(v)$, $\rho(\mathcal{M}(v))$, using Karp algorithm. *Complexity:* $O(|\mathcal{J}|+|\mathcal{R}|)^3$.

Output: $\lambda_J = |v|_J \times \rho(\mathcal{M}(v))^{-1}$.

Total complexity: $O(|v|(|\mathcal{J}|+|\mathcal{R}|) + (|\mathcal{J}|+|\mathcal{R}|)^3)$.

The $|v|-1$ products of matrices in $\mathcal{M}(v)$ can be computed in a sparse way. Indeed, the matrices $\mathcal{M}(a)$, $a \in \mathcal{T}$, differ from the identity matrix only on two row and two column indices, so that the complexity of an operation $M\mathcal{M}(a)$ for any matrix M of size $|\mathcal{J}|+|\mathcal{R}|$ is $O(|\mathcal{J}|+|\mathcal{R}|)$. For details on Karp algorithm, see e.g. [2, Th. 2.19].

For comparison, we next give the most efficient variant known to us of the "classical" performance evaluation algorithm. The general method is borrowed from [2], but the refinements which greatly reduce the execution time can not be found in the literature.

Algorithm V.4 (Classical or Event Graph-based)

Same input and output as Algorithm V.4.

1. Build the timed EG representation of the system, following [2, § 2.6] or [27].
2. Write the (max,+) linear representation

$$x(n) = A_0 x(n) \oplus A_1 x(n-1) \ , \tag{16}$$

where x denotes the vector of *dater functions* of the transitions of the timed Event Graph, [2, § 5.1].

3. Let C denote the set of transitions with at least one token in one downstream place. Compute the $C \times C$ submatrix $\overline{A}$ of $A_0^* A_1$. *Complexity:* $O(|v|(|\mathcal{J}|+|\mathcal{R}|)) + (|\mathcal{J}|+|\mathcal{R}|)^2)$.
4. Compute the eigenvalue $\rho(\overline{A})$, using Karp algorithm. *Complexity:* $O(|\mathcal{J}|+|\mathcal{R}|)^3$.

Total complexity: $O(|v|(|\mathcal{J}|+|\mathcal{R}|) + (|\mathcal{J}|+|\mathcal{R}|)^3)$.

The timed EG built in step 1 of Algorithm V.4 has $|v|$ transitions and $|\mathcal{J}|+|\mathcal{R}|$ tokens. Hence, the matrices A_0 and A_1 in Eqn (16) are of dimension $|v| \times |v|$. For live timed event graphs, the matrix A_0 has no circuits. Then, we can derive from (16) the canonical form

$$x(n) = A_0^* A_1 x(n-1) \ ,$$

where $A_0^* = A_0^0 \oplus A_0 \oplus A_0^2 \oplus \cdots \oplus A_0^{|v|-1}$ (see [2, Th. 3.17]). By construction of C, if $j \notin C$, $(A_0^* A_1)_{ij} = \mathbb{0}$, for all i. Hence, $A_0^* A_1$ has only one nontrivial diagonal block $\overline{A}$, in position $C \times C$, and the growth rate of $(A_0^* A_1)^k$ coincides with the growth rate of $\overline{A}^k$.

[11] This is the execution time, the usual operations (comparison, addition, etc) counting for one unit.

The cardinal $|C|$ varies with the marking but is of order $|\mathcal{J}|+|\mathcal{R}|$. To compute $(A_0^* A_1)_{CC}$, we have 1) to select the rows of indices $i \in C$ of A_0^*, 2) to multiply each row of index $i \in C$ of A_0^* by each column of index $j \in C$ of A_1. Since the matrix A_0 has no circuits, using a rank function, we can compute a row of A_0^* in time $O(E)$, where E is the number of arcs of the graph of A_0 (see [24, Chap. 2,§2.4]). Here we have $E = O(|v|)$. We conclude that the complexity of computing $|C|$ rows of A_0^* is $O(|v|(|\mathcal{J}|+|\mathcal{R}|))$. On each column of A_1, there are at most two terms different from $\mathbb{0}$. Thus, computing the $|C|^2$ entries of $(A_0^* A_1)_{CC}$ takes a time $O(|v|(|\mathcal{J}|+|\mathcal{R}|)) + (|\mathcal{J}|+|\mathcal{R}|)^2)$.

The total complexities of both algorithms are the same. However, we did not take into account the "modeling" complexities for both methods. This aspect can be considered as a strong argument in favor of the automata-based algorithm. A new timed Event graph must be built for each new schedule in the traditional method, whereas the heap realization $\mathcal{M}$ is built only once, and remains valid for all (even non-periodic!) schedules. Moreover, the size of the timed EG grows with the length of the pattern, whereas the size of the heap automaton remains constant.

Remark V.5: Additional features can be incorporated to the jobshop of Def. V.1, the above modeling by heap automaton remaining valid. First, a task may require several resources at the same time to be executed. Second, there might be general precedence relations between the tasks of a job (it corresponds to replacing the live and safe circuits in Fig. 11 by live and safe Event Graphs).

Example V.6: We illustrate Theorem V.2 with the jobshop described in [27], § III. There are three machines: $\mathcal{R} = \{M_1, M_2, M_3\}$, four production sequences $\mathcal{J} = \{J_1, \ldots, J_4\}$, $J_1 = a_1 a_2 a_3$, $J_2 = b_1 b_2$, $J_3 = c_1 c_2$, $J_4 = d_1 d_2$. The resource allocation and time execution maps are given in the following table:

Task	a_1	a_2	a_3	b_1	b_2	c_1	c_2	d_1	d_2
R	M_1	M_2	M_3	M_3	M_2	M_1	M_3	M_1	M_3
τ	1	3	3	1	2	2	1	2	1 .

One requires a production mix of $1/4, 1/4, 1/4, 1/4$ between J_1, J_2, J_3, J_4. A schedule satisfying this constraint (and compatible with the order of precedence of the elementary tasks) is v^ω, with $v = a_1 a_2 a_3 b_1 b_2 c_1 c_2 d_1 d_2$. The matrix of the resource automaton of a_1 is (omitting the $\mathbb{0}$ entries):

$$\mathcal{M}(a_1) = \begin{array}{c} \\ M_1 \\ M_2 \\ M_3 \\ J_1 \\ J_2 \\ J_3 \\ J_4 \end{array} \begin{array}{c} \begin{array}{ccccccc} M_1 & M_2 & M_3 & J_1 & J_2 & J_3 & J_4 \end{array} \\ \begin{pmatrix} 1 & & & 1 & & & \\ & \mathbb{1} & & & & & \\ & & \mathbb{1} & & & & \\ 1 & & & 1 & & & \\ & & & & \mathbb{1} & & \\ & & & & & \mathbb{1} & \\ & & & & & & \mathbb{1} \end{pmatrix} \end{array}$$

We leave it to the reader to write the other matrices, and just give

$$\mathcal{M}(v) = \begin{pmatrix} 7 & 10 & 9 & 5 & 10 & 10 & 10 \\ \mathbb{0} & 3 & 2 & \mathbb{0} & 3 & 3 & 3 \\ \mathbb{0} & \mathbb{0} & 3 & 4 & \mathbb{0} & 5 & 5 \\ 7 & 10 & 9 & 5 & 10 & 10 & 10 \\ 6 & 9 & 8 & \mathbb{0} & 9 & 9 & 9 \\ 3 & 6 & 5 & \mathbb{0} & 6 & 6 & 6 \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & 2 & \mathbb{0} & 3 & 3 \end{pmatrix} .$$

We have $\rho(\mathcal{M}(v)) = 9$, which yields the throughput of 1/9, as in §V of [27]. We used a toy implementation[12] in Maple V.3. There are $\binom{3+2+2+2}{3,2,2,2} = 9!/(3!(2!)^3) = 7560$ schedules (a schedule is a word in the shuffle product L of the four words $a_1a_2a_2$, b_1b_2, c_1c_2, d_1d_2). But the performance of a schedule only depends on its equivalence class modulo the partial commutations $xy \sim xy$, for the couples of tasks (x, y) belonging either to the same job or to the same machine. There are only 216 such equivalence classes. Moreover, since the periodic throughput is invariant by cyclic conjugacy of the pattern (recall that two words of the form uv and vu are *cyclic conjugates*), we only kept one word by class of cyclic conjugacy (192 words remained). Finally, we had to compute the matrix product $\mathcal{M}(w)$ for these 192 words. We found an optimal throughput of 1/7, attained for instance for the cyclic schedule $\tilde{v}^\omega$, with $\tilde{v} = a_1d_1b_1a_2b_2c_1a_3d_2c_2$. This approach has been developed in [26]. It can be combined with branch and bound techniques that are classical in scheduling.

Remark V.7: 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, Ex. III.5, Ex. IV.2, Ex. IV.6, §V-A) and the schedule *abcdcdabab*. The two added pieces *ab* seem to "float" in the air because one critical modeling variable is lacking. Indeed, the smallest heap representation is of size three, see Fig. 6,(3).

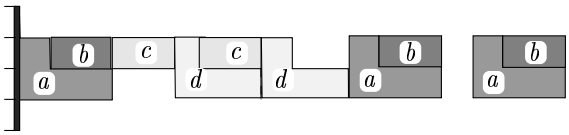


Fig. 12. (Conventional) Gantt charts are not (max,+) linear.

VI. Epilogue: Algebraic Status of Petri Net Heap Representations

In order to apply the machinery of automata to performance evaluation problems, we next discuss at a more algebraic level the different models used in this paper. 1) At the logical level, the set of admissible behaviors of a Petri net $\mathcal{G}$ is described by its language L, which is recognized by a "classical" automaton (deterministic Boolean automaton), the marking automaton. 2) At the time level, the execution time of an admissible sequence $w \in L$ is recognized by a heap automaton. It is very natural to embed both models in a common algebraic framework, as follows.

[12]Available on the author's web pages:
http://amadeus.inria.fr/gaubert/jobshop.html, and
http://www.liafa.jussieu.fr/~mairesse/jobshop.html

A. (Max,+) automaton and heap automaton

Classically, a (max,+) automaton $\mathcal{A} = (Q, I, F, \mathcal{M})$ over the alphabet $\mathcal{T}$ can be represented by a finite $\mathbb{R}_{\max}$-valued and $\mathcal{T}$-labeled graph as follows. One draws a finite graph with nodes $q \in Q$. There are three types of arcs. For each q such that $I_q \neq \mathbb{0}$, one draws an input arc valued by the scalar I_q (with no label). Such a node q is called *initial*. Dually, for each q such that $F_q \neq \mathbb{0}$, one draws an output arc valued by the scalar F_q (with no label). Such a node q is called *final*. For each triple $(q, a, q') \in Q \times \mathcal{T} \times Q$ such that $\mathcal{M}(a)_{qq'} \neq \mathbb{0}$, one draws an arc from q to q', labeled with the letter a and valued by the scalar $\mathcal{M}(a)_{qq'}$.

The *weight* of a path $\mathrm{p} = (q_1 \xrightarrow{a_1} q_2 \dots q_n \xrightarrow{a_n} q_{n+1})$ is the product $w(\mathrm{p}) = I_{q_1}\mathcal{M}(a_1)_{q_1q_2} \dots \mathcal{M}(a_n)_{q_nq_{n+1}}F_{q_{n+1}}$, evaluated in the $\mathbb{R}_{\max}$ semiring. The *label* of this path is the product of the labels of the edges: $\ell(\mathrm{p}) = a_1 \dots a_n$. Then, the multiplicity of the word w is equal to the max of the weights of the paths of label w: $y_{\mathcal{A}}(w) = \bigoplus_{\mathrm{p}:\ell(\mathrm{p})=w} w(\mathrm{p})$.

Example VI.1: As an illustration, we provide in Fig. 13 the graphical representation of the (max,+) automaton of Ex. II.6.

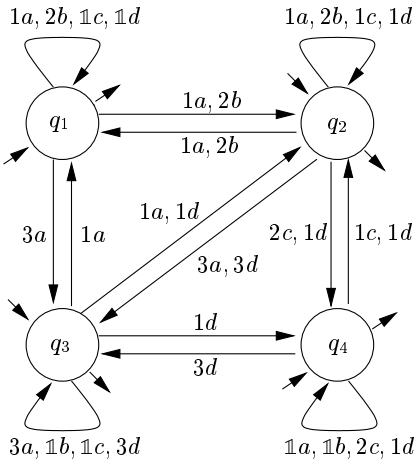


Fig. 13. (Max,+) automaton.

B. Boolean automaton and marking automaton

Starting from a (max,+) automaton, by specialization to the Boolean semiring[13], one obtains the classical notion of (nondeterministic) automaton: the multiplicity of w is $\mathbb{1}$ if the word w is *accepted*, i.e. if there is a path with label w from an initial state to a final state, and $\mathbb{0}$ otherwise. The *language* of a (Boolean) automaton is the set of words accepted by this automaton.

A Boolean automaton is *deterministic* if for all $(q, a) \in Q \times \mathcal{T}$, there is at most one q' such that $\mathcal{M}(a)_{qq'} \neq \mathbb{0}$, and if there is a unique q such that $I_q \neq \mathbb{0}$.

Let $\mathcal{G} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, M)$ be a safe Petri net. The *marking automaton*[14] of $(\mathcal{G}, M)$ is the deterministic Boolean automaton $(\mathsf{R}(M), I', \mathbb{1}_{\mathsf{R}(M)}, \mathcal{M}')$, where the initial vector I'

[13]The Boolean semiring $\mathbb{B} = (\{\text{false}, \text{true}\}, \text{or}, \text{and})$ is isomorphic to the subsemiring $(\{\mathbb{0}, \mathbb{1}\}, \oplus, \otimes)$ of $\mathbb{R}_{\max}$.

[14]Also known as *marking graph* or *reachability graph*.

is defined by $I'_M = \mathbb{1}$, and $I'_m = \mathbb{0}$ for $m \neq M$, and the morphism $\mathcal{M}'$ is defined by $\mathcal{M}'(a)_{M'M''} = \mathbb{1}$ if $M' \xrightarrow{a} M''$ and $\mathcal{M}'(a)_{M'M''} = \mathbb{0}$ otherwise, for all $a \in \mathcal{T}$.

By construction, a word $w \in \mathcal{T}^*$ is a firing sequence of the Petri net $(\mathcal{G}, M)$ *iff* it is accepted by the marking automaton. The language of the Petri net, defined in § III, is the language of the marking automaton. We remark that by definition of the independence relation $\mathcal{I}$ (see (5)), matrices associated with independent transitions commute:

$$(a,b) \in \mathcal{I} \implies \mathcal{M}'(a)\mathcal{M}'(b) = \mathcal{M}'(b)\mathcal{M}'(a) \ . \qquad (17)$$

Example VI.2: The marking automaton of the Petri net of Fig. 2 is shown on Fig. 14. The state corresponding to marking m is denoted by the couple $(m(p_1), m(p_6))$.

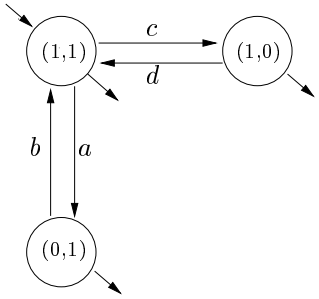


Fig. 14. Marking automaton of the Petri net of Fig. 2.

The language of the automaton, $L = (ab \cup cd)^*(e \cup a \cup c)$, coincides with the language of the net (see Ex. III.3).

C. Heap representation theorems revisited

We extend the daters of the net $x_{\mathcal{G}}$ and $y_{\mathcal{G}}$, which were preciously only defined on L, by setting, for any $w \in \mathcal{T}^* \setminus L$, $\forall p \in \mathcal{P}$, $x_{\mathcal{G}}(w)_p = \mathbb{0}$, $y_{\mathcal{G}}(w) = \mathbb{0}$. We have the following generalization of Theorem IV.1.

Theorem VI.3: The daters $(x_{\mathcal{G}})_p$, $p \in \mathcal{P}$, and $y_{\mathcal{G}}$ of a safe timed Petri net are recognized by a (max,+) automaton.

Proof: We prove that $y_{\mathcal{G}}$ is (max,+) recognizable (the argument for the entries of $x_{\mathcal{G}}$ is identical). Consider the characteristic dater of the language L: $\operatorname{char} L(w) = \mathbb{0}$ if $w \notin L$, $\operatorname{char} L(w) = \mathbb{1}$ if $w \in L$. The dater $\operatorname{char} L$ is recognized by the *marking automaton* $\mathcal{B} = (\mathsf{R}(M), I', \mathbb{1}_{\mathsf{R}(M)}, \mathcal{M}')$. Let $\mathcal{H} = (\mathcal{R}, \mathbb{1}^t_{\mathcal{R}}, \mathbb{1}_{\mathcal{R}}, \mathcal{M})$ be any heap automaton recognizing $y_{\mathcal{G}}$. Classically [20], the product $y_{\mathcal{G}}(w) = y_{\mathcal{H}}(w)\operatorname{char} L(w)$ is recognized by the *tensor product* of the automata $\mathcal{H}$ and $\mathcal{B}$, which is the automaton $\mathcal{C} = (\mathcal{R} \times \mathsf{R}(M), I'', F'', \mathcal{M}'')$, with:

$$I''_{(r,m)} = (\mathbb{1}^t_{\mathcal{R}})_r I'_m = I'_m, \quad F''_{(r,m)} = (\mathbb{1}_{\mathcal{R}})_r (\mathbb{1}_{\mathsf{R}(M)})_m = \mathbb{1},$$
$$\mathcal{M}''(a)_{(r,m),(s,m')} = \mathcal{M}(a)_{rs}\mathcal{M}'(a)_{mm'} \ . \qquad (18)$$

■

Due to the commutations in (4) and (17), $\mathcal{M}''(a)\mathcal{M}''(b) = \mathcal{M}''(b)\mathcal{M}''(a)$, for all $(a,b) \in \mathcal{I}$. This implies that $y_{\mathcal{G}}$, seen as a function from the trace monoid $\mathcal{T}^*/\sim \to \mathbb{R}_{\max}$, is *recognizable*. Such functions are well studied objects, see [19] and the chapter of Duchamp and Krob in [18].

Remark VI.4: As it was discussed above, the dimension of $\mathcal{H}$ is the number of places (normal representation) or approximately the number of tokens (reduced realization) of the Petri net $\mathcal{G}$. On the other hand, the dimension of $\mathcal{B}$ can be extremely large, the only bound a priori being $2^{\mathcal{P}}$ which is deduced from the safeness assumption. A natural idea is to look for reduced representations of the marking automaton $\mathcal{B}$. Assume that n transitions are concurrently enabled and need to fire before any new transition becomes enabled. Then, the corresponding part of the marking automaton, which contains 2^n states, can be reduced to only n states by selecting a single firing sequence. Reduced marking automata can be incorporated to the modeling proposed in this paper. We have not insisted on this point as it is a well-documented problem, see e.g. Valmari [39] for a systematic way to construct a reduced marking automaton.

Theorem VI.3 allows us to apply the machinery of automata to safe timed Petri nets. In particular, the algorithms given in [21, §V] and [22, §3.2] enable us to compute very simply the *worst case Lyapunov exponent*

$$\gamma_{\max} = \limsup_n \frac{\max_{w \in \mathcal{A}^n \cap L} y_{\mathcal{H}}(w)}{n} \ , \qquad (19)$$

which measures the maximal growth rate of the makespan, for long sequences of admissible events (in L). Indeed, $\gamma_{\max}$ is equal to the maximal (max,+) eigenvalue of $\bigoplus_{a \in \mathcal{A}} \mathcal{M}''(a)$, where $\mathcal{M}''$ is the morphism defined in (18).

Other interesting quantities, such as the *optimal case Lyapunov exponent* (which, dually, measures the minimal growth rate of the makespan for long schedules)

$$\gamma_{\min} = \liminf_n \frac{\min_{w \in \mathcal{A}^n \cap L} y_{\mathcal{H}}(w)}{n} \ , \qquad (20)$$

can (in certain cases) be attacked with automata techniques, either along the lines of Cerin and Petit [11], or using the determinization techniques of [21]. But the difficulty of the computation of the optimal Lyapunov exponent $\gamma_{\min}$ if one order of magnitude above that of $\gamma_{\max}$.

In Eqn (19) and (20), we can consider a language L which is more general than the language of the Petri net. For example, as in §V-A, we can consider a language $L_1 \cap L_2$ where L_1 is the language of the Petri net and L_2 is a language corresponding to a ratio constraint. We postpone the discussion of these questions to a companion paper.

Concluding remarks

Let us indicate some possible extensions of this work. 1. It remains to develop heuristics and performance bounds for scheduling, based on heap and automata representations. 2. Matrix representations allow us to apply standard time parallelization methods (breaking long matrix products in subproducts, mapped on different processors) to the parallel simulation of stochastic Petri nets. 3. The main limitation of the heap-modeling presented here is the restriction to *safe* Petri nets. A natural question consists in characterizing the subclasses of non safe Petri nets for which such a heap modeling remains valid.

Acknowledgments

This work was initiated at BRIMS, Hewlett-Packard Laboratories, Bristol, were the first author was visiting and the second author was doing a post-doc. The authors thank Jeremy Gunawardena for his hospitality during their sojourn at BRIMS. The authors thank Eric Hautecloque and the referees of this paper for their useful comments.

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*. Wiley, 1992.

[3] F. Baccelli, S. Foss, and B. Gaujal. Free Choice Petri Nets: an Algebraic Approach. *IEEE Trans. on Automatic Control*, 41(12):1751–1778, 1996.

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

[5] G. W. Brams. *Réseaux de Petri, Théorie et pratique, Tome 1*. Masson, Paris, 1983.

[6] W. Brauer, W. Reisig, and G. Rozenberg, editors. *Petri Nets: Central Models and Their Properties*. Number 254 in LNCS. Springer, 1987.

[7] M. Brilman and J.M. Vincent. Dynamics of synchronized parallel systems. *Comm. Statist. Stochastic Models* (13)3:605–617, (1997).

[8] P. Brucker. *Scheduling Algorithms*. Springer, 1995.

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

[10] P. Chretienne, E. Coffman, J. Lenstra, and Z. Liu, editors. *Scheduling Theory and Its Applications*, Wiley, 1995.

[11] C. Cérin and A. Petit. Speedup of recognizable trace languages. *Proc. MFCS 93*. Number 711 in LNCS, Springer, 1993.

[12] 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*, (30):210–220, 1985.

[13] G. Cohen, S. Gaubert, and J.P. Quadrat. Algebraic system analysis of timed Petri nets. Appears in [25].

[14] G. Cohen, S. Gaubert, and J.P. Quadrat. Asymptotic throughput of continuous timed Petri nets. *Proceedings of the 34th Conference on Decision and Control*, New Orleans, Dec. 1995.

[15] R. Cuninghame-Green. *Minimax Algebra*. Number 166 in Lect. Notes in Economics and Math. Systems, Springer, 1979.

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

[17] V. Diekert. *Combinatorics on traces*. Number 454 in LNCS. Springer, 1990.

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

[19] M. Droste and P. Gastin. On recognizable and rational formal power series in partially commuting variables. *Proceedings of ICALP'97*, number 1256 in LNCS, p. 682-692, 1997.

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

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

[22] S. Gaubert and J. Mairesse. Task resource models and (max,+) automata. Appears in [25].

[23] S. Gaubert and M. Plus. Methods and applications of (max,+) linear algebra. In R. Reischuk and M. Morvan, editors, *Proceedings of STACS'97*, number 1200 in LNCS, Springer, 1997.

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

[25] J. Gunawardena, editor. *Idempotency*. Publications of the Newton Institute. Cambridge University Press, 1998.

[26] E. Hautecloque. *Empilements de pièces, semianneau (max,+) et ordonnancement*. Mémoire ENSTA et DEA "Modélisation et Méthodes Mathématiques en Économie", Université de Paris I, 1997.

[27] 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.

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

[29] L. Libeaut. *Sur l'utilisation des dioïdes pour la commande des systèmes à événements discrets*. Thèse de Doctorat, Université de Nantes, Sept. 1996.

[30] V. Maslov and S. Samborskiĭ, editors. *Idempotent analysis*, volume 13 of *Adv. in Sov. Math.* AMS, 1992.

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

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

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

[34] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[35] J.E. Pin. Tropical semirings. Appears in [25].

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

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

[38] I. Simon. On semigroups of matrices over the tropical semiring. *Theor. Infor. and Appl.*, 28(3-4):277–294, 1994.

[39] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri nets*, number 483 in LNCS, pages 491–515. Springer, 1991.

[40] 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.

[41] J.M. Vincent. Some ergodic results on stochastic iterative discrete events systems. *DEDS: Theory and Applications*, 7(2):209–233, 1997.

[42] U. Zimmermann. *Linear and Combinatorial Optimization in Ordered Algebraic Structures*. North Holland, 1981.