Project 22703: SYRF

Deliverable 4/5.3

Integrating synchrony & asynchrony: timing analysis

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 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.

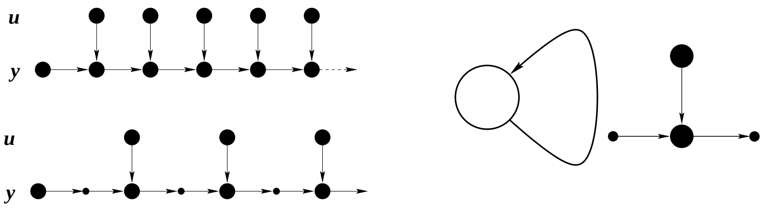

Figure 1: *Computation:* $\forall n \; y_n = y_{n-1} + u_n$, *modeled as a graph automaton.* The first diagram depicts the abstraction of the computation $\forall n : y_n = y_{n-1} + u_n$ in the form of an infinite string of dependence graphs. In this formula, u and y are *flows*, i.e., infinite sequences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

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 a partial order–automaton.

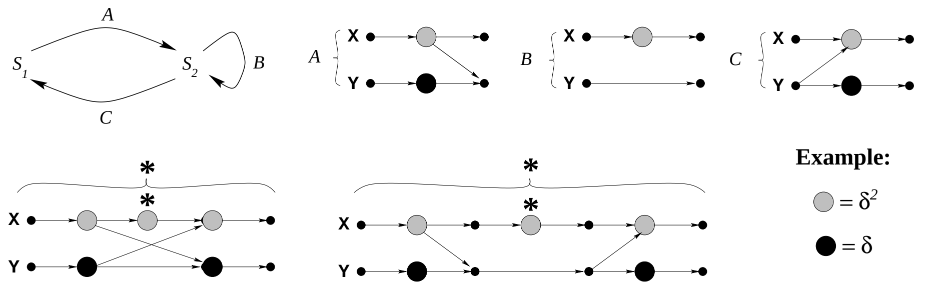

Figure 2: *The main example.* The top left diagram depicts the automaton. Its action labels A, B, C are directed graphs connecting flow occurrences. These directed graphs are shown on the right hand side, they are interpreted as labelled partial orders. Pins are depicted by small bullets, while actual 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 middle 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.

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 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 §2.1 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 (middle). 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.

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 $\delta^t w$.

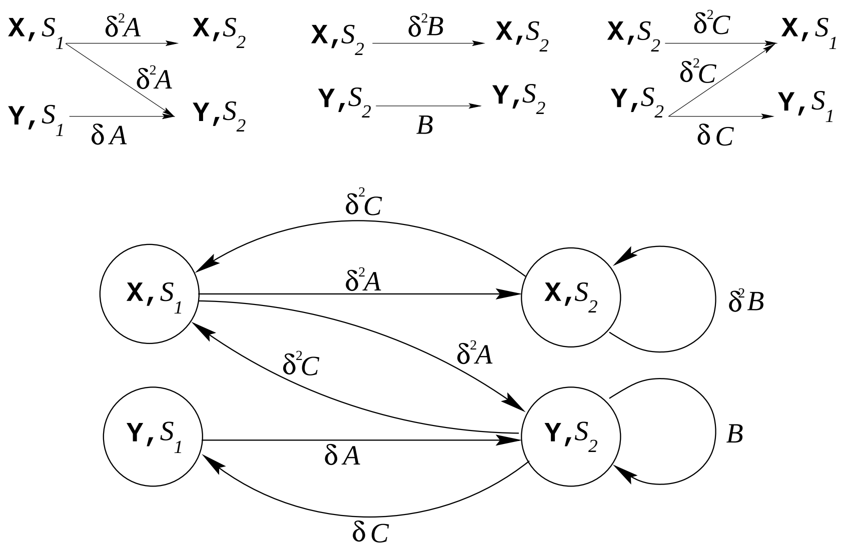

Figure 3: *A timed diagram.* Each pattern A,B and C of the order-automaton of Figure 2 is abstracted as the top diagram. The "union" of these graphs forms the timed diagram on the bottom.

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 completed, one cannot have a date $s, s < t$,

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? This is addressed in the next section.

2 A new formal power series domain

2.1 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 adddition 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:

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

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

2.2 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 [4], 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 $x \in \mathcal{M}(\Sigma, \delta)$,

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

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 [5, 1] of Cohen, Moller, Quadrat and Viot to derive symmetric codings for event graphs (the case considered in [5, 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

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

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

Theorem 1 (the $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ quotient dioid)

1. *The following holds :*

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

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* $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ *obtained by taking the quotient of dioid* $\mathcal{M}(\Sigma,\delta)$ *by the* $\sim$ *equivalence relation.*

2. *The following formula holds :*

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

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

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

COMMENT : Points 2 and 3 of theorem 1 enlight that dioid $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ really implements specification (1), if one interprets monomials as *informations* or *constraints* (see (1)), and addition as *logical conjonction.* Indeed, the rule (4) simply means that the conjonction 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 $\max(s,t)$".

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

$$x_{t,w} = \mathbf{1} \quad \text{iff} \quad \text{word } w \text{ is completed at the } \textit{earliest} \text{ at time } t \text{ for trajectory } x. \tag{5}$$

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

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

2.3 Timed diagrams as fixpoint equations in $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$

This coding is based on the following simple remarks:

(i) Given $x \in \mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$, then $y = \delta x$ is the unique element of $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ such that $y_{t+1,w} = \mathbf{1}$ iff $x_{t,w} = \mathbf{1}$

meaning that (pre– or post–)multiplying an element of $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ by δ amounts to delaying time by one unit.

(ii) Given $x \in \mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$, then $y = x.v$ is the unique element of $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ such that $y_{t,wv} = \mathbf{1}$ iff $x_{t,w} = \mathbf{1}$

meaning that postmultiplying an element of $\mathcal{M}^{\mathrm{ax}}(\Sigma,\delta)$ by v amounts to postfixing the trajectory by word v without incrementing the date.

(iii) We denote by $\mathbf{0}$ the unique element of $\mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$ having all its coefficients equal to $\mathbf{0}$, $\mathbf{0} \in \mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$ encodes the trajectory with no event at all. Also we denote by $\mathbf{1}$ the unique element of $\mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$ 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 $\mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$:

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

where the entries $\mathtt{X}_{S_1}, \mathtt{Y}_{S_1}, \mathtt{X}_{S_2}, \mathtt{Y}_{S_2} \in \mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$ encode the trajectories of flows $\mathtt{X}$ and $\mathtt{Y}$ when hitting states S_1 and S_2 respectively, cf. (5). Keeping points (i), (ii), and (iii) in mind, we get that Ξ is solution of the following fixpoint equation in $\mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$:

$$\Xi \;=\; \Xi \left[\begin{array}{cccc} \mathbf{0} & \mathbf{0} & \delta^2 A & \delta^2 A \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \delta A \\ & & & \\ \delta^2 C & \mathbf{0} & \delta^2 B & \mathbf{0} \\ \delta^2 C & \delta C & \mathbf{0} & B \end{array} \right] \bigoplus \left[\begin{array}{cccc} \mathbf{1} & \mathbf{1} & \mathbf{0} & \mathbf{0} \end{array} \right] \tag{6}$$

Expanding (6), we obtain:

$$\left\{ \begin{array}{lcl} \mathtt{X}_{S_1} & = & \mathtt{X}_{S_2}\, \delta^2 C \oplus \mathtt{Y}_{S_2}\, \delta^2 C \oplus \mathbf{1} \\ \mathtt{X}_{S_2} & = & \mathtt{X}_{S_1}\, \delta^2 A \oplus \mathtt{X}_{S_2}\, \delta^2 B \\ \mathtt{Y}_{S_1} & = & \mathtt{Y}_{S_2}\, \delta C \oplus \mathbf{1} \\ \mathtt{Y}_{S_2} & = & \mathtt{X}_{S_1}\, \delta^2 A \oplus \mathtt{Y}_{S_1}\, \delta A \oplus \mathtt{Y}_{S_2}\, B \end{array} \right.$$

Comparison with automata interpreted as fixpoint equations in Kleene-Schützenberger series. A particular case of our timed diagrams is just the case in which to time symbol is used. In this case,

- timed diagrams reduce to ordinary diagrams for depicting automata
- $\mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$ reduces standard Kleene-Schützenberger series.

In this case, the interpretation of diagrams as fixpoint equations is wellknown.

2.4 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 (6) is too verbose: the system matrix in (6) 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 (6) as

$$\Xi = \left[\begin{array}{cc} \Xi_1 & \Xi_2 \end{array} \right] \text{, where, for } i = 1,2 : \Xi_i = \left[\begin{array}{cc} \mathtt{X}_{S_i} & \mathtt{Y}_{S_i} \end{array} \right]$$

We can rewrite fixpoint equation (6) as follows:

$$\Xi_1 = \Xi_2 \begin{bmatrix} \delta^2 C & \mathbf{0} \\ \delta^2 C & \delta C \end{bmatrix} \oplus \begin{bmatrix} \mathbf{1} & \mathbf{1} \end{bmatrix} \tag{7}$$

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

Substituting (8) in (7) and using the star operator twice yields:

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

$$\mathcal{A} = \left(\begin{bmatrix} \delta^2 A & \delta^2 A \\ \mathbf{0} & \delta A \end{bmatrix} \begin{bmatrix} \delta^2 B & \mathbf{0} \\ \mathbf{0} & B \end{bmatrix}^\star \begin{bmatrix} \delta^2 C & \mathbf{0} \\ \delta^2 C & \delta C \end{bmatrix} \right)^\star =_{\text{def}} (\mathbf{A}_\delta \mathbf{B}_\delta^\star \mathbf{C}_\delta)^\star\ , \tag{9}$$

where $\mathbf{A}_\delta$, etc., denote the corresponding polynomial matrices involved in formula (9).

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.

3 Applications: a few basic exercises

3.1 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 diagram of Figure 3, for which we have the following theorem:

Theorem 2 (retiming) *Both diagrams of Figure 4 represent the same transition matrix.*

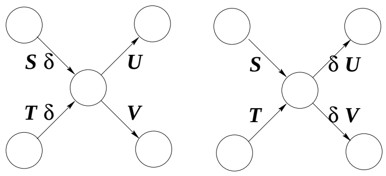

Figure 4: *The basic rule of retiming.* Labels S, T, U, V, δ represent elements of $\mathcal{M}^{\text{ax}}(\Sigma, \delta)$ attached to the associated branches of the diagrams.

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

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

3.2 Exchanging actions and time

Representation (6) 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 (6) 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 (6) is no longer suitable, and we have, so to say, to "symmetrize" it. This will be achieved by exhibiting another representation, equivalent to (6), 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 5 (equivalent to Figure 3), in which we again use retiming.

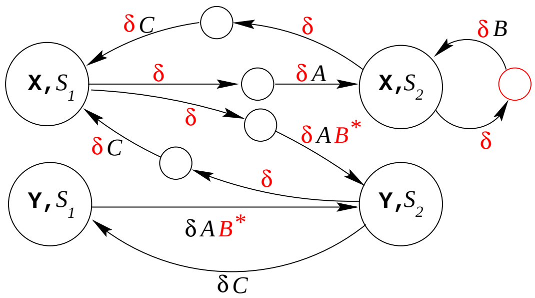

Figure 5: *Exchanging time and actions in the diagram of Figure 3.* 1/ Check on diagram of Figure 3 which labels have degree zero wrt. δ : there is exactly one, attached to the bottom right circle and action B. 2/ Shift symbol B one step backward along the paths of the directed graph, and concatenate it to the corresponding labels. 3/ Remove arrows with no label. All arrows of the resulting diagram are labelled with power series in $\mathcal{M}^{\mathrm{ax}}(\Sigma, \delta)$ 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.

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 $X = \{\delta^t w \mid x_{t,w} = 1\}$. Then, such a subset can be identified either to a function $\Sigma^* \to \mathcal{P}(\delta^*)$, $X'(w) = \{\delta^t \mid x_{t,w} = 1\}$, which to an event w, associates the set of legal time constraints, or to the (inverse) function $X'' : \delta^* \to \mathcal{P}(\Sigma^*)$, $X''(\delta^t) = \{w \mid x_{t,w} = 1\}$, 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* [4] 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.

4 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 [11], 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.

Mode machines mix in a simple way dataflow diagrams and automata [15] [11]. A hierarchical mode machine (hMM) has the form shown in Figure 6. 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 [11]. In our Concur'98 paper, we y establish the encoding of hMM into timed diagrams.

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] Berstel. *Transductions and context-free languages,* Teubner, 1979.

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

[5] 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.

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

[7] S. Gaubert. Resource optimization and (min,+) spectral theory. *IEEE Trans. on Automatic Control*, 40(11), Nov. 1995.

[8] 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.

[9] 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.

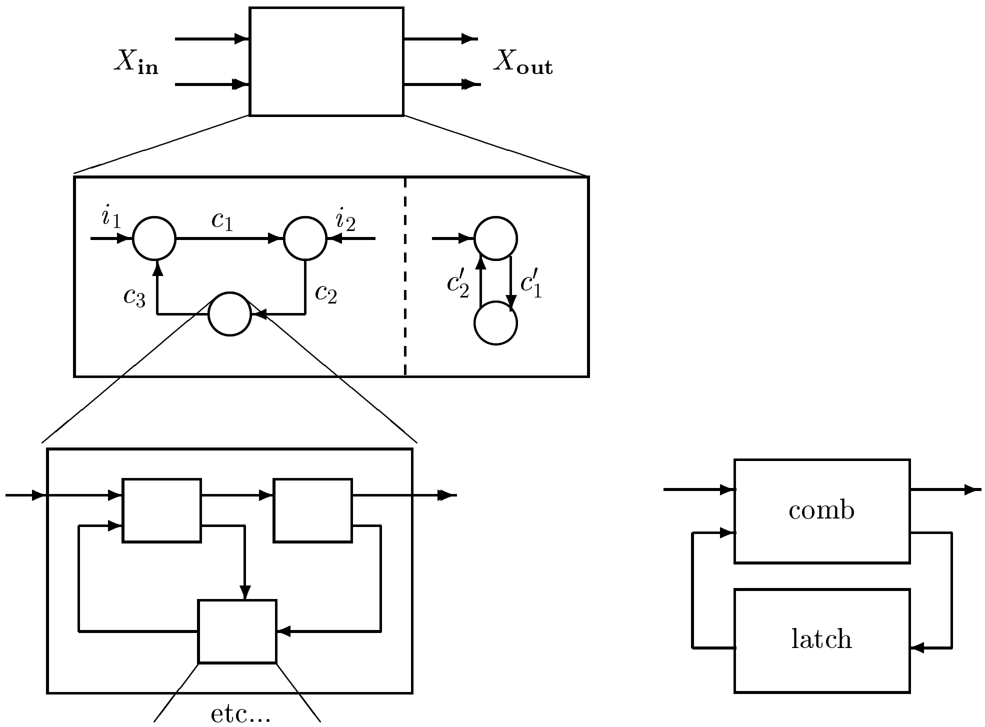

Figure 6: *A view of* hMM. hMM are nested dataflow diagrams and automata. At the lowest level (right hand side) we simply consider simple dataflow diagrams equivalent to synchronous circuits: the circuit has a (circuitfree) combinational part, in feedback loop with a set of latches. Then, the diagram 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 statechart (i.e., a synchronous product of state machines), 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.

[10] S. Gaubert and Max Plus. Methods and applications of (max,+) linear algebra. Rapport de recherche 3088, INRIA, Jan 1997.

[11] 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.

[12] N. Halbwachs. *Synchronous programming of reactive systems,*. Kluwer Academic Pub., 1993.

[13] 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.

[14] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire, "Programming real-time applications with SIGNAL", *Another look at real-time programming*, special section of *Proc. of the IEEE,* vol. 9 n^o 9, September 1991, 1321–1336.

[15] F .Maraninchi, Y. Rémond Mode-Automata: About Modes and States in Reactive Systems. Research Report, Verimag, 1997.