Algebraic techniques for timed systems ⋆

Albert Benveniste[1], Claude Jard[2], and Stéphane Gaubert[3]

[1] IRISA/INRIA, Campus de Beaulieu, F-35042 Rennes Cedex, France
`Albert.Benveniste@irisa.fr`
[2] IRISA/CNRS, Campus de Beaulieu, F-35042 Rennes Cedex, France
`Claude.Jard@irisa.fr`
[3] INRIA, BP 105, F-78153 Le Chesnay Cedex, France
`Stephane.Gaubert@inria.fr`

Abstract. Performance evaluation is a central issue in the design of complex real-time systems. In this work, we propose an extension of so-called "Max-Plus" algebraic techniques to handle more realistic types of real-time systems. In particular, our framework encompasses graph or partial order automata, and more generally abstract models of real-time computations (including synchronous programs running over distributed architectures). To achieve this, we introduce a new dioid of partially commutative power series (transductions), whose elements encode timed behaviors. This formalism extends the traditional representation of timed event graphs by (rational) commutative transfer series with coefficients in the Max-Plus semiring. We sketch how this framework can be used to symbolically solve several problems of interest, related to real-time systems. Then we illustrate the use of this framework to encode a nontrivial mixed formalism of dataflow diagrams and automata.

1 Motivations

Performance evaluation is a central issue in the design of complex real-time systems. The general situation we consider in this paper can be described as follows : 1/ The real-time system in consideration consists of a finite collection of tasks. Tasks are triggered by events originating from both the environment and the system itself, depending on its internal state. 2/ Tasks can be concurrent or serialized, depending on their causality interactions. And this may change dynamically depending on the state of the environment and of the system itself. 3/ Tasks need resources for their completion, and they wait until all resources needed are available.

Restrictions are listed next : 1/ Both system and environment states can take a *finite* number of values. If this is not the case, then some kind of abstraction is needed to enforce this situation (e.g., values of integer state variables are abstracted). 2/ State transitions are not triggered by the awaiting/reception of resources, i.e., watchdog/timeout mechanisms are not modelled. Again, if

⋆ This work is supported in part by Esprit LTR-SYRF project (EP 22703).

watchdog/timeout are involved, then they need to be abstracted, typically in the form of a possible nondeterministic exception, prior to enter our framework.

The general term of "real-time computing" refers to the type of computing that can be embedded on a real-time system with bounded memory and bounded response time requirements. Here our aim is to model the behaviour of real-time computing running on a given architecture with a given degree of available concurrency or parallelism. Basic real-time computations often have the form of single-clocked machines performing identical computations at each instant. The more general situation can be abstracted as a finite set of basic real-time computations having two-sided interaction with some finite state machine (the computations being performed depend on some discrete state of the system, and in turn can trigger discrete state transitions) [2, 13]. In this context, questions of interest include: latency, throughput, bounded time safety properties (guaranteing that some property will occur within some given bounded period of time).

These applications motivated us for developing a general algebraic framework. In the modelling of timed discrete event systems, one traditionally uses dater functions, which give completion times, as a function of numbers of events (see [6] and [1, Ch. 5]). Dater functions are non-decreasing. We extend this modelling to the case of multiform logical and physical times, which are needed to model concurrent behaviors. We represent event sequences and time instants by words. A dater is a map, which associates to a word a word, or a set of words, and which is non-decreasing for the subword order. The formal series associated with these generalized dater functions live in a finitely presented semiring, which is equipped with some remarkable relations, due to the monotone character of daters.

The systematic study of the underlying algorithmic problems is beyond the scope of this paper, and the adapted software tools remain (mostly) to be developed. Our aim here is only to illustrate the interest of the formalism, by showing how the model fits the above mentioned particular applications. Questions of effectiveness and complexity are deferred to a subsequent paper.

2 Discussing models of real-time computation

To introduce our model we shall discuss informally the case of real-time computation. Figure 1 depicts a simple example of a "basic computation", in which the same computation is repeated each cycle indefinitely.

Of course, in more realistic situations, different computations would be performed for different discrete states of the control of the program. Using the same notations for graph concatenation as in Figure 1, a simple prototype example of such a model for computation is given in Figure 2, using an order-automaton.

We assume that performing any event (thick bullet) takes an integer number of cycles (time units). Our aim is to model the timing behaviour of this machine. Let us first concentrate on action A. Denote by $d_{\mathtt{X}}$ and $d_{\mathtt{Y}}$ some current date attached to the flows X and Y respectively, and by $d'_{\mathtt{X}}$ and $d'_{\mathtt{Y}}$ corresponding dates

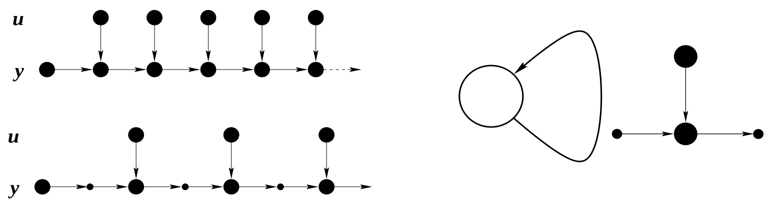

Fig. 1. *Computation :* $\forall n\ y_n = y_{n-1} + u_n$, *modeled as a graph automaton.* The first diagram depicts the abstraction of the computation $\forall n\ :\ y_n = y_{n-1} + u_n$ in the form of an infinite string of dependence graphs. In this formula, u and y are *flows,* i.e., infinite sequences of data. In the second diagram, small bullets indicate the switching from one instant n to the next one. These small bullets are used as "pins" which glue each instantaneous dependency graph to its predecessor and successor : this yields a notion of concatenation. This notion of concatenation is used in the third diagram to construct the "language" a^{ω}, where a is the symbol consisting of the graph sitting as label for the unique transition of the depicted automaton.

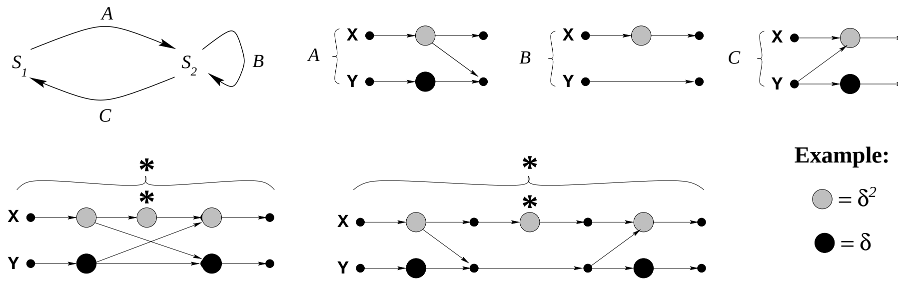

Fig. 2. *The main example.* The top left diagram depicts the automaton. Its action labels A, B, C are directed graphs connecting flow occurrences. These directed graphs are shown on the right hand side, they are interpreted as labelled partial orders. Pins are depicted by small bullets, while actual elementary computations (called "events" in the sequel) are depicted by thicker bullets. Black and gray events cost one (δ) or two (δ^2) units of time respectively. Since concatenation is performed on a flow-by-flow basis, with pins subsequently erased, the resulting partial order is shown on the bottom left diagram. In this diagram, symbol $*$ denotes the star operator, it applies in a nested way, both to the innermost event on flow labelled $\mathtt{X}$, as well as to the whole partial order shown.

after completing action A. One has $d'_{\mathtt{X}} = d_{\mathtt{X}} + 2$ and $d'_{\mathtt{Y}} = \max\left(d_{\mathtt{X}} + 2\ ,\ d_{\mathtt{Y}} + 1\right)$. This equation is linear over the max-plus semiring (see §3.1 below). Indeed, introducing the delay operator $\delta : \delta x = x + 1$, and using the max-plus notation $a \oplus b = \max(a, b)$, we can write: $d'_{\mathtt{X}} = \delta^2 d_{\mathtt{X}}$ and $d'_{\mathtt{Y}} = \delta^2 d_{\mathtt{X}} \oplus \delta d_{\mathtt{Y}}$.

In order to express such relations, we propose to abstract order-automata, as illustrated in Figure 3, by counting the dates on each flow and taking into account the inter-flow dependencies due to the obliques in the patterns. Each pattern may be represented by the timed-dependence-graphs shown in the left of Figure 3. The execution times of the events (one or two units) are materialized by delays operators (δ or δ^2). Notice that a dependency edge can be of 0-delay and will only propagate the dependency without increasing the delay. We equip the name of each involved flow with an attribute consisting of the states of the transition relation of the automaton. We also note on each dependency edge the name of the pattern from which it is defined.

Merging these graphs, we obtain the *timed diagram* of Figure 3 (right). As usual, to each state S we can associate a set of runs (a trajectory) which can be executed from the initial state of the automaton to state S.

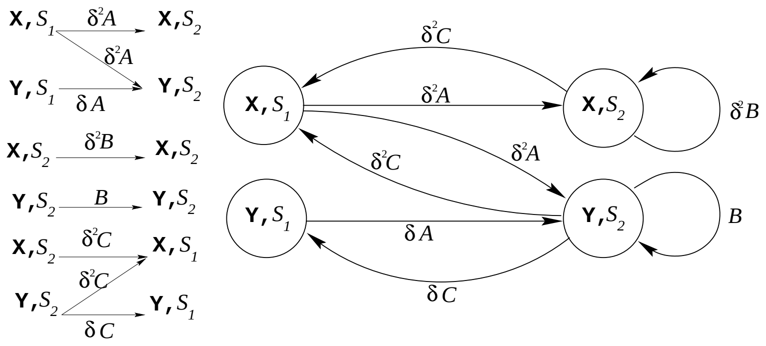

Fig. 3. *A timed diagram.* Each pattern A,B and C of the order-automaton of Figure 2 is abstracted as the left figure. The "union" of these graphs forms the timed diagram of the right. This diagram defines a family of orders, described by words (trajectories) of patterns A,B and C. For each trajectory, we want to compute the time at which it is completed at the earliest. From this diagram, we can infer that action A is completed at the earliest at the time 2 ($max(\delta, \delta^2)$) and that the words AB and AC are completed at the earliest at time 4 ($max(\delta^3, \delta^4)$).

We think that all the useful information about propagation of delays along the sequences of actions is captured in this representation. The question now is to equip such a graph with a mathematical semantics.

Let us consider an edge of a trajectory. It is labeled with words of events, possibly interleaved with the delay symbol δ. For modelling the elapsed time, we decide that delays and events commute. Edges are thus labelled with symbols $\delta^t w$.

What we seek for is that symbol $\delta^t w$ should encode the information

$$\delta^t w \ : \ \text{“}w \text{ is completed at the earliest at time } t.\text{”} \tag{1}$$

This is equivalently rephrazed as

- when w is completed, one cannot have a date $s, s < t$,
- at time $s < t$, one cannot have performed *more than* w, where performing more than w means performing some v such that w is a subword of v, written $w \ \sqsubset \ v$, i.e., w is obtained by replacing in v some symbols by the empty word.

Note that we consider a *subword* w, not a *prefix*: the reasons for this will become clear soon.

The next section presents the encoding of the trajectories into an algebraic framework. This will provide semantics for our timed diagrams. Its presentation (as usual for automata) consists in writing a system of equations in some suitable formal power series domain. The central question is to define which domain of formal power series must be used: 1/ What is the domain of $\mathtt{X}_S$? 2/ What do addition and multiplication mean?

3 A new formal power series domain

3.1 A glimpse of dioid algebra

A *semiring* is a set S, equipped with two laws $\oplus$, $\otimes$, such that $(S, \oplus)$ is a commutative monoid (the zero element will be noted $\mathbf{0}$), $(S, \otimes)$ is a monoid (the unit element will be noted $\mathbf{1}$), the law $\otimes$ is right and left distributive over $\oplus$, and $\mathbf{0}$ is absorbing. A semiring whose addition is idempotent ($a \oplus a = a$) is a *dioid*. Dioids are canonically ordered by the relation $a \leq b$ iff $b = a \oplus b$. We will consider two particular dioids. The *max-plus semiring* $\mathbf{Z}_{\max}$ is the set $\mathbf{Z} \cup \{-\infty\}$, equipped with: $a \oplus b = \max(a, b), a \otimes b = a + b, \mathbf{0} = -\infty$, $\mathbf{1} = 0$, and the order relation $\leq$ is the ordinary one. The *boolean semiring* $\mathbf{B}$ can be identified to the subsemiring $\{\mathbf{0}, \mathbf{1}\}$ of $\mathbf{Z}_{\max}$. In any semiring, the matrix product can be defined as usual:

$$(A \otimes B)_{ij} =_{\mathrm{def}} \bigoplus_k A_{ik} \otimes B_{kj}$$

where A, B are matrices with compatible dimensions. We shall write AB instead of $A \otimes B$, as usual.

3.2 Modelling time elapsing

A rough dioid of formal power series. Consider the set $\mathcal{M}\,(\Sigma, \delta)$ of formal power series

- with coefficients belonging to the boolean dioid $\{\mathbf{B}, \oplus, \otimes\}$ — in the sequel, to be consistent with our generic dioid notations, we write $\{\mathbf{0}, \mathbf{1}\}$ instead of $\{0, 1\}$;
- with variables δ to encode the dates, and $\Sigma = \{A, B, C\}$ to encode the moves of the automaton.
  Variables A, B, C do not commute, a word built with these variables shall be denoted by the generic letter w. On the other hand, w and δ globally commute: $w\delta = \delta w$.

Following the usual notation [19, 5], such formal power series are written

$$x = \bigoplus_{t \in \mathbf{Z}\,,\, w \in \Sigma^*} x_{t,w} \;.\; \delta^t w \;, \quad \text{or} \quad \bigoplus_{t \in \mathbf{Z}\,,\, w \in \Sigma^*} \delta^t w \;.\; x_{t,w}$$

for convenience, where $x_{t,w}$ denotes the *coefficient* of x at the monomial $\delta^t w$. The set $\mathcal{M}\,(\Sigma, \delta)$ inherits the following dioid structure from that of its coefficients domain:

– $x \oplus y$ is obtained by taking the $\oplus$ componentwise.
– $x \otimes y$ is the Cauchy product defined by $(x \otimes y)_{t,w} = \bigoplus_{r+s=t,\ uv=w} (x_{r,u} \otimes y_{s,v})$

We shall denote again by $\mathbf{0}$ the element of $\mathcal{M}(\Sigma, \delta)$, which has all its coefficients equal to $\mathbf{0}$.

The idea is that formal series x models *timed runs, or trajectories* i.e., languages together with the time needed to complete words. For $x \in \mathcal{M}(\Sigma, \delta)$,

$$x_{t,w} = \mathbf{1} \text{ iff word } w \text{ is completed at time } t \text{ for trajectory } x. \qquad (2)$$

The zero series $\mathbf{0} \in \mathcal{M}(\Sigma, \delta)$ represents the trajectory with no event at all. Note at this point that statement (2) is different from requirement (1), thus $\mathcal{M}(\Sigma, \delta)$ is not our desired domain and further work is needed.

The quotient dioid $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$. Here we follow the beautiful idea [6, 1] of Cohen, Moller, Quadrat and Viot to derive symmetric codings for event graphs (the case considered in [6, 1] corresponds to our situation for the particular case in which the automaton has a single action label).

We are now ready to formally construct the quotient dioid which provides the semantics requested in (1). For

$$x = \bigoplus_{t \in \mathbf{Z},\, w \in \Sigma^*} x_{t,w} \,.\, \delta^t w \quad \text{we set} \quad [x] = \bigoplus_{t \in \mathbf{Z},\, w \in \Sigma^*} \delta^t w \,.\, \bigoplus_{\substack{s \geq t \\ v \sqsubseteq w}} x_{s,v} \qquad (3)$$

Note that (3) formally encodes statement (1), by attaching a $\mathbf{1}$ to $\delta^t w$ as soon as some $x_{s,v} = \mathbf{1}$ for $s \geq t$ (first rephrazing of (1)) or $v \sqsubseteq w$ (second rephrazing of (1)). Then, the following holds:

Theorem 1 (the $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ quotient dioid).

1. *The following holds:*

$$[x \oplus y] = [x] \oplus [y] \;,\; [x \otimes y] = [x] \otimes [y]$$

which implies that the equivalence relation $\sim$ ($x \sim y$ iff $[x] = [y]$) is compatible with the dioid structure of $\mathcal{M}(\Sigma, \delta)$. Thus we can consider the dioid $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ obtained by taking the quotient of dioid $\mathcal{M}(\Sigma, \delta)$ by the $\sim$ equivalence relation.

2. *The following formula holds:*

$$\left[\delta^t w\right] = \bigoplus_{\substack{s \leq t \\ v \sqsupseteq w}} \delta^s v \;.$$

3. *In the quotient dioid $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$, the monomials obey the following rules:*

$$\left[\delta^s w\right] \oplus \left[\delta^t w\right] = \left[\delta^{\max(s,t)} w\right] \qquad (4)$$

$$\left[\delta^t v\right] \oplus \left[\delta^t w\right] = \left[\delta^t \min(v, w)\right] \qquad (5)$$

if one of the words v, w is a subword of the other one, and $\min(v, w)$ *denotes this subword.*

Comment : Points 2 and 3 of theorem 1 enlight that dioid $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ really implements specification (1), if one interprets monomials as *informations* or *constraints* (see (1)), and addition as *logical conjunction.* Indeed, the rule (4) simply means that the conjunction of the informations "w occurs at the earliest at time s" and "w occurs at the earliest at time t" is "w occurs at the earliest at time $\max(s,t)$". The interpretation of (5) is dual.

Proof: For the first formula, $[x \oplus y] = \bigoplus_{t,w} \delta^t w \,.\, \bigoplus_{s \geq t,\, v \sqsubseteq w} (\, x_{s,v} \oplus y_{s,v} \,) = [x] \oplus [y]$ just because $\oplus$ is associative componentwise. This was the easy part. Now comes the subtle one, as well as the justification for considering "subwords" instead of "prefixes".

For
$$x \otimes y = \bigoplus_{t,w} \delta^t w \,.\, \bigoplus_{\substack{r+s=t \\ uv=w}} (\, x_{r,u} \otimes y_{s,v} \,) \ ,$$

then it holds that $[x \otimes y] = \bigoplus_{t,w} \delta^t w \,.\, \bigoplus_{\substack{t' \geq t \\ w' \sqsubseteq w}} \bigoplus_{\substack{r'+s'=t' \\ u'v'=w'}} [\, x_{r',u'} \otimes y_{s',v'} \,] = [x] \otimes [y]$ where the last equality follows from the fact that the following two sets are equal : $\{(u',v') \ : \ u'v' \sqsubseteq w\} = \{(u',v') \ : \ u' \sqsubseteq u \,,\, v' \sqsubseteq v \,,\, uv = w\}$

Note that this latter property would be *false* if we had chosen "is prefix of" instead of "is a subword of".

We move to points 2 and 3. We have :

$$\begin{aligned} [\delta^t w] &= \bigoplus_{s,v} \delta^s v. \bigoplus_{\substack{r \geq s \\ u \sqsubseteq v}} (\delta^t w)_{r,u} \\ &= \bigoplus_{s,v} \delta^s v \,.\, (\text{ if } t \geq s \text{ and } w \sqsubseteq v \ , \quad \text{then } \mathbf{1} \text{ else } \mathbf{0}) = \bigoplus_{\substack{s \leq t \\ v \sqsupseteq w}} \delta^s v \ , \end{aligned}$$

since $(\delta^t w)_{r,u} = \mathbf{1}$ iff $t = r$ and $w = u$.

From this, rules (4,5) follow easily. This finishes the proof of this fundamental theorem. □

Notation. From now on we shall work with quotient dioid $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ *and simply write x instead of* $[x]$. With this in mind, for $x \in \mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ a trajectory,

$$x_{t,w} = \mathbf{1} \text{ iff word } w \text{ is completed at the } \textit{earliest} \text{ at time } t \text{ for trajectory } x. \quad (6)$$

This is in agreement with requirement (1) (compare with (2)).

As a trajectory of a timed diagram is represented by an element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$, the next question is : how to encode *all* the trajectories of this automaton in a finitary manner ? This is addressed next.

Modelling elapsed time for graph automata. This coding is based on the following simple remarks:

(i) Given $x \in \mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$, then $y = \delta x$ is the unique element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ such that $y_{t+1,w} = \mathbf{1} \quad \text{iff} \quad x_{t,w} = \mathbf{1}$
meaning that (pre– or post–)multiplying an element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ by δ amounts to delaying time by one unit.

(ii) Given $x \in \mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$, then $y = x.v$ is the unique element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ such that $y_{t,wv} = \mathbf{1} \quad \text{iff} \quad x_{t,w} = \mathbf{1}$
meaning that postmultiplying an element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ by v amounts to postfixing the trajectory by word v without incrementing the date.

(iii) We denote by $\mathbf{0}$ the unique element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ having all its coefficients equal to $\mathbf{0}$, $\mathbf{0} \in \mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ encodes the trajectory with no event at all. Also we denote by $\mathbf{1}$ the unique element of $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ having a coefficient $\mathbf{1}$ for its monomial $\delta^0 \varepsilon$ (ε being the empty word), and $\mathbf{0}$ otherwise.

With this in mind, we are ready to proceed on our coding. Consider the following row vector of formal power series in $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$: $\Xi = \left[\mathtt{X}_{S_1}\ \mathtt{Y}_{S_1}\ \mathtt{X}_{S_2}\ \mathtt{Y}_{S_2}\right]$,

where the entries $\mathtt{X}_{S_1}, \mathtt{Y}_{S_1}, \mathtt{X}_{S_2}, \mathtt{Y}_{S_2} \in \mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ encode the trajectories of flows `X` and `Y` when hitting states S_1 and S_2 respectively, cf. (6).

Keeping points (i), (ii), and (iii) in mind, we get that Ξ is solution of the following fixpoint equation in $\mathcal{M}_{\text{in}}^{\text{ax}}(\Sigma, \delta)$ (the constant vector codes the initial state of the system):

$$\Xi = \Xi \begin{bmatrix} \mathbf{0} & \mathbf{0} & \delta^2 A & \delta^2 A \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta A \\ \delta^2 C & \mathbf{0} & \delta^2 B & \mathbf{0} \\ \delta^2 C & \delta C & \mathbf{0} & B \end{bmatrix} \bigoplus \left[\mathbf{1}\,\mathbf{1}\ \mathbf{0}\,\mathbf{0}\right] \tag{7}$$

Expanding (7), we obtain:

$$\begin{cases} \mathtt{X}_{S_1} = \mathtt{X}_{S_2}\ \delta^2 C \oplus \mathtt{Y}_{S_2}\ \delta^2 C \oplus \mathbf{1} \\ \mathtt{X}_{S_2} = \mathtt{X}_{S_1}\ \delta^2 A \oplus \mathtt{X}_{S_2}\ \delta^2 B \\ \mathtt{Y}_{S_1} = \mathtt{Y}_{S_2}\ \delta C \oplus \mathbf{1} \\ \mathtt{Y}_{S_2} = \mathtt{X}_{S_1}\ \delta^2 A \oplus \mathtt{Y}_{S_1}\ \delta A \oplus \mathtt{Y}_{S_2}\ B \end{cases}$$

3.3 Modelling multiform time elapsing

In this subsection, we revisit the example of the preceding section. Now we wish to consider that events involving flow `X` and events involving flow `Y` are measured using *different* time units. For instance, we may know that these two types of events take different amount of time, but we don't know yet how much each one would take. Thus the idea is to take different symbols to measure time for different flows. This is an example of dealing with *multiform time.* This is illustrated in Figure 4. Referring to Figure 2, just model time subsumption for grey and block patches via two independent symbols $\delta_{\mathtt{X}}$ and $\delta_{\mathtt{Y}}$ respectively. This leads to the timed diagram with multiform time below. As a justification, replace in $\mathcal{M}(\Sigma, \delta)$ the single symbol δ by a finite time alphabet Δ ($= \{\delta_{\mathtt{X}}, \delta_{\mathtt{Y}}\}$ in our example). Then replace in formula (3) the domain $s \geq t$ for the $\oplus$ by $\alpha \sqsupseteq \beta$, for computing in $[x]$ the coefficient associated with symbol $\beta w \in \Delta^* \times \Sigma^*$. Theorem 1 can be easily extended to this case.

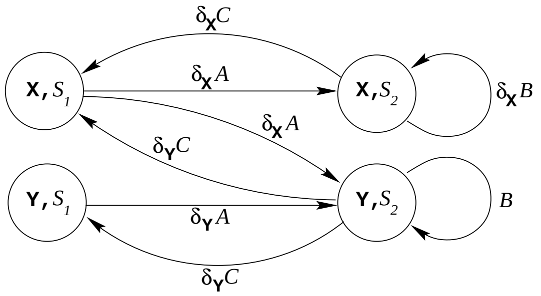

Fig. 4. *Multiform time fixpoint equations as flow graphs in* $\mathcal{M}_{\mathrm{in}}^{\mathrm{ax}}(\Sigma, \delta)$.

3.4 From fixpoint equations to regular expressions

We will only discuss uniform time case, the extension to multiform time being immediate.

Looking carefully at the above example, it is clear that model (7) is too verbose: the system matrix in (7) is very sparse, involving mostly $\mathbf{0}$'s. In the same vein, this representation explicitly uses states, while is it known that Kleene-Schützenberger formal power series for regular languages don't. Thus we should be able to do the same. In fact, this is easy.

Decompose Ξ in (7) as

$$\Xi = \begin{bmatrix} \Xi_1 \ \Xi_2 \end{bmatrix} \ , \text{ where, for } i = 1, 2 : \Xi_i = \begin{bmatrix} \mathtt{X}_{S_i} \ \mathtt{Y}_{S_i} \end{bmatrix}$$

We can rewrite fixpoint equation (7) as follows:

$$\Xi_1 = \Xi_2 \begin{bmatrix} \delta^2 C & \mathbf{0} \\ \delta^2 C & \delta C \end{bmatrix} \oplus \begin{bmatrix} \mathbf{1} \ \mathbf{1} \end{bmatrix} \tag{8}$$

$$\Xi_2 = \Xi_1 \begin{bmatrix} \delta^2 A & \delta^2 A \\ \mathbf{0} & \delta A \end{bmatrix} \oplus \Xi_2 \begin{bmatrix} \delta^2 B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \tag{9}$$

Substituting (9) in (8) and using the star operator twice yields:

$$\Xi_1 = \Xi_1^{\mathrm{init}} \mathcal{A} \ , \text{ where } \Xi_1^{\mathrm{init}} = [\mathbf{1} \ \mathbf{1}] \ , \text{ and}$$

$$\mathcal{A} = \left(\begin{bmatrix} \delta^2 A & \delta^2 A \\ \mathbf{0} & \delta A \end{bmatrix} \begin{bmatrix} \delta^2 B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}^{\star} \begin{bmatrix} \delta^2 C & \mathbf{0} \\ \delta^2 C & \delta C \end{bmatrix} \right)^{\star} =_{\mathrm{def}} (\mathbf{A}_\delta \mathbf{B}_\delta^{\star} \mathbf{C}_\delta)^{\star} \ , \tag{10}$$

where $\mathbf{A}_\delta$, etc., denote the corresponding polynomial matrices involved in formula (10).

The intuitive interpretation of these equations is the following. Assuming initial and terminal state is S_1, as before, the language corresponding to automaton of the Figure 2 is $(AB^\star C)^\star$. Getting the corresponding timed model is performed by 1/ associating with each action, say A, its corresponding timed version in the form of polynomial matrix $\mathbf{A}_\delta$, and, 2/ substituting A by $\mathbf{A}_\delta$, and so on, in the regular expression defining the considered language of actions.

It is easy to derive from (10) a rational expression for the different entries of Ξ. In fact, the above discussion is merely an illustration of the weak version of the Kleene-Schützenberger theorem [5] that holds in the semiring $\mathcal{L}$: the series that code the timed behavior of graph automata are exactly the rational series in $\mathcal{L}$, i.e. the series given by finite expressions involving the operators $\oplus, \otimes, *$ and monomials.

4 Applications: exchanging actions and time

This technique relies on the automaton of Figure 3, for which we have the following theorem:

Theorem 2 (retiming). *Both diagrams of Figure 5 represent the same transition matrix.*

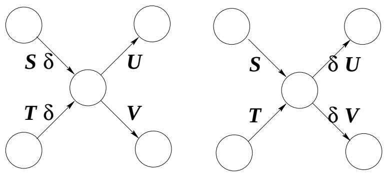

Fig. 5. *The basic rule of retiming.* Labels S, T, U, V, δ represent elements of $\mathcal{M}_{\rm in}^{\rm ax}(\Sigma, \delta)$ attached to the associated branches of the diagrams.

The proof is obvious, as both diagrams represent the same transition matrix from source to sink states of the diagram.

Representation (7) of transition matrix $\boldsymbol{\mathcal{A}}$ has the following particular feature: each entry in $\boldsymbol{\mathcal{A}}$ has degree exactly *one* in the Σ action alphabet. This means that it is easy to read on (7) how much time it costs to perform an action or a sequence of actions.

If it is wanted to answer the symmetric question: *how many actions can I perform per unit of time, or per* 10 *units of time,* then the form (7) is no longer suitable, and we have, so to say, to "symmetrize" it. This will be achieved by exhibiting another representation, equivalent to (7), i.e., having the same solution for associated fixpoint equation, but whose entries are, either $\mathbf{0}$, or have exactly degree *one* wrt. symbol δ. This can be achieved by adding new states in the diagram as shown in Figure 6 (equivalent to Figure 3), in which we again use retiming.

The technique used here is clearly general. Indeed, a series $x \in \mathcal{L}$ can be identified to a subset of $\delta^* \times \Sigma^*$, namely $X = \{\delta^t w \mid x_{t,w} = \mathbf{1}\}$. Then, such

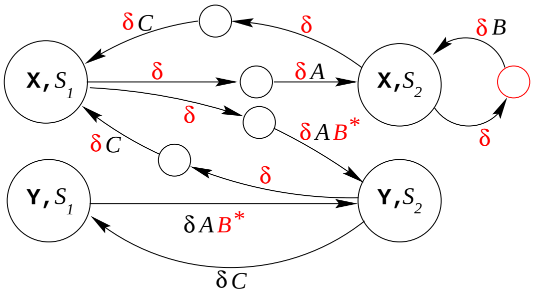

Fig. 6. *Exchanging time and actions in the diagram of Figure 3.* 1/ Check on diagram of Figure 3 which labels have degree zero wrt. δ : there is exactly one, attached to the bottom right circle and action B. 2/ Shift symbol B one step backward along the paths of the directed graph, and concatenate it to the corresponding labels. 3/ Remove arrows with no label. All arrows of the resulting diagram are labelled with power series in $\mathcal{M}_{\mathrm{in}}^{\mathrm{ax}}(\Sigma, \delta)$ of degree 1 with respect to δ : the diagram represents what actions can be performed within one time unit, starting from any state of the machine. In particular, so-called time bounded safety properties ("nothing bad can happen within 10 time units") can be checked in this way.

a subset can be identified either to a function $\Sigma^* \to \mathcal{P}(\delta^*)$, $X'(w) = \{\delta^t \mid x_{t,w} = \mathbf{1}\}$, which to an event w, associates the set of legal time constraints, or to the (inverse) function $X'' : \delta^* \to \mathcal{P}(\Sigma^*)$, $X''(\delta^t) = \{w \mid x_{t,w} = \mathbf{1}\}$, which to a time instant, associates the set of legal event constraints at this time. Our problem simply consists in computing effectively X'' from x. When the series $x \in \mathcal{L}$ is rational, which is the case if x is produced by a graph automaton model, the series X'' is *recognizable* [5] over the semiring of ordinary rational subsets of Σ^*, i.e. we can find a matrix μ, a row vector λ, and a column vector γ with compatible sizes, whose entries are ordinary rational expressions in Σ^*, such that $X'' = \lambda(\mu\delta)^*\gamma$. In particular, the set of admissible event constraints at time t is nothing but the coefficient of this series at δ^t, namely $\lambda\mu^t\gamma$. As discussed before, this is a basic tool for performing so-called bounded time safety properties, i.e., checking if "nothing bad" can happen within e.g. 10 units of time.

5 Structural modelling of mixed dataflow/state based timed systems

In this section, we introduce a useful model of mixed dataflow/state-based systems, which is essentially derived from [15] (similar efforts are currently underway around Ptolemy's group [12], but with a slightly different semantics). Then we show how to perform a structural translation of a timed version of such models into our formalism.

5.1 Hierarchical Mode Machines (hMM)

Mode machines mix in a simple way dataflow diagrams and automata [15] [12]. A hierarchical mode machine (hMM) has the form shown in Figure 7.

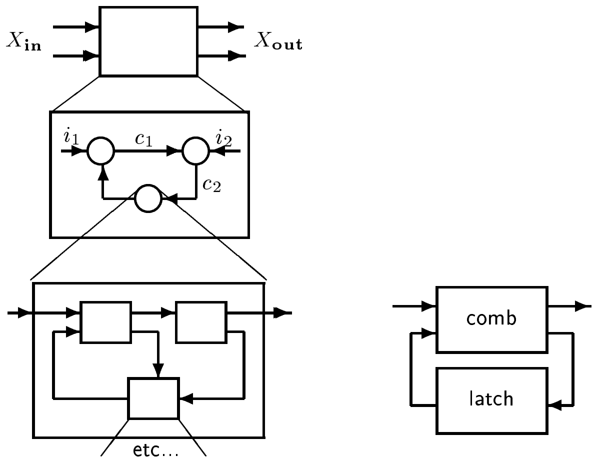

Fig. 7. *A view of* hMM. hMM are nested dataflow diagrams and automata. At the lowest level (right hand side) we simply consider simple dataflow diagrams equivalent to synchronous circuits: the circuit has a (circuitfree) combinational part, in feedback loop with a set of latches. Then, the diagram on the left hand side shows how successive levels are refined. In particular, the mid-diagram shows that a box of a dataflow diagram is refined into a state machine, and that, in each state, a dataflow diagram of lower level is activated, meaning that the input/output map specified by the top-box is refined into different diagrams, for each different state of the automaton.

hMM's can be flatened and expanded into a single (huge) synchronous circuit, see [15] for how to derive a LUSTRE program out of an hMM. Actually, as we only rely, for our coding into timed diagrams, on the partial orders of events specified by such hMM's, we can as well consider a token based dataflow semantics following [12].

Our aim is to show that timed hMM have a nice structural coding into our framework. For this we first need to slightly adapt our use of timed diagrams.

5.2 Capturing actions, counters, and daters

Until now, only actions and time were considered. In this model however, it appears a new concept: the notion of logical time implemented by latches and captured by counters. Our situation is the following. We have:

- a finite set of *flows,* denoted $\mathtt{X}, \mathtt{Y}, \ldots$. To flow $\mathtt{X}$ we attach
  - a *counting* symbol $\mu_{\mathtt{X}}$, and
  - a *dating* symbol $\delta_{\mathtt{X}}$. The alphabet of the $\mu_{\mathtt{X}}$'s, where $\mathtt{X}$ ranges over some collection $\mathbf{X}$ of flows, is denoted by $\boldsymbol{\mu}_{\mathbf{X}}$ or simply $\boldsymbol{\mu}$, while the alphabet of the $\delta_{\mathtt{X}}$ is denoted by $\Delta_{\mathbf{X}}$, or simply Δ.
- a finite alphabet of *actions* as before, we denote it by A.

As flow occurrences are bound to the actions performed, we handle actions and counters in the same way, by considering the product alphabet $\Sigma = A \times \boldsymbol{\mu}$. Hence we are back to our general framework and thus shall work in dioid $\mathcal{M}_{\mathrm{in}}^{\mathrm{ax}}(\Sigma, \Delta)$ as before. An example of how to use this framework is shown in Figure 8.

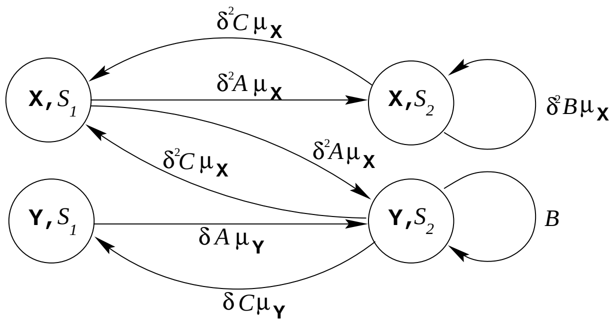

Fig. 8. *Capturing actions, counters, and daters on the example of figure 3.*

5.3 Structural modelling of timed hMM

We use the framework of subsection 5.2. We first model the primitives, and then, by structural induction, whole systems. To avoid the burden of generic notations, we show one illustrative example for each case, see figures 9 to 12. In each figure, we depict both the hMM model in consideration, and its structural translation into a timed diagram. What counter/action and dater symbols should be used in the resulting model can be inferred from refinements shown in figures 9 (where dater symbols are introduced), 10 (where counter symbols are introduced), and 11 (where action symbols are introduced).

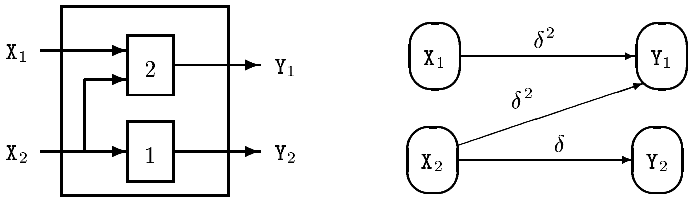

Fig. 9. *Encoding a combinational circuit.* Square boxes indicate "computations". Labels 1 and 2 in the boxes refer to the corresponding latency for each box, evaluated in clock cycles. The translation into a timed diagram is shown on the right hand side, where symbol δ is used to date events according to clock cycles.

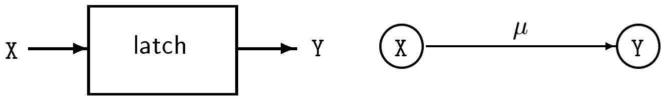

Fig. 10. *Encoding the latch* $\forall n \ : \ Y_n = X_{n-1}$. Time index "$n$" is shared by the two flows X, Y. Corresponding counting symbol μ counts successive occurrences for X or Y, equivalently.

6 Related work

As our model captures concurrency, we shall only discuss relations with other approaches having this feature also.

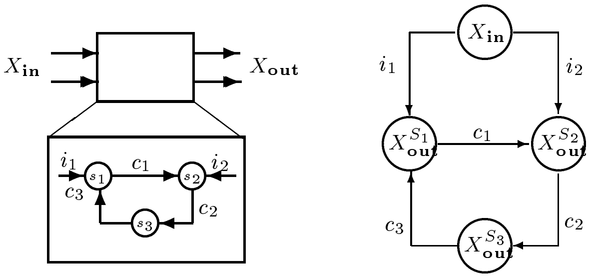

Fig. 11. *Refining an automaton.* Labels c_1, c_2, c_3 denote actions that trigger state transitions. Labels i_1, i_2 denote actions which specify in which state the automaton is entered when activated. Note that we have refined output formal power series into three series, one for each different state. Thus $\mathtt{X}_{\mathbf{out}} = \left[\, \mathtt{X}^{S_1}_{\mathbf{out}}\ \mathtt{X}^{S_2}_{\mathbf{out}}\ \mathtt{X}^{S_3}_{\mathbf{out}} \,\right]$.

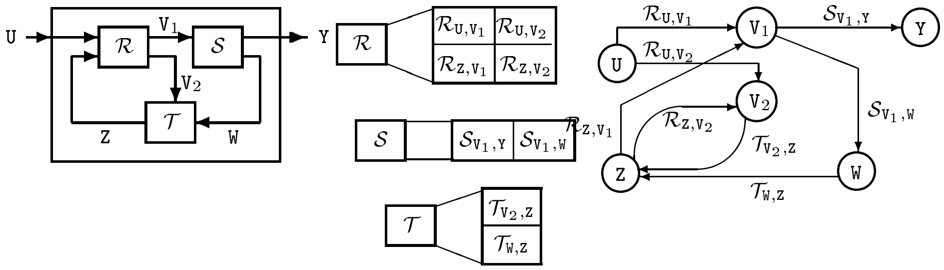

Fig. 12. *Refining a dataflow diagram.* Symbols $\mathcal{R}, \mathcal{S}, \mathcal{T}$ refer to matrices of appropriate dimensions, the entries of which are regular expressions. These matrices are partitioned as indicated on the top right hand side of the picture. Corresponding refined timed diagram is shown on the bottom diagram.

As timed Petri Nets are first natural candidates, it is worth discussing relations to such models. It is known that the mutual exclusion mechanism in PN's cannot be directly captured via maxplus algebra. Indeed, the widest class of timed PN considered so far is restricted to the case of *free choice* nets [16], handled with a combination of two heterogeneous maxplus theories. Our model does not capture mutual exclusion either, it is however fitted to capture performance evaluation of systems involving concurrency; on the other hand, our maxplus algebra is homogeneous as it involves a unique dioid, not a combination of two different ones.

Another interesting related work is the more recent one [17] [18]. In this work timed (and actually hybrid) automata are considered together with composition rules defined via general synchronisation modes for timed actions. Such general modes involve the **and**-synchronisation (i.e., conjunction), the **max**-synchronisation, and the **min**-synchronisation as particular cases. Our model corresponds to the particular case of the **max**-synchronisation : as compared to the above reference, we provide an algebraic setting for this subclass, something not considered in the referred papers.

7 Conclusion and discussion

We have introduced a linear algebraic framework for a certain class of timed systems models. In contrast to more usual timed automata, which are dedicated to the specification and verification of timing properties of discrete systems, our framework has performance evaluation in its objectives.

We have illustrated how our framework can be used to address different problems related to performance evaluation. As this framework is new, we have only sketched a few tentative applications.

The interest of linking discrete system timing evaluation to "maxplus" type of approaches is manifold. First, we inherit the representation of timed transition systems via matrices in our dioid $\mathcal{M}_{\rm in}^{\rm ax}(\Sigma, \delta)$ of formal power series. As such matrices can be in particular multiplied to represent iterates of such transition systems, this gives rise to reachability analysis via symbolic techniques involving regular expressions in our algebra. This yields a more compact representation than via state enumeration, and not subject to approximation such as those recently proposed via polyhedron approximation techniques [14]. Efficiency of our technique has to be further explored.

Second, we hope to take advantage of the techniques developed in [1, 7–9, 11] for the asymptotic performance analysis of timed systems[1], which use in particular the max-plus spectral theory. Indeed, the rational series of $\mathcal{L}$ are special (monotone) recognizable (or rational) dater functions in the sense of [7], hence the performance evaluation techniques of [7] can be applied to this case. In particular, worst case measures of performance can be computed in a time that is polynomial in the size of the defining automaton. We should stress, however, that by comparison with [7], the dioid $\mathcal{M}_{\rm in}^{\rm ax}(\Sigma, \delta)$ implements the *new* simplification rule (5), hence some additional work is needed to incorporate this rule, in order to obtain improved algorithms. The exact location of this new dioid in the Max-Plus mathematical scenery is detailed in [3].

Third, we have proposed a graphical notation on which many interesting questions translate as simple graphical manipulations. How this advantage can be turned into efficient algorithms has to be further investigated.

References

1. F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity.* Wiley, 1992.
2. A. Benveniste, G. Berry. Real-Time systems design and programming. *Another look at real-time programming*, special section of *Proc. of the IEEE,* 9(9):1270–1282, sep 1991.
3. A. Benveniste, S. Gaubert, and C. Jard. Algebraic techniques for timed systems : monotone rational series and max-plus models Rapport de recherche, INRIA, July 1998. In preparation.
4. J. Berstel. *Transductions and context-free languages,* Teubner, 1979.

[1] including, for example, maximal throughput.

5. J. Berstel, C. Reutenauer. *Les séries rationnelles et leurs langages,* Etudes et recherches en informatique, Masson, Paris, 1984.
6. G. Cohen, P. Moller, J-P. Quadrat, and M. Viot. Algebraic Tools for the Performance Evaluation of Discrete Event Systems. *Proc. of the IEEE*, 1989.
7. S. Gaubert. Performance evaluation of (max,+) automata. *IEEE Trans. on Automatic Control*, 40(12), Dec 1995.
8. S. Gaubert. Resource optimization and (min,+) spectral theory. *IEEE Trans. on Automatic Control*, 40(11), Nov. 1995.
9. S. Gaubert and J. Mairesse. Task resource systems and (max,+) automata. In J. Gunawardena, editor, *Idempotency*, Publications of the Newton Institute. Cambridge University Press, 1996.
10. S. Gaubert, J. Mairesse. Modelling and Analysis of Timed Petri Nets using Heaps of Pieces, Submitted. Abridged version in the proc. of the ECC'97, Bruxelles, 1997.
11. S. Gaubert and Max Plus. Methods and applications of (max,+) linear algebra. Rapport de recherche 3088, INRIA, Jan 1997.
12. Alain Girault, Bilung Lee, and E. A. Lee, A Preliminary Study of Hierarchical Finite State Machines with Multiple Concurrency Models, Memorandum UCB/ERL M97/57, Electronics ResearchLaboratory, U. C. Berkeley, August 1997.
13. N. Halbwachs. *Synchronous programming of reactive systems,*. Kluwer Academic Pub., 1993.
14. N. Halbwachs, Y.E. Proy and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11(2): 157–185, Aug 1997.
15. F .Maraninchi, Y. Rémond Mode-Automata: About Modes and States in Reactive Systems. Research Report, Verimag, 1997.
16. F. Baccelli, S. Foss and B. Gaujal, Free-choice Petri Nets — an algebraic approach, *IEEE Trans. on Automatic Control,* vol AC-41, No12, Dec 1996, 1751-1778.
17. J. Sifakis and S. Yovine, Compositional specification of timed systems. In *13th annual symposium on Theoretical Aspects of Computer Science,* STACS'96, 347-359. LNCS 1046, Springer-Verlag, 1996.
18. S. Bornot and J. Sifakis, Relating time progress and deadlines in hybrid systems. In *Int. Workshop HART'97*, 286-300, Grenoble, France, March 1997. LNCS 1201, Springer-Verlag.
19. Handbook of TCS. Jan Van Leeuwen, editor. Volume B. Formal models and semantics. Chap. 3. Formal languages and power series, by A. Salomaa.