Project 22703: SYRF

Deliverable 5.2.1

Strong synchrony/true concurrency

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 MaxPlus algebraic techniques to handle more realistic types of real-time systems. In particular, our framework encompasses timed Petri Nets (with otherwise no restriction), 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 propose a new dioid of formal power series à la Kleene-Schützenberger, which has both 1/ formal power series corresponding to regular languages, and 2/ MaxPlus power series associated to timed event graphs by Cohen & Quadrat, as subcases. We provide the associated linear algebraic setting, together with a graphical notation for corresponding systems. We sketch how this framework can be used to symbolically solve several problems of interest, related to real-time systems.

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:

- 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.
- 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.
- Tasks need resources for their completion (they may be inputs or external signals, or alternatively physical resources for their execution), and they wait until all resources needed are available.

Restrictions are listed next:

1. Both system and environment states are assumed to be *finitely* many. 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 watchdog/timeout are involved, then they need to be abstracted, typically in the form of a possible nondeterministic exception, prior to enter our framework.

With this general setup in mind, here follow a list of frameworks which we wanted to support with our approach.

Linear algebraic approach for Petri Nets. MaxPlus techniques have been first introduced by Cohen, Quadrat et al. in the eighties [1] for timed event graphs, a restricted class of Petri Nets in which conflicts of any type are forbidden : each place has at most one transition before it and one after it. Recently, Baccelli et al. have proposed an extension of this algebraic approach to handle timed free choice nets (FCN). FCN allow a restricted type of conflict : if two transitions share a common input place, then they have only this place as input. Our approach will allow to model general timed PNs.

Real-time computing and architectures. 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).

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).

Task schedulers in real-time kernels. Operating systems for real-time embedded applications often consist in simplified, specialized task schedulers offering restricted services with some kind of guaranteed performance. Knowing the time subsumption of each particular task and for a given task scheduler, the same questions are of interest as for the previous case.

Retiming and reasoning on pipelining for real-time embedded architectures. Real-time embedded architectures refer to the type of architecture that can be embedded on a real-time system with bounded memory and bounded response time requirements. Fine modelling of such architectures is often performed for ASICs at logical gate level. It is also of interest, however, to consider more abstract views in which basic computing actions are regarded as black-boxes with some time and memory needs, and controlled by some contoller. Some registers may still kept visible at this level of abstraction and it is of interest to formally model available possibilities for pipelining that are correct w.r.t. the desired overall behaviour. So-called "retiming" techniques have been proposed in particular by Leiserson et al. for the

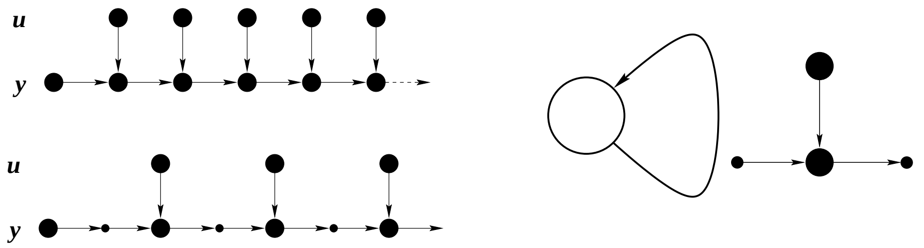

Figure 1: *Computation:* $\forall n\ y_n = y_{n-1} + u_n$*, modeled as a graph automaton.* The first diagram depicts the abstraction of the computation $\forall n : y_n = y_{n-1} + u_n$ in the form of an infinite string of dependence graphs. In this formula, u and y are *flows,* i.e., infinite sequences of data. In the second diagram, small bullets indicate the switching from one instant n to the next one. Use these small bullets 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.

particular case in which pure computations are performed, i.e., no controller enters the game.

These applications motivated us for developing a MaxPlus framework that would be general enough to support all of them. After having introduced our model, we shall discuss how this model fits the above applications. Then we shall sketch how the problems listed could be answered. Actual software to perform this is not available yet, however, hence we will not provide a detailed discussion but only the general idea. Questions of effectiveness and complexity of the proposed approach are deferred to a subsequent paper.

2 Discussing models of real-time computation

To introduce our model we shall discuss 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.

Assume that performing any event (thick bullet) takes δ cycles, where "cycle" denotes any given time unit. 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 after completing action A. One has

$$\begin{aligned} d'_{\mathtt{X}} &= d_{\mathtt{X}} + \delta \\ d'_{\mathtt{Y}} &= \max\left(d_{\mathtt{X}} + \delta\ ,\ d_{\mathtt{Y}} + \delta\right) \end{aligned} \tag{1}$$

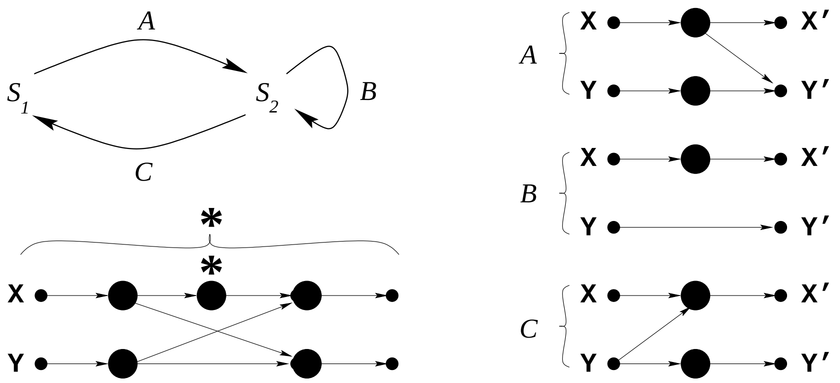

Figure 2: *The example.* The top left diagram depicts the automaton. Its action labels A, B, C are directed graphs connecting flow occurrences. These directed graphs are shown on the right hand side, they are interpreted as labelled partial orders. Pins are depicted by small bullets, while actual elementary computations (called "events" in the sequel) are depicted by thicker bullets. 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 X, as well as to the whole partial order shown.

Accept for the moment the game to follow: in formula (1), write $\oplus$ instead of "max", and $\otimes$ instead of $+$. Then (1) rewrites:

$$\begin{array}{rcl} d'_{\mathtt{X}} & = & d_{\mathtt{X}} \otimes \delta \\ d'_{\mathtt{Y}} & = & (d_{\mathtt{X}} \otimes \delta) \oplus (d_{\mathtt{Y}} \otimes \delta) \end{array} \tag{2}$$

Now, (2) looks like a formula in linear algebra, besides the fact that the $(\oplus, \otimes)$ operations are not the usual $(+, \times)$ ones. To justify what seems like an abuse of notation, let us introduce a glimpse of so-called MaxPlus algebra.

A glimpse of MaxPlus algebra. We consider the $\{\mathbf{Z}, \max, \text{plus}\}$ algebra, which we write $\{\mathbf{Z}, \oplus, \otimes\}$. This algebra is by definition the set $\mathbf{Z} \cup \{-\infty\}$, equipped with its laws "max" (denoted by $\oplus$) and $+$ (denoted by $\otimes$), e.g, $2 \otimes 1 = 3, 2 \oplus -1 = 2$. Operations

Basically, it is like the ordinary $\mathbf{Z}$ with its ring structure, except that one cannot use substraction. Units are

$$\text{for } \oplus \ : \ -\infty, \text{ written } \mathbf{0} \tag{3}$$

$$\text{for } \otimes \ : \ 0, \text{ written } \mathbf{1} \tag{4}$$

We shall also use symbol

$$\delta \quad \text{to denote} \quad 1 \in \mathbf{Z} \tag{5}$$

in order to avoid confusing with the units of our dioid. Matrices can be used and their product be defined:

$$\begin{array}{rcl} (A \otimes B)_{ij} & =_{\text{def}} & \bigoplus_k A_{ik} \otimes B_{kj} \\ & = & \max_k \, [A_{ik} + B_{kj}] \end{array}$$

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

Modelling the system of Figure 2 using MaxPlus algebra, a first attempt. Consider again formulæ (2), we rewrite them using matrix notations, and writing $\mathtt{X}, \mathtt{X}'$ in lieu of $d_\mathtt{X}, d'_\mathtt{X}$ for the sake of simplicity:

$$\left[\begin{array}{cc} \mathtt{X}' & \mathtt{Y}' \end{array}\right] = \left[\begin{array}{cc} \mathtt{X} & \mathtt{Y} \end{array}\right] \underbrace{\left[\begin{array}{cc} \delta & \delta \\ \mathbf{0} & \delta \end{array}\right]}_{[A]}, \tag{6}$$

we denote by $[A]$ the MaxPlus matrix underbraced. Thus formula (6) models how time progresses while performing transition A in automaton of Figure 2. Similar formulæ can be written for transitions B and C. The bad news is that there is no immediate way to encode at once how time progresses while the automaton is running. In particular, the tricky point is how to capture that moving to state S_2 can occur while originating from state S_2 itself (by performing action B) or state S_1 (by performing action A). Alas, a naive writing such as

$$\left[\begin{array}{cc} \mathtt{X}'_{S_2} & \mathtt{Y}'_{S_2} \end{array}\right] = \left[\begin{array}{cc} \mathtt{X}_{S_1} & \mathtt{Y}_{S_1} \end{array}\right] [A] \oplus \left[\begin{array}{cc} \mathtt{X}_{S_2} & \mathtt{Y}_{S_2} \end{array}\right] [B]$$

would not do, since it would model waiting for the completion of the latest transition time, while a disjuncting choice was indeed the intention (perform either A or B, not waiting for the latest).

What we need instead is a way to encode, within a unique framework, both actions and time elapsed. This is the subject of the section to follow.

3 A new MaxPlus algebra

3.1 Modelling time elapsing

In this subsection, we focus on the example of the preceding section. This example involves quantitative time measured with a *single* time unit, say, the machine "cycle", or the microsecond, or...

A rough dioid of formal power series. Consider the set $\mathcal{L}_0$ of formal power series

- with coefficients belonging to the boolean dioid ($\{0, 1\}$, equipped with operations (max,min) as $\oplus, \otimes$ respectively) — in the sequel, to be consistent with our generic dioid notation, we write $\{\mathbf{0}, \mathbf{1}\}$ instead of $\{0, 1\}$;
- with variables

$$\begin{array}{rcl} \delta & : & \text{to encode the dates} \\ A, B, C & : & \text{to encode the moves of the automaton} \end{array}$$

  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, such formal power series are written

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

for convenience. The idea is that formal series x models *timed runs,* i.e., languages together with the time needed to complete words. In $\mathcal{L}_0$, having a $\mathbf{1}$ for coefficient $x_{t,w}$ is intended to encode the completion of word w at time t for run x. Then $\mathcal{L}_0$ inherits the following dioid structure from that of its coefficients:

- $x \oplus y$ is obtained by taking the $\oplus$ componentwise.
- $x \otimes y$ is the Cauchy product defined by

$$(x \otimes y)_{t,w} \;=\; \bigoplus_{\substack{r+s=t \\ uv=w}} (x_{r,u} \otimes y_{s,v})$$

We shall denote by $\mathbf{0}$ the element of $\mathcal{L}_0$ having all its coefficients equal to $\mathbf{0}$: $\mathbf{0} \in \mathcal{L}_0$ represents the run with no event at all.

The quotient dioid $\mathcal{L}$. Here we follow the beautiful idea [1, 2] from Cohen and Quadrat to derive symmetric codings for event graphs. The case considered in [1, 2] corresponds to our situation for the particular case in which the automaton has a single state and a single action.

What we seek for is that symbol $\delta^t w$ should rather encode statement

$$\delta^t w \quad : \quad \text{“}t \text{ is the earliest possible date for } w \text{ being completed”} \tag{7}$$

This is equivalently rephrazed as

- when w completed, on cannot have a date $s, s < t$,
- at time 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 “subword”, not “prefix”, the reason for this will become clear soon.

We are now ready to formally construct the quotient dioid which provides this coding (7). For

$$x \;=\; \sum_{t\in\mathbf{Z}\,,\,w} x_{t,w} \,.\, \delta^t w$$

we set

$$[x] \;=\; \sum_{t\in\mathbf{Z}\,,\,w} \delta^t w \,.\, \bigoplus_{\substack{s \geq t \\ v \sqsubseteq w}} x_{s,v} \tag{8}$$

Then, the following holds:

Theorem 1 (the $\mathcal{L}$ quotient dioid)

1. *The relation*

$$x \sim y \quad \textit{iff} \quad [x] = [y]$$

is an equivalence relation.

2. *The following holds:*

$$[x \oplus y] = [x] \oplus [y] \quad , \quad [x \otimes y] = [x] \otimes [y] \tag{9}$$

which implies that equivalence relation $\sim$ is compatible with the dioid structure of $\mathcal{L}_0$. Thus we can consider the dioid $\mathcal{L}$ obtained by taking the quotient of dioid $\mathcal{L}_0$ by the $\sim$ equivalence relation.

3. *The following formula holds:*

$$\left[\delta^t w\right] \quad = \quad \bigoplus_{\substack{s \leq t \\ v \sqsupseteq w}} \quad \delta^s v \ . \tag{10}$$

4. *On the quotient dioid $\mathcal{L}$, the monomials obey the following rules:*

$$\delta^s w \oplus \delta^t w \quad = \quad \delta^{\max(s,t)} w \tag{11}$$
$$\delta^t v \oplus \delta^t w \quad = \quad \delta^t \min(v, w) \tag{12}$$

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

COMMENT: Points 3 and 4 of theorem 1 enlight that dioid $\mathcal{L}$ really implements specification (7).

Proof: Point 1 is obvious, we move on proving formulæ (9).
For the first one,

$$\begin{aligned}[x \oplus y] \quad &= \quad \sum_{t,w} \delta^t w \, . \bigoplus_{s \geq t \, , \, v \sqsubseteq w} [\ x_{s,v} \oplus y_{s,v}\ ] \\ &= \quad [x] \oplus [y]\end{aligned}$$

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

$$x \otimes y \quad = \quad \sum_{t,w} \delta^t w \, . \quad \bigoplus_{\substack{r + s = t \\ uv = w}} \quad [\ x_{r,u} \otimes y_{s,v}\ ] \ ,$$

then it holds that

$$\begin{aligned}[x \otimes y] &= \sum_{t,w} \delta^t w \,. \bigoplus_{\substack{t' \geq t \\ w' \sqsubseteq w}} \bigoplus_{\substack{r' + s' = t' \\ u'v' = w'}} [\, x_{r',u'} \otimes y_{s',v'} \,] \\ &= [x] \otimes [y]\end{aligned}$$

where the last equality follows from the fact that the following two sets are equal:

$$\{(u', v') \ : \ u'v' \sqsubseteq w\} \quad = \quad \{(u', v') \ : \ u' \sqsubseteq u \,,\, v' \sqsubseteq v \,,\, uv = w\}$$

Note that this latter property would be *false* if we had chosen "is prefix of" instead of "is a subword of". This finishes the proof of point 2 of the theorem.

We move to points 3 and 4. We have:

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

since

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

From this, rules (11,12) follow easily. This finishes the proof of this fundamental theorem. $\diamond$

From now on a run of one particular flow `X` or `Y` of the graph automaton of Figure 2 shall be represented by an element of $\mathcal{L}$. The next question is: how to encode *all* runs for both flows `X` and `Y` 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 remark:

1. Given $x \in \mathcal{L}$, then $y = \delta x$ is the unique element of $\mathcal{L}$ 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 $\mathcal{L}$ by δ amounts to delaying time by one unit.

2. Given $x \in \mathcal{L}$, then $y = x.v$ is the unique element of $\mathcal{L}$ such that

   $$[y]_{t,wv} = \mathbf{1} \quad \text{iff} \quad [x]_{t,w} = \mathbf{1}$$

   meaning that postmultiplying an element of $\mathcal{L}$ by w amounts to postfixing the run by word v without incrementing the date.

3. We denote by $\mathbf{0}$ the unique element of $\mathcal{L}$ having all its coefficients equal to $\mathbf{0}$, $\mathbf{0} \in \mathcal{L}$ encodes the run with no event at all.

With this in mind, we are ready to proceed on our coding.

Consider the following row vector of formal power series in $\mathcal{L}$:

$$\Xi \;=\; \left[\begin{array}{cccc} \mathtt{X}_{S_1} & \mathtt{Y}_{S_1} & \mathtt{X}_{S_2} & \mathtt{Y}_{S_2} \end{array}\right], \tag{13}$$

where the entries $\mathtt{X}_{S_1}, \mathtt{Y}_{S_1}, \mathtt{X}_{S_2}, \mathtt{Y}_{S_2} \in \mathcal{L}$ encode the runs of flows $\mathtt{X}$ and $\mathtt{Y}$ when hitting states S_1 and S_2 respectively.

Keeping points 1, 2, and 3 in mind, we get that Ξ is solution of the following fixpoint equation in $\mathcal{L}$:

$$\Xi \;=\; \Xi \left[\begin{array}{cccc} \mathbf{0} & \mathbf{0} & \delta A & \delta A \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta A \\ & & & \\ \delta C & \mathbf{0} & \delta B & \mathbf{0} \\ \delta C & \delta C & \mathbf{0} & B \end{array}\right] \bigoplus \left[\begin{array}{cccc} \delta^0 & \delta^0 & \mathbf{0} & \mathbf{0} \end{array}\right] \tag{14}$$

$$\Psi \;=\; \Xi \left[\begin{array}{cccccc} \delta A & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \delta A & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ & & & & & \\ \mathbf{0} & \mathbf{0} & \delta B & \mathbf{0} & \delta C & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta C & \delta C \end{array}\right] \tag{15}$$

Equations (14) and (15) respectively represent the fixpoint equation satisfied by the formal power series associated with the pins in Figure 2, and the equation defining the formal power series associated with the dates of the actual events $\mathtt{X}$ and $\mathtt{Y}$ when transitions A, B, C are performed:

$$\left[\begin{array}{cccc} \mathbf{0} & \mathbf{0} & \delta A & \delta A \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta A \\ & & & \\ \delta C & \mathbf{0} & \delta B & \mathbf{0} \\ \delta C & \delta C & \mathbf{0} & B \end{array}\right] \;\leftrightarrow\; \text{transition}: \left[\begin{array}{cc} S_1 \rightarrow S_1 & S_1 \rightarrow S_2 \\ & \\ S_2 \rightarrow S_1 & S_2 \rightarrow S_2 \end{array}\right]$$

$$\Psi \;\leftrightarrow\; \left[\begin{array}{ccc} \text{runs of } A & \text{runs of } B & \text{runs of } C \end{array}\right]$$

Note that "runs of A" has two columns, one for $\mathtt{X}$ and one for $\mathtt{Y}$, and similarly for other events.

3.2 Modelling multiform time elapsing

In this subsection, we revisit the example of the preceding section. Now we wish to consider that events involving flow $\mathtt{X}$ and events involving flow $\mathtt{Y}$ 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,* a notion which we will discuss next systematically.

A rough dioid of formal power series for multiform time. Consider the set $\mathcal{L}_0$ of formal power series

- with coefficients belonging to the boolean dioid ($\{0,1\}$, equipped with operations (max,min) as $\oplus, \otimes$ respectively) — in the sequel, to be consistent with our generic dioid notation, we write $\{\mathbf{0},\mathbf{1}\}$ instead of $\{0,1\}$;
- with alphabets

  $$\begin{array}{lll} \Delta & : & \text{to encode the dates} \\ \Sigma & : & \text{to encode the actions} \end{array}$$

  Variables from Σ (resp. Δ) do not commute, a word built with these variables shall be denoted by the generic letter w (resp. δ). On the other hand, w and δ globally commute:

  $$w\delta = \delta w \ .$$

Following the usual notation, such formal power series are written

$$x \ = \ \sum_{\delta\,,\,w} x_{\delta,w} \ . \ \delta w \ , \quad \text{or} \ \sum_{\delta\,,\,w} \delta w \ . \ x_{\delta,w}$$

for convenience. In $\mathcal{L}_0$, having a $\mathbf{1}$ for coefficient $x_{\delta,w}$ is intended to encode the completion of word w at time δ for our automaton. Then $\mathcal{L}_0$ inherits the following dioid structure from that of its coefficients:

- $x \oplus y$ is obtained by taking the $\oplus$ componentwise.
- $x \otimes y$ is the Cauchy product defined by

  $$(x \otimes y)_{\delta,w} \ = \ \bigoplus_{\substack{\alpha\beta = \delta \\ uv = w}} x_{\alpha,u} \otimes y_{\beta,v}$$

We shall denote by $\mathbf{0}$ the element of $\mathcal{L}_0$ having all its coefficients equal to $\mathbf{0}$: $\mathbf{0} \in \mathcal{L}_0$ represents the run with no event at all. This can be summarized by the single formula:

$$\mathcal{L}_0 \ = \ \mathcal{B}\,\langle\!\langle \Delta^\star \times \Sigma^\star \rangle\!\rangle \tag{16}$$

where $\mathcal{B}$ denotes the boolean dioid.

Equivalently, we can also define $\mathcal{L}_0$ in a different, but equivalent, way.

$$\mathcal{L}_0 \ = \ \mathcal{P}\,(\Delta^\star \times \Sigma^\star) \tag{17}$$

The $\oplus$ is the union, and the product $L \otimes L'$ is defined by

$$L \otimes L' \ = \ \{(\delta\delta', ww') \,|\, (\delta, w) \in L\,,\, (\delta', w') \in L'\} \tag{18}$$

The quotient dioid $\mathcal{L}$ for multiform time. For

$$x \;=\; \sum_{\delta\,,\,w} x_{\delta,w}\;.\;\delta w$$

we set

$$[x] \;=\; \sum_{\delta\,,\,w} (\delta, w)\,.\; \bigoplus_{\substack{\delta' \sqsupseteq \delta \\ w' \sqsubseteq w}} x_{\delta',w'} \tag{19}$$

Then, the following holds:

Theorem 2 (the $\mathcal{L}$ quotient dioid)

1. *The relation*

   $$x \sim y \quad \textit{iff} \quad [x] = [y]$$

   is an equivalence relation.

2. *The following holds:*

   $$[x \oplus y] = [x] \oplus [y] \quad , \quad [x \otimes y] = [x] \otimes [y] \tag{20}$$

   which imples that equivalence relation $\sim$ is compatible with the dioid structure of $\mathcal{L}_0$. Thus we can consider the dioid $\mathcal{L}$ obtained by taking the quotient of dioid $\mathcal{L}_0$ by the $\sim$ equivalence relation.

3. *The following formula holds:*

   $$[(\delta, w)] \;=\; \bigoplus_{\substack{\alpha \sqsubseteq \delta \\ v \sqsupseteq w}} (\alpha, v)\ , \tag{21}$$

4. *On the quotient dioid $\mathcal{L}$, the monomials obey the following rules:*

   $$(\delta, w) \oplus (\delta', w) \;=\; (\max(\delta, \delta')\,,\, w) \tag{22}$$
   $$(\delta, v) \oplus (\delta, w) \;=\; (\delta\,,\, \min(v, w)) \tag{23}$$

   if one of the words v, w (resp. δ, δ') is a subword of the other one, and $\min(v, w)$ (resp. $\max(\delta, \delta')$) denotes this subword (resp. the longest word among δ, δ').

Comment: Points 3 and 4 of theorem 2 enlight that dioid $\mathcal{L}$ really implements specification (7).

Proof: Point 1 is obvious, we move on proving formulæ (20).

For the first one,

$$\begin{aligned}[x \oplus y] &= \sum_{\delta,w} (\delta, w) \,.\, \bigoplus_{\delta' \sqsupseteq \delta \,,\, w' \sqsubseteq w} [\; x_{\delta',w'} \oplus y_{\delta',w'} \;] \\ &= [x] \oplus [y]\end{aligned}$$

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

$$x \otimes y \;=\; \sum_{\delta,w} (\delta, w) \,.\, \bigoplus_{\substack{\alpha\beta = \delta \\ uv = w}} [\; x_{\alpha,u} \otimes y_{\beta,v} \;] \;,$$

then it holds that

$$\begin{aligned}[x \otimes y] &= \sum_{\delta,w} (\delta, w) \,.\, \bigoplus_{\substack{\delta' \sqsupseteq \delta \\ w' \sqsubseteq w}} \; \bigoplus_{\substack{\alpha'\beta' = \delta' \\ u'v' = w'}} [\; x_{\alpha',u'} \otimes y_{\beta',v'} \;] \\ &= [x] \otimes [y]\end{aligned}$$

where the last equality follows from the fact that the following two sets are equal:

$$\{(u', v') \;:\; u'v' \sqsubseteq w\} \;=\; \{(u', v') \;:\; u' \sqsubseteq u \,,\, v' \sqsubseteq v \,,\, uv = w\}$$

and, symmetrically,

$$\{(\alpha', \beta') \;:\; \alpha'\beta' \sqsupseteq \delta\} \;=\; \{(\alpha', \beta') \;:\; \alpha' \sqsupseteq \alpha \,,\, \beta' \sqsupseteq \beta \,,\, \alpha\beta = \delta\}$$

This finishes the proof of point 2 of the theorem.

We move to points 3 and 4. We have:

$$\begin{aligned}[(\delta, w)] &= \sum_{\alpha,v} (\alpha, w) \bigoplus_{\substack{\alpha' \sqsupseteq \alpha \\ u \sqsubseteq v}} \underbrace{(\delta w)_{\alpha',u}}_{\substack{= \mathbf{1} \text{ iff} \\ \delta = \alpha' \text{ and } w = u}} \\ &= \sum_{\alpha,v} (\alpha, w) \,.\, (\text{ if } \delta \sqsupseteq \alpha \text{ and } w \sqsubseteq v \;, \quad \text{then } \mathbf{1} \text{ else } \mathbf{0}) \\ &= \bigoplus_{\substack{\alpha \sqsubseteq \delta \\ v \sqsupseteq w}} (\alpha, v) \;,\end{aligned}$$

from which rules (22,23) follow easily. This finishes the proof of this fundamental theorem. $\diamond$

From now on a run of one particular chain of our partial order automaton shall be represented by an element of $\mathcal{L}$. The next question is: how to encode *all* runs of this automaton in a finitary manner? This is addressed next.

Encoding systems with multiform time. We take, as before,

$$\Sigma \;=\; \{A, B, C\} \tag{24}$$

But, now, instead of the unique symbol δ for time, we introduce the second alphabet

$$\Delta \;=\; \{\delta_{\mathtt{X}}, \delta_{\mathtt{Y}}\} \tag{25}$$

for multiform time — we distinguish time for each separated flow. For actions A, B, C corresponding to the graphs of figure 2, we have the following counterpart of formulæ (14,15):

$$\Xi \;=\; \Xi \begin{bmatrix} \mathbf{0} & \mathbf{0} & \delta_{\mathtt{X}} A & \delta_{\mathtt{X}} A \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta_{\mathtt{Y}} A \\ & & & \\ \delta_{\mathtt{X}} C & \mathbf{0} & \delta_{\mathtt{X}} B & \mathbf{0} \\ \delta_{\mathtt{Y}} C & \delta_{\mathtt{Y}} C & \mathbf{0} & B \end{bmatrix} \bigoplus \begin{bmatrix} \delta^0 & \delta^0 & \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{26}$$

$$\boldsymbol{\Psi} \;=\; \Xi \begin{bmatrix} \delta_{\mathtt{X}} A & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \delta_{\mathtt{Y}} A & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ & & & & & \\ \mathbf{0} & \mathbf{0} & \delta_{\mathtt{X}} B & \mathbf{0} & \delta_{\mathtt{X}} C & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta_{\mathtt{Y}} C & \delta_{\mathtt{Y}} C \end{bmatrix} \tag{27}$$

3.3 From fixpoint equations to regular expressions

The uniform time case. Looking carefully at the above example, it is clear that model (14,15) is too verbose: both system matrix in (14) and observation matrix in (15) are 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 (13) as

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

We can rewrite fixpoint equation (14) as follows:

$$\Xi_1 \;=\; \Xi_2 \begin{bmatrix} \delta C & \mathbf{0} \\ \delta C & \delta C \end{bmatrix} \oplus \begin{bmatrix} \delta^0 & \delta^0 \end{bmatrix} \tag{28}$$

$$\Xi_2 \;=\; \Xi_1 \begin{bmatrix} \delta A & \delta A \\ \mathbf{0} & \delta A \end{bmatrix} \oplus \Xi_2 \begin{bmatrix} \delta B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \tag{29}$$

Substituting (29) in (28) and using the star operator twice yields:

$$\Xi_1 \;=\; \Xi_1^{\text{init}} \mathcal{A} \text{ , where } \Xi_1^{\text{init}} = \begin{bmatrix} \delta^0 & \delta^0 \end{bmatrix} \text{ , and} \tag{30}$$

$$\begin{aligned} \mathcal{A} \;&=\; \left(\begin{bmatrix} \delta A & \delta A \\ \mathbf{0} & \delta A \end{bmatrix} \begin{bmatrix} \delta B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}^{\star} \begin{bmatrix} \delta C & \mathbf{0} \\ \delta C & \delta C \end{bmatrix} \right)^{\star} \\ &=_{\text{def}} \; (\mathbf{A}_\delta \mathbf{B}_\delta^{\star} \mathbf{C}_\delta)^{\star} \text{ ,} \end{aligned} \tag{31}$$

where $\mathbf{A}_\delta$, etc., denote the corresponding polynomial matrices involved in formula (31).

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.

Now, we did not consider yet that any particular terminal state was pointed. Hence, instead of considering the language $(AB^\star C)^\star$, we should rather take

$$(AB^\star C)^\star(\mathbf{1} \oplus A \oplus AB^\star) \tag{32}$$

where 1 denotes the language $\{\emptyset\}$, and corresponding system matrix

$$\mathcal{A} \;\; = \;\; (\mathbf{A}_\delta \mathbf{B}_\delta^\star \mathbf{C}_\delta)^\star(\mathbf{1} \oplus \mathbf{A}_\delta \oplus \mathbf{A}_\delta \mathbf{B}_\delta^\star) \ . \tag{33}$$

The corresponding equation defining the solution ξ of our fixpoint equation now reads:

$$\xi \;\; = \;\; \xi^{\text{init}} \, \mathcal{A} \ , \text{ where } \ \xi^{\text{init}} = \begin{bmatrix} \delta^0 & \delta^0 \end{bmatrix} \ . \tag{34}$$

Next, we must obtain the counterpart of observation equation (15). It is simply given by:

$$\psi \;\; = \;\; \xi \left(\mathbf{A}_\delta^{-1} \begin{bmatrix} \delta A & \mathbf{0} \\ \mathbf{0} & \delta A \end{bmatrix} \oplus \mathbf{B}_\delta^{-1} \begin{bmatrix} \delta B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \oplus \mathbf{C}_\delta^{-1} \begin{bmatrix} \delta C & \mathbf{0} \\ \delta C & \delta C \end{bmatrix} \right) \tag{35}$$

where $\xi \mathbf{A}_\delta^{-1}$, denotes the right quotient of power series ξ by $\mathbf{A}_\delta$, with corresponding definitions for other quotients. Recall the definition of the right quotient of a formal power series x by some element $\mathbf{A}_\delta$: [3]:

$$x\mathbf{A}_\delta^{-1} \;\; =_{\text{def}} \;\; \sum_{\mathbf{w}_\delta} \mathbf{w}_\delta \; x_{\mathbf{w}_\delta \mathbf{A}_\delta} \ .$$

Intuitively speaking, equation (35) means that actual dates of events are obtained by 1/ taking the power series associated with the pins, 2/ removing the effect of the last action, and 3/ replacing it by the time elapsed from the last pin to the considered event.

Discussion. What makes the move from (32) to (33) so "miraculous" is precisely the particular quotient diod of formal power series $\mathcal{L}$ we have introduced in Theorem 1 and its homomorphism property (11). Clearly, representation (33,34,35) is much more appealing than (14,15): the dimension of $\boldsymbol{\Xi}$ is equal to that of ξ times the number of states of the originally considered language of actions.

Summary. We summarize here the important formulae: for

$$L \;\; = \;\; (AB^\star C)^\star(\mathbf{1} \oplus A \oplus AB^\star) \ , \tag{36}$$

and actions A, B, C corresponding to the graphs of figure 2, we have

$$\mathcal{A} \;\; = \;\; (\mathbf{A}_\delta \mathbf{B}_\delta^\star \mathbf{C}_\delta)^\star(\mathbf{1} \oplus \mathbf{A}_\delta \oplus \mathbf{A}_\delta \mathbf{B}_\delta^\star) \tag{37}$$

$$\mathcal{C} = \begin{bmatrix} \begin{bmatrix} \delta A & \mathbf{0} \\ \mathbf{0} & \delta A \end{bmatrix} \\ \\ \begin{bmatrix} \delta B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\ \\ \begin{bmatrix} \delta C & \mathbf{0} \\ \delta C & \delta C \end{bmatrix} \end{bmatrix} \tag{38}$$

and

$$\begin{cases} \xi & = & \xi^{\text{init}} \mathcal{A} \\ \psi & = & \xi \begin{bmatrix} \mathbf{A}_\delta^{-1} & \mathbf{B}_\delta^{-1} & \mathbf{C}_\delta^{-1} \end{bmatrix} \mathcal{C} \end{cases} \tag{39}$$

where

$$\xi^{\text{init}} = \begin{bmatrix} \delta^0 & \delta^0 \end{bmatrix} . \tag{40}$$

Formula (39) is generic for any system having alphabet of actions $\{A, B, C\}$ (and is thus immediately generalized to an arbitrary alphabet of actions), and other formulae are specific to the example considered here.

The multiform time case. We take, as before,

$$\Sigma = \{A, B, C\} \tag{41}$$

But, now, instead of the unique symbol δ for time, we introduce the second alphabet

$$\Delta = \{\delta_{\mathtt{X}}, \delta_{\mathtt{Y}}\} \tag{42}$$

for multiform time — we distinguish time for each separated flow. For

$$L = (AB^\star C)^\star(\mathbf{1} \oplus A \oplus AB^\star) , \tag{43}$$

and actions A, B, C corresponding to the graphs of figure 2, we have

$$\mathcal{A} = (\mathbf{A}_\delta \mathbf{B}_\delta^\star \mathbf{C}_\delta)^\star(\mathbf{1} \oplus \mathbf{A}_\delta \oplus \mathbf{A}_\delta \mathbf{B}_\delta^\star) \tag{44}$$

$$\mathbf{A}_\delta = \begin{bmatrix} \delta_{\mathtt{X}} A & \delta_{\mathtt{X}} A \\ \mathbf{0} & \delta_{\mathtt{Y}} A \end{bmatrix} \tag{45}$$

$$\mathbf{B}_\delta = \begin{bmatrix} \delta_{\mathtt{X}} B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix} \tag{46}$$

$$\mathbf{C}_\delta = \begin{bmatrix} \delta_{\mathtt{X}} C & \mathbf{0} \\ \delta_{\mathtt{Y}} C & \delta_{\mathtt{Y}} C \end{bmatrix} \tag{47}$$

$$\mathcal{C} = \begin{bmatrix} \begin{bmatrix} \delta_{\mathtt{X}}A & \mathbf{0} \\ \mathbf{0} & \delta_{\mathtt{Y}}A \end{bmatrix} \\ \\ \begin{bmatrix} \delta_{\mathtt{X}}B & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \\ \\ \begin{bmatrix} \delta_{\mathtt{X}}C & \mathbf{0} \\ \delta_{\mathtt{Y}}C & \delta_{\mathtt{Y}}C \end{bmatrix} \end{bmatrix} \tag{48}$$

and

$$\left\{ \begin{array}{rcl} \xi & = & \xi^{\mathrm{init}}\, \mathcal{A} \\ \psi & = & \xi \begin{bmatrix} \mathbf{A}_\delta^{-1} & \mathbf{B}_\delta^{-1} & \mathbf{C}_\delta^{-1} \end{bmatrix} \mathcal{C} \end{array} \right. \tag{49}$$

where

$$\xi^{\mathrm{init}} = \begin{bmatrix} \delta^0 \ \delta^0 \end{bmatrix} . \tag{50}$$

Formula (49) is generic for any system having alphabet of actions $\{A, B, C\}$ (and is thus immediately generalized to an arbitrary alphabet of actions), and other formulae are specific to the example considered here.

4 Graphical notations for MaxPlus fixpoint equations

4.1 The graphical notation

The uniform time case. In this section we show how to represent regular expressions in $\mathcal{L}$ as solutions of fixpoint equations of the form (14) in a graphical way. This graphical representation, shown in figure 3, shall conveniently replace representation of figure 2. Its main advantage is that we can let each individual component of state vector Ξ progress independently, rather than in a synchronised way as in figure 2.

The multiform time case. The multiform extension to Figure 2 is shown in Figure 4.

4.2 Applications : a few basic exercises

4.2.1 Retiming

Retiming is just the "multiclock" extension of the familiar technique, due to Leiserson and Kung for the synthesis of pipelines and systolic architectures in the case of so-called "regular algorithms", in which algorithms are specified using recurrent equations with a single time index.

This technique relies on the diagrams of Figures 3 or 4, for which we have the following theorem :

Theorem 3 (retiming) *Both diagrams of figure 5 represent the same transition matrix.*

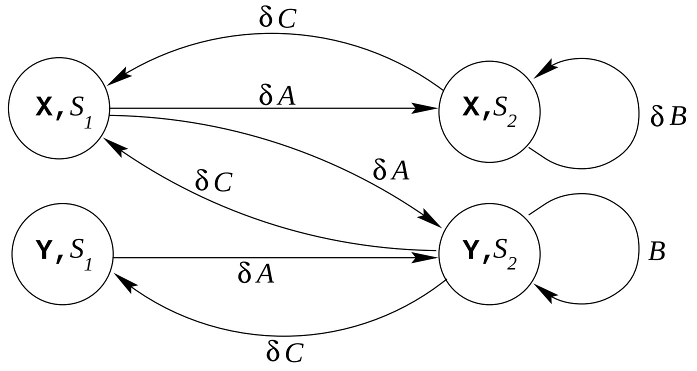

Figure 3: *Fixpoint equations as flow graphs in $\mathcal{L}$.* This diagram encodes fixpoint equation (14) as follows: the four circles represent the four components of Ξ. Labels on the arrows indicate taking the $\otimes$-product in $\mathcal{L}$ of the power series from which the arrow originates, with the element of $\mathcal{L}$ attached to the arrow. Then, the element of $\mathcal{L}$ sitting in a given circle is the $\oplus$-sum of its incoming arrows. For example, considering the top right circle, we can write $\mathtt{X}_{S_2} = \mathtt{X}_{S_1}\ \delta A \oplus \mathtt{X}_{S_2}\ \delta B$.

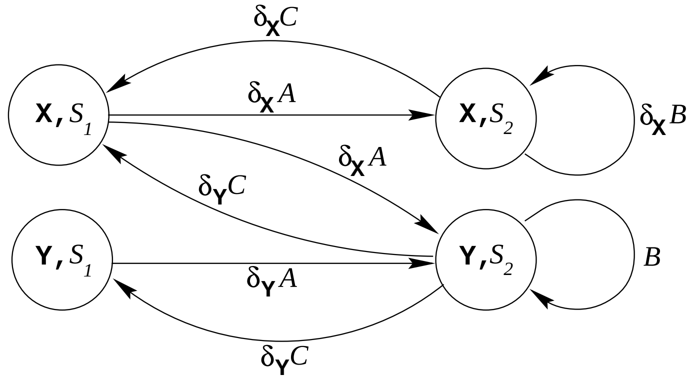

Figure 4: *Multiform time fixpoint equations as flow graphs in $\mathcal{L}$.* For example, this diagram encodes equation $\mathtt{Y}_{S_2} = \mathtt{X}_{S_1}\ \delta_{\mathtt{X}} A \oplus \mathtt{Y}_{S_1}\ \delta_{\mathtt{Y}} A \oplus \mathtt{Y}_{S_2}\ B$.

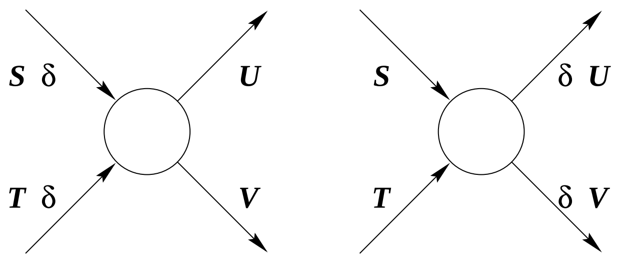

Figure 5: *The basic rule of retiming.* Labels S, T, U, V, δ represent elements of $\mathcal{L}$ attached to the associated branches of the diagrams.

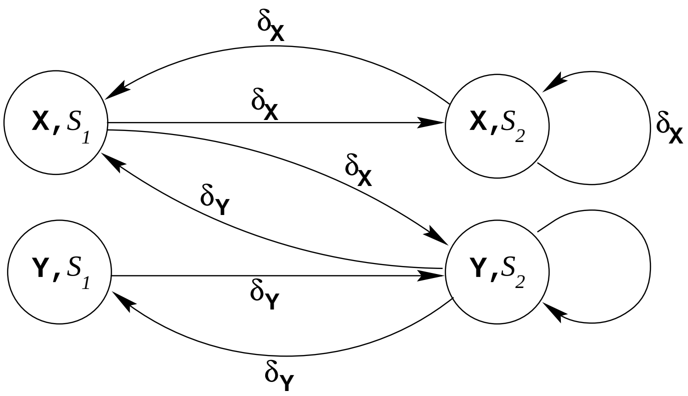

Figure 6: *Multiform time fixpoint equations as flow graphs in $\mathcal{L}$,* where action labels have been deleted.

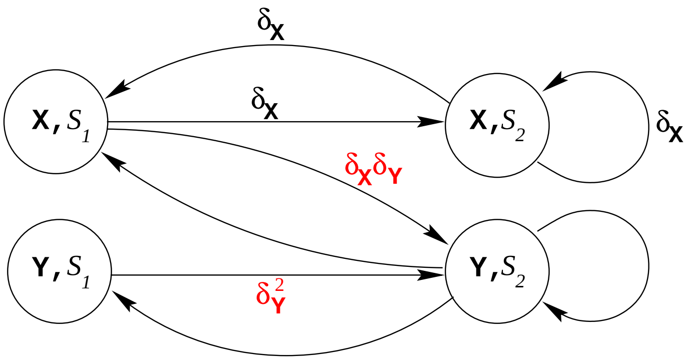

Figure 7: *Applying retiming.* In diagram of figure 6, move δ_{Y} backward.

The proof is obvious, as both diagrams represent the transition matrix

$$\mathcal{T} = \begin{bmatrix} S\delta U & S\delta V \\ T\delta U & T\delta V \end{bmatrix} \tag{51}$$

As a first example of use of retiming, we shall rewrite the graph automaton of figure 2 quite differently. Look at figure 4 : note that action labels can be in fact deleted, since they are redundant with the explicit description of states we have here (A labels moves from S_1 to S_2, etc). This yields Figure 6. Then, consider Figure 7. Finally, the diagram of Figure 7 can be reinterpreted as a rewriting of the graph automaton of Figure 2 as in Figure 8.

4.2.2 Exchanging actions and time

Representation (14,15) 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 (14,15) 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 (14,15) is no longer suitable,

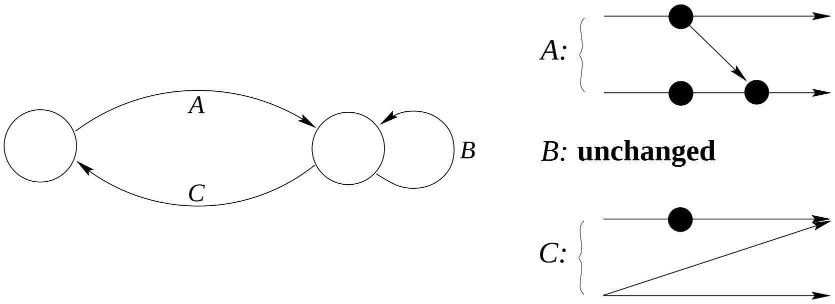

Figure 8: *Rewriting the graph automaton of figure 2.* Finally, reinterpret the diagram of figure 7 as a resynchronised partial order automaton — action labels are attached to the different state transitions; action B is as before, and actions A, C have been modified as indicated.

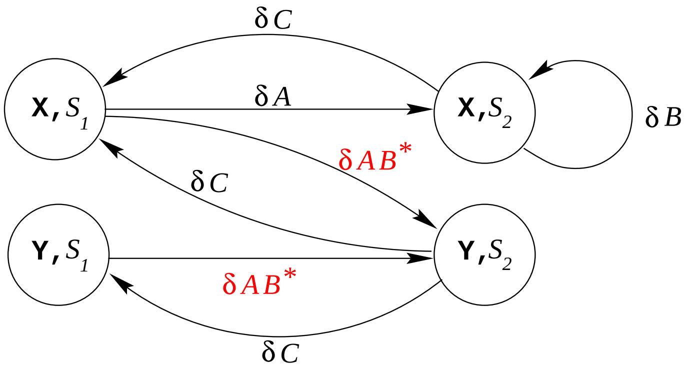

Figure 9: *Exchanging time and actions in the diagram of figure 3.* 1/ Check on diagram of figure 3 which labels have degree zero wrt. δ: there is exactly one, attached to the bottom right circle and action B. 2/ Shift symbol B one step backward along the paths of the directed graph, and concatenate it to the corresponding labels. 3/ Remove arrows with no label. All arrows of the resulting diagram are labelled with power series in $\mathcal{L}$ 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.

and we have, so to say, to "symmetrize" it. This will be achieved by exhibiting another representation, equivalent to (14,15), 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. We show this in Figure 9, in which we again use retiming.

Figure 3 shows the particular case in which labels on transitions are of degree either 1 or 0 with respect to δ. Figure 10 shows how to handle the case where labels on transitions are of finite degree with respect to δ (i.e., no star operation involving symbol δ occurs).

4.2.3 Load balancing

Figure 11 illustrates a situation typical of load balancing problems. The general situation can be modelled as follows. Consider the $\mathcal{L}$ dioid with multiform time: time and actions

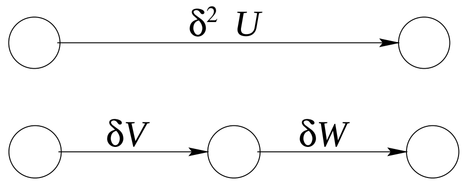

Figure 10: *Handling labels with higher degree w.r.t. δ.* We just need to add new states in the diagram : the diagram on the top shows the original transition. For any decomposition of action word U as $U = VW$, the diagram on the bottom is equivalent to the one on the top, and has degree 1 with respect to δ.

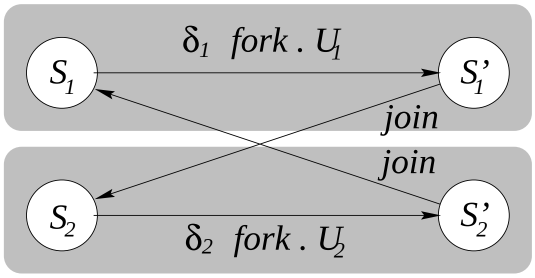

Figure 11: *Load balancing.* The two gray boxes represent two processes performing concurrent computations. Inside each box, a diagram is shown, with two states : S_i, S_i' for $i = 1, 2$. S_i' is a synchronization state : leaving S_i' occurs while the two processors perform a shared "*join*" action, in no time. Then, each process performs a "*fork*", followed by some sequence U_i of actions, in time δ_i, where δ_i denotes some regular expression in several symbols of some alphabet Δ to model multiform time. It is wished to balance load between the two processes, i.e., to enforce that the concurrent computations performed by the two processes take the same time. This leads to imposing the constraint $\delta_1 = \delta_2$, a relation linking the symbols of alphabet Δ.

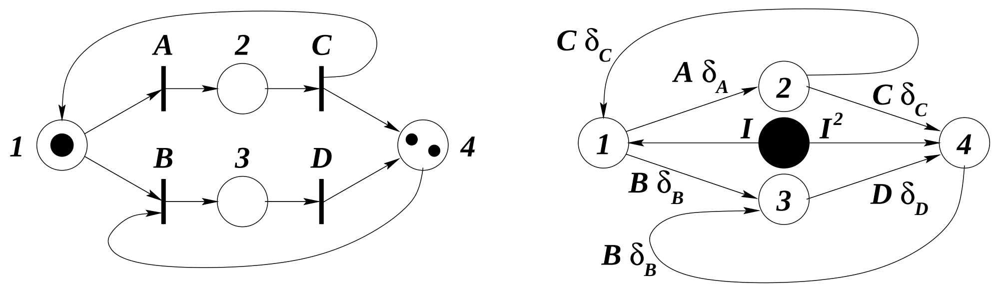

Figure 12: *Modelling timed Petri Nets.* The diagram on the left hand side depicts a timed PN. Transitions A, B, C, D take $\delta_A, \delta_B, \delta_C, \delta_D$ time units for their firing, respectively. Corresponding MaxPlus diagram is shown on the right hand side. In this diagram, states $1, 2, 3, 4$ correspond to the places of the PN with same index. Action symbols are A, B, C, D, and time is uniform, and represented by symbol δ: this defines the $\mathcal{L}$ dioid we use. Expressions $A\delta_A, B\delta_B, C\delta_C, D\delta_D$ are thus regular expressions in $\mathcal{L}$, and states $1, 2, 3, 4$ hold power series $\mathtt{X}_1, \mathtt{X}_2, \mathtt{X}_3, \mathtt{X}_4$ respectively. Finally, the black state holds the formal power series $\mathbf{1} \in \mathcal{L}$, modelling "no action at time zero", and supplementary action symbol I represents an initial token. The fixpoint equations corresponding to this figure are given in formulæ (53).

alphabets respectively are Δ and Σ. We consider two regular expressions

$$\delta_i \in \Delta^\star \ , \ i = 1, 2 \, ,$$

and a constraint

$$\delta_1 \quad = \quad \delta_2 \tag{52}$$

Constraint (52) typically models some constraint of load balancing such as depicted in Figure 11. This means that the appropriate framework to model the system in consideration is not $\mathcal{L}$ any more, but the *quotient* dioid $\mathcal{L}_{\delta_1=\delta_2}$ of elements of $\mathcal{L}$ modulo the weakest congruence generated by constraint $\delta_1 = \delta_2$[1]. In practice, this means that we wish to reparameterize multiform time alphabet Δ with fewer free symbols. This means that we wish to find some smaller alphabet Δ', and regular expressions for each symbol of Δ in terms of symbols of Δ', such that constraint (52) gets satisfied. A particular case is that of Δ' consisting of a singleton $\{\delta_o\}$, modelling, say, machine cycle. How such a reparameterization can actually be realized is deferred to forthcoming work.

5 Back to the examples of section 1

Petri nets

Consider Figure 12. It shows a timed Petri Net and its associated MaxPlus diagram. The

[1] Here we mean the weakest equivalence relation $\sim$ such that 1/ $\delta_1 \sim \delta_2$, and 2/ relation $\sim$ is stable by the dioid operations $\oplus, \otimes$.

fixpoint equations corresponding to this diagram are :

$$\left\{\begin{array}{lcl} \mathtt{X}_1 & = & \mathtt{X}_2 C\delta_C \oplus I \\ \mathtt{X}_2 & = & \mathtt{X}_1 A\delta_A \\ \mathtt{X}_3 & = & (\mathtt{X}_1 \oplus \mathtt{X}_4) B\delta_B \\ \mathtt{X}_4 & = & \mathtt{X}_2 C\delta_C \oplus \mathtt{X}_2 C\delta_C \oplus I^2 \end{array}\right. \tag{53}$$

In these equations, having a $x_{w,\delta^t} = \mathbf{1}$ coefficient in power series $\mathtt{X}_i$ for $i = 1, 2, 3, 4$ indicates that actions word w has been completed within at least t time units.

Possible generalizations and/or variations are : setting durations on places, using multiform time by regarding $\delta_A, \delta_B, \delta_C, \delta_D$ as independent time symbols. Problems for investigation are : bounded safety properties (cf. subsection 4.2.2), load balancing (cf. subsection 4.2.3), and throughput (a question we did not discuss here).

Real-time computing and architectures

This is the example that was extensively discussed in support of our approach.

Task schedulers in real-time kernels

As previously discussed out in points 1 and 2 of general motivations (cf. section 1), our algebraic approach can model task schedulers subject to the restriction that the triggering of tasks occurs due to changes in (both external and internal) states of the system, and there are only finitely many such states. This rules out the accurate modelling of priority-based scheduling mechanisms such as Early Deadline First for example, or alternatively this requires some kind of an abstraction, say, by abstracting priorities via external oracles telling which task gets the priority. Questions of interest are the same as discussed above for Petri Nets.

Retiming and reasoning on pipelining for real-time embedded architectures

Retiming techniques have proven very useful for the design of systolic architectures tightly associated to specific algorithms (e.g., matrix $\times$ vector multiplication, filtering or convolution, QR-factorization, etc.). They allow an optimized design of pipelining and register allocation in the design of special purpose ASICs. Such techniques are restricted to regular algorithms, however, in which no data-dependent control occurs. Our approach actually provides an extension of retiming to such more general situations. Unfortunately, the rule of retiming our theory supports is restricted to the local rule depicted in Figure 5. Due to the noncommutativity of symbols in the monomials of our power series, no global generalization of the rule of Figure 5 (in which arbitrary cuts of the graph can be considered) is available. As a consequence, the complexity of exploring the various configurations for pipelining is expected to increase dramatically, as compared to the classical case.

References

[1] G. Cohen, P. Moller, J-P. Quadrat, M. Viot, "Algebraic Tools for the Performance Evaluation of Discrete Event Systems", *Proc. of the IEEE,* 1989.

[2] F. Baccelli, G. Cohen, G.J. Olsder, J-P. Quadrat, *Synchronisation and Linearity,* Wiley, New York, 1992.

[3] Berstel, Reutenauer, *Les séries rationnelles et leurs langages,* Etudes et recherches en informatique, Masson, Paris, 1984.