(EVENT|TIME)-VARIANT OPERATORS

B.COTTENCEAU, L.HARDOUIN, J.TRUNK

A C++ toolbox to handle series for event-variant/time-variant (max,+) systems

LARIS, University of Angers

– Revision 2022 –

CONTENTS

1 INTRODUCTION

ETVO is a C++ library to handle the behaviour of a class of timed Discrete Event Systems (DES). This library contains a set of C++ classes to describe and to compute the formal series involved in the description of ordinary Timed Event Graphs (TEGs), Weighted Timed Event Graphs (WTEGs) and Timed Event Graphs with periodic holding times (PTEGs). The WTEGs are the TEGs whose the arcs can be weighted with integer values that indicate how many tokens are consumed/produced by the transition firings. The PTEGs are made of TEGs where the holding times can be described by periodic functions.

The systems concerned by the use of ETVO belong to the theory of (max,+) linear system[1]. More precisely, ETVO is relevant to handle systems that can be qualified as *event-variant* and *time-variant*. For WTEGs, the system response to a solicitation generally varies depending on the input event number. Not all inputs will have the same effect. It's why we say that the input-output behaviour of a WTEG is event-variant. In a symmetric way, for PTEGs, the system response depends on the date of the input and the input-output behavior is said to be time-variant. Finally, ETVO can describe systems that are simultaneously event-variant and time-variant.

The variables of the formal series considered in this work can be assimilate to basic systems called *operators*. In our context, an operator is a mapping able to transform a *signal*, where a signal is a list of events distributed on a time axis. The ETVO library handles a set of elementary operators that allow to describe event and temporal modifications.

This project builds on existing tools. Before this project, a library called MinMaxGD was already existing [8] to handle formal series in the idempotent semiring called $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$. In MinMaxGD, series are well suited to describe time-invariant and event-invariant (min,+)/(max,+) systems. It is the appropriate tool to describe and compute the behaviour of ordinary TEGs. ETVO encompasses the library MinMaxGD as a part of it and extends its set of classes to manage formal series for specific event-variant and time-variant systems.

However, for a user of MinMaxGD, some similarities remain in ETVO. For instance, series are still written in a standard form with an ultimate periodic pattern

$$s = p \oplus q(\gamma^{\nu}\delta^{\tau})^{*}.$$

1 even if afterwards there will be few equations on the (max,+) algebra itself.

But in general, the product is no longer commutative and we can exhibit two equivalent forms

$$\begin{aligned} s &= p \oplus q(\gamma^{\nu}\delta^{\tau})^{*} \\ &= p' \oplus (\gamma^{\nu'}\delta^{\tau'})^{*}q'. \end{aligned}$$

This document gives an introduction to DES and their modelling with ETVO library. The presentation ends with a treatment of some examples. The reader will find complementary presentations in [5], [11], [14] and the PhD thesis of Johannes Trunk [12].

2 DISCRETE EVENT SYSTEMS

2.1 SEQUENCES, COUNTER, DATER

We first recall some features related to the modelling of Discrete Event Systems (DES). A DES is a dynamic system which is driven by the occurrence of punctual phenomena called *events*. An event reflects the moment when the system operates a state evolution. In a manufacturing system modelled as a DES, the events are for instance: the arrival of a part in a stock, the moment when a task is starting or ending, the moment when a resource is being seized or released etc.

More precisely, in a DES, we tend to classify events by type of event. For example, the arrival of a new part in a production cell constitutes a specific type of event ("part arrival"), and the start of a robot cycle another type of event. And for each type of event, there can be several occurrences. Each arrival of a part is a new occurrence of the event type "part arrival". For chaque type of event we use a different label. And we will distinguish the different occurences of a type of event by a number.

Considering a system as a "Discrete Event System" means that its evolution is described by a sequence of occurences of different event types along a time axis. For instance in Fig.1, a type

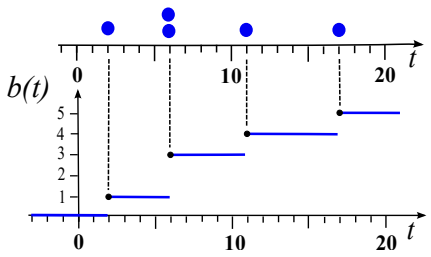

Figure 1: Sequence of events and counter function.

of events labelled b is depicted. Each occurrence of b is depicted by a big dot. This sequence of events can be described by a set of pairs (b_k, t_k) where $k \in \mathbb{N}$, $t_k, b_k \in \mathbb{Z}$, b_k denotes the k-th

occurrence of event b and t_k its occurrence time. For the sequence in Fig.1, the occurrences of b are given by

$$\{b\} = \{(b_0, 2), (b_1, 6), (b_2, 6), (b_3, 11), (b_4, 17), ...\}.$$

The first occurrence of b is at date $t = 2$, then two simultaneous occurrences appear at $t = 6$, etc. A sequence of events can be infinite,

$$\{a\} = \{(a_k, 2k+1)\} = \{(a_0, 1), (a_1, 3), (a_2, 5), ...\}_{k \in \mathbb{N}}$$

or finite

$$\{b\} = \{(b_0, 2), (b_1, 6), (b_2, 6), (b_3, 11), (b_4, 17), (b_5, +\infty)\},$$

which means that the sixth occurrence of b never occurs.

For a reader familiar with the (max,+) approach [1][10], there is two alternative representations of event sequences. The first one is given by a *counter function*

[counter function]

$$\begin{array}{rcl} b(t) & : & \mathbb{Z} \to \mathbb{Z}, \\ & & t \mapsto \text{ the number of events } b \text{ occurred before date } t. \end{array}$$

Fig.1 depicts the counter function $b(t)$ associated to the occurrences of events b. Let us remark that a counter function is naturally monotonic.

The other representation, symmetrical, is called *dater function*. Such a function is defined by

[dater function]

$$\begin{array}{rcl} b(k) & : & \mathbb{Z} \to \mathbb{Z}, \\ & & k \mapsto \text{ date of the k-th occurrence of event b.} \end{array}$$

For the sequence $\{b\}$ depicted in Fig.1, the first values of $b(k)$ are $b(0) = 2, b(1) = 6$, etc.

Sequences of events, counter functions and dater functions, all act as *signals* for DES since they encode the history of the occurrences of a given type of event. Therefore, the generic name "signal" will be sometimes used in place of sequence, counter or dater.

Notation 1 (Sets of signals) *In this document, we will denote by*

$$\Sigma_s \ : \ \textit{the set of event sequences,} \tag{1}$$

$$\Sigma_c \ : \ \textit{the set of counter functions,} \tag{2}$$

$$\Sigma_d \ : \ \textit{the set of dater functions.} \tag{3}$$

2.2 SYNCHRONIZATION

In the DES considered here, the synchronization is clearly the prevailing phenomenon. The synchronization of two sequences of events (denoted by $\oplus$) is a sequence of events. The synchronization can be expressed as follows: $\{a\}, \{b\} \in \Sigma_s$, then $\{c\} = \{a\} \oplus \{b\}$ means that each occurrence c_k is as soon as possible after a_k and b_k.

Formally,

$$\{(c_k, \tau_k)\} = \{(a_k, t_k)\} \oplus \{(b_k, t'_k)\} = \{(c_k, \max(t_k, t'_k)\}$$

For the sequences given in Fig.2, we have $\{a\} = \{(a_0, 1), (a_1, 7), (a_2, 10), (a_3, 10), (a_4, 14)\}$ and $\{b\} = \{(b_0, 2), (b_1, 4), (b_2, 8), (b_3, 13)\}$. Therefore the synchonization of $\{a\}$ and $\{b\}$ is given by

$$\{c\} = \{(c_0, 2), (c_1, 7), (c_2, 10), (c_3, 13)\}.$$

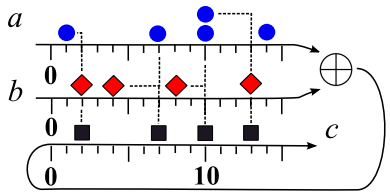

Figure 2: Synchronization of two sequences $\{c\} = \{a\} \oplus \{b\}$

When signals are described as counter or dater functions, the synchronization is expressed as follows. For the counter description, $a, b \in \Sigma_c$,

$$\forall t \in \mathbb{Z},\ (a \oplus b)(t) = \min(a(t), b(t)).$$

For the dater description, $a, b \in \Sigma_d$,

$$\forall k \in \mathbb{Z},\ (a \oplus b)(k) = \max(a(k), b(k)).$$

It is important to remark that the synchronization is an idempotent operation on signals : $\forall a \in \Sigma, a \oplus a = a$.

2.3 OPERATORS

In the ETVO library, the key feature is not on the description of signals. The core of the library lies on the description of *systems* able to transform signals. A system which maps a signal to a signal is called an *operator*. For example in Fig.3, the system S can be considered as an operator able to transform the sequence $\{a\} = \{(a_0,1),(a_1,5),(a_2,9)\}$ into another sequence of events, namely $\{b\} = \{(b_0,3),(b_1,6),(b_2,6),(b_3,9)\}$. System S is an operator s.t. $S\{a\} = \{b\}$.

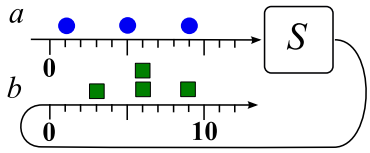

Figure 3: Operator S mapping $\{a\}$ to $\{b\} : \{b\} = S\{a\}$

Remark 1 *It is important to note that changing the type of signal does not change the nature of the operator considered. Whether the signal is a sequence in Σ_s or a counter function in Σ_c, the operator remains the same.*

The ETVO library provides a set of 6 basic operators that can be composed to describe more complex systems. The most simple operators that are described in ETVO are the time-shift and the event-shift operator, *i.e.,* the operators already introduced in [4] to describe the behaviour of the ordinary TEGs.

Time-shift operator δ^τ

The *time-shift* of τ time units is an operator denoted δ^τ. The δ^τ operator transforms a sequence of events into a sequence where each event is time shifted. For $\{a\},\{b\} \in \Sigma_s$, $\delta^\tau\{a\} = \{b\}$ is expressed as

$$\delta^\tau\{(a_k,t_k)\} = \{(b_k,t_k+\tau)\}.$$

In Fig.4, δ^3 operates a shift of 3 time units, say

$$\delta^3\{(a_0,1),(a_1,5),(a_2,5),(a_3,9)\} = \{(b_0,4),(b_1,8),(b_2,8),(b_3,12)\}.$$

When expressed as a mapping on counter functions, the δ^τ operator can be defined by :

$$\forall a \in \Sigma_c, \forall t, (\delta^\tau a)(t) = a(t-\tau).$$

When expressed as a mapping on dater functions, the δ^τ operator can be defined by :

$$\forall a \in \Sigma_d, \forall k, (\delta^\tau a)(k) = a(k) + \tau.$$

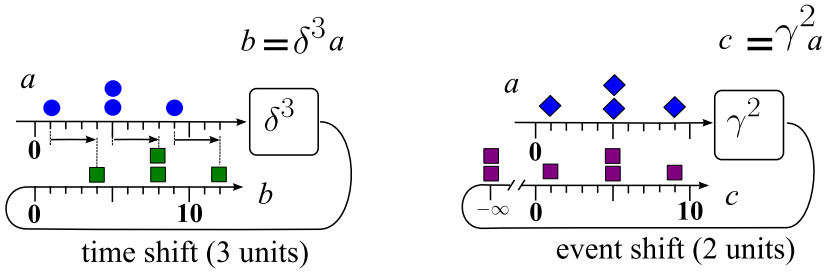

Figure 4: Time-shift/Event-shift : $\{b\} = \delta^3\{a\}$ and $\{c\} = \gamma^2\{a\}$

Event-shift operator γ^ν

The *event-shift* of ν is a basic operator denoted γ^ν. The γ^ν operator produces a sequence of events where the event numbering is shifted. At any given time, the difference between the number of output and input events is fixed and equals to ν, say $\gamma^\nu\{(a_k, t_k)\} = \{(b_{k+\nu}, t_k)\}$. Operator γ^ν maps the event a_k to $b_{k+\nu}$. In order to be consistent, even if the input sequence has no event, ν events are produced by the γ^ν operator at date $-\infty$. More explicitly,

$$\gamma^\nu\{(a_0, +\infty)\} = \{(b_0, -\infty), (b_1, -\infty), ..., (b_{\nu-1}, -\infty), (b_\nu, +\infty)\}.$$

It is why in Fig.4, the first two occurrences of b are at date $-\infty$,

$$\gamma^2\{(a_0,1),(a_1,5),(a_2,5),(a_3,9)\} = \{(b_0,-\infty),(b_1,-\infty),(b_2,1),(b_3,5),(b_4,5),(b_5,9)\}.$$

Operator γ^ν can be expressed as well as a mapping on counter or dater functions. When expressed as a mapping on counter functions:

$$\forall a \in \Sigma_c, \forall k, (\gamma^\nu a)(t) = a(t) + \nu.$$

When expressed as a mapping on dater functions : $a \in \Sigma_d$,

$$\forall a \in \Sigma_d, \forall k, (\gamma^\nu a)(k) = a(k-\nu).$$

2.4 SEMIRING OF OPERATORS

The set of operators introduced in ETVO has an idempotent semiring[1] structure recalled now. This algebraic context gives properties to addition and multiplication operations.

The main feature is that all the basic operators handled in ETVO are *additive*. For a, b being two signals, an operator S is said to be additive if,

$$\forall a, b \in \Sigma,\ S(a \oplus b) = Sa \oplus Sb.$$

Clearly, operators γ^{ν} and δ^{τ} have this property. But hereafter we will introduce four other operators satisfying this property as well.

The set of additive operators can be endowed with an idempotent semiring structure as follows [5].

Notation 2 (Idempotent semiring of additive operators) *The set $\mathcal{O}$ of additive operators, endowed with the sum and the product given below, is an idempotent semiring: $h_1, h_2 \in \mathcal{O}$, $x \in \Sigma$,*

$$h_1 \oplus h_2 \triangleq \forall x, (h_1 \oplus h_2)(x) = h_1 x \oplus h_2 x \tag{4}$$

$$h_1.h_2 \triangleq \forall x, (h_1.h_2)(x) = h_1(h_2 x) \tag{5}$$

In this semiring, the neutral element for the addition is an operator denoted ε and the neutral element for the product is the identity operator denoted e, $\forall x \in \Sigma, e(x) = x$. The semiring $\mathcal{O}$ is not commutative, $h_1.h_2 \neq h_2.h_1$.

Remark 2 *The idempotent semiring structure is the characteristic shared by all the algebraic structures used in the (max,+) system theory. The difference between the (max,+) algebra, that is a semiring, and the algebra of operators denoted $\mathcal{O}$ is the nature of their elements. The (max,+) algebra is an algebra of numbers whereas the semiring $\mathcal{O}$ is an algebra of mappings.*

Notation 3 (Semiring $\mathcal{M}_{\mathbf{in}}^{\mathbf{ax}}[\![\gamma, \delta]\!]$) *The set of operators obtained by composing γ^{ν}, δ^{τ} and ε is a subsemiring of $\mathcal{O}$ denoted $\mathcal{M}_{in}^{ax}[\![\gamma, \delta]\!]$. The identity operator of $\mathcal{M}_{in}^{ax}[\![\gamma, \delta]\!]$ can be expressed by $e = \gamma^0 = \delta^0$ and the null operator by $\varepsilon = \gamma^{+\infty}\delta^{-\infty}$.*

The semiring $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma, \delta]\!]$ is introduced in [4] and also detailed in [1]. It gives an algebraic framework to formally handle TEGs. The next theorem recalls some well-known facts on $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma, \delta]\!]$. The software library called MinMaxGD [8] is dedicated to handle rational computations in $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma, \delta]\!]$ and is totally included in ETVO.

1 This algebraic structure is also called *dioid* in [1]

Theorem 1 *In* $\mathcal{M}_{in}^{ax}[\![\gamma,\delta]\!]$, *we have*

$$\gamma^n\delta^t = \delta^t\gamma^n \tag{6}$$
$$\gamma^n\gamma^{n'} = \gamma^{n+n'} \tag{7}$$
$$\delta^t\delta^{t'} = \delta^{t+t'} \tag{8}$$
$$\gamma^n \oplus \gamma^{n'} = \gamma^{\min(n,n')} \tag{9}$$
$$\delta^t \oplus \delta^{t'} = \delta^{\max(t,t')} \tag{10}$$

Because of (6), $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$ is a commutative idempotent semiring.

Event-variant operators μ_m and β_b

In addition to γ^n and δ^t, ETVO library introduces two event-variant operators denoted μ_m (multiplier) and β_b (batch). The μ_m operator multiplies events. Each input event produces instantaneously m output events. For the example depicted in Fig.5, $\{a\} \in \Sigma_s$, $\{b\} = \mu_2\{a\}$,

$$\mu_2\{(a_0,1),(a_1,4),(a_2,7),...\} = \{(b_0,1),(b_1,1),(b_2,4),(b_3,4),(b_4,7),(b_5,7),...\}$$

Conversely, for the batch operator β_b, b input events are needed to produce one output event. In Fig.5, we have, $\{a\} \in \Sigma_s$, $\{c\} = \beta_3\{a\}$,

$$\beta_3\{(a_0,1),(a_1,4),(a_2,7),...\} = \{(c_0,7),(c_1,15),...\}.$$

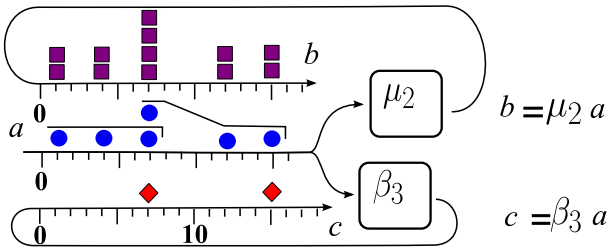

Figure 5: Event Muliplier/Batch : $\{b\} = \mu_2\{a\}$ and $\{c\} = \beta_3\{a\}$

Operators μ_m and β_b can be expressed as mapping on counter functions as follows: $a \in \Sigma_c$

$$\forall a, \forall t \in \mathbb{Z},\ (\mu_m a)(t) = a(t) \times m, \qquad (\beta_b a)(t) = \lfloor a(t)/b \rfloor.$$

Remark 3 *To be exact, only the β_b operator is event-variant. We can see that not all inputs produce the same effect. Only one input event every b inputs leads to an output event.*

Time-variant operators ν_v and ω_w

Finally, the ETVO library introduces two time-variant operators denoted ν_v (time multiplier) and ω_w (time division). The ν_v operator multiplies the dates by v. For the example depicted in Fig.6, $\{a\} \in \Sigma_s$, $\{b\} = \nu_2\{a\}$,

$$\nu_2\{(a_0,-1),(a_1,1),(a_2,1),(a_3,4),...\} = \{(b_0,-2),(b_1,2),(b_2,2),(b_3,8),...\}$$

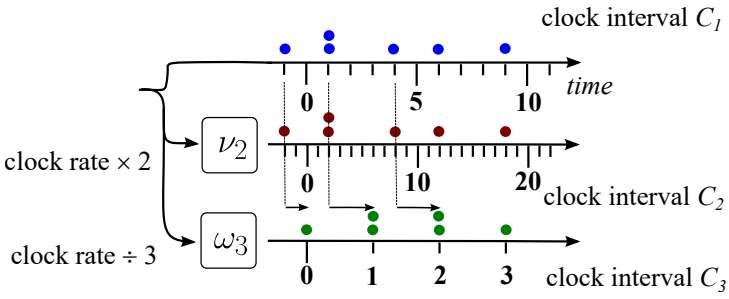

Figure 6: Time multiplication/division : $\{b\} = \Delta_3\{a\}$

Conversely, for the operator ω_w, dates are divided by w. Since the result must be integer, the division of a date t is expressed by $\lceil t/w \rceil$.

In Fig.6, we have, $\{a\} \in \Sigma_s$, $\{c\} = \omega_3\{a\}$,

$$\omega_3\{(a_0,-1),(a_1,1),(a_2,1),(a_3,4),...\} = \{(c_0,0),(c_1,1),(c_2,1),(c_3,2),...\}.$$

Operators ν_v and ω_w can be expressed as mapping on dater functions as follows: $a \in \Sigma_d$

$$\forall a, \forall k \in \mathbb{Z},\ (\nu_v a)(k) = a(k) \times v, \qquad (\omega_w a)(k) = \lceil a(k)/w \rceil.$$

Remark 4 *An effort of interpretation is required to keep these operators practical. As shown in Fig.6, the operators ν_v and ω_w should be interpreted as* time-basis modifiers. *This means that the dates are possibly expressed with different time units (different clock intervals). In Fig.6, three time basis denoted C_1 to C_3 are used to describe the dates of events a, b and c.*

Even if operators ν_v and ω_w are quite hard to interpret, the composed operator obtained by $\Delta_T = \nu_T \omega_T$ can be seen as a specific synchronization on dates which are a multiple of T. For instance, the Δ_3 operator delays all the input events up to the next date in $3\mathbb{Z}$. All the output events are then synchronized on dates in $3\mathbb{Z}$.

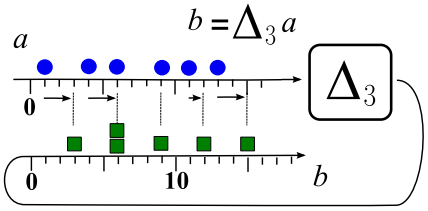

Figure 7: Date synchronization : $\{b\} = \Delta_3\{a\}$

In the example given Fig.7, we have $\{b\} = \Delta_3\{a\}$,

$$\Delta_3\{(a_0,1),(a_1,4),(a_2,6),(a_3,9),...\} = \{(b_0,3),(b_1,6),(b_2,6),(b_3,9),...\}.$$

The operator Δ_T can be expressed as a mapping on dater functions as follows: $a \in \Sigma_d$,

$$\forall a,\ (\Delta_T a)(k) = \lceil a(k)/T \rceil \times T.$$

It is worth noticing that this operator can be also interpreted as a time-variant time-shift operator. For instance in Fig.7, the gap between a_0 and b_0 is 2 time units whereas the gap is 0 time unit between a_2 and b_2. In [14], this feature is denoted $\Delta_3 = \delta^{\langle 0,2,1 \rangle}$ to underline the fact that operator Δ_3 behaves like a time-varying delay.

In summary, the ETVO library introduces the 6 basic operators recalled below. Three of them correspond to a modification in the event domain and are called E-operators (γ, μ, β). The three others implie modification in the time domain and are called T-operators (δ, ν, ω).

		E-operators			T-operators
γ^{ν}	:	event-shift	δ^{τ}	:	time-shift
μ_m	:	event-multiplier	ν_v	:	date-multiplier
β_b	:	event-batch	ω_w	:	date-divider

By considering only some subsets of these operators, we obtain four different idempotent semirings, which are subsemirings of $\mathcal{O}$, that are usefull for DES modelling

$$\begin{array}{rcl}
\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!] & : & \text{semiring of sums and products in } \{\varepsilon,\gamma^{\nu},\delta^{\tau}\} \\
\mathcal{E}[\![\delta]\!] & : & \text{semiring of weight-balanced sums and products in } \{\varepsilon,\gamma^{\nu},\delta^{\tau},\mu_m,\beta_b\} \\
\mathcal{T}[\![\gamma]\!] & : & \text{semiring of weight-balanced sums and products in } \{\varepsilon,\gamma^{\nu},\delta^{\tau},\nu_v,\omega_w\} \\
\mathcal{ET} & : & \text{semiring of weight-balanced sums and products in} \{\varepsilon,\gamma^{\nu},\delta^{\tau},\mu_m,\beta_b,\nu_v,\omega_w\}
\end{array}$$

The semiring $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$ is detailed in [4], [1] and the MinMaxGD toolbox to handle periodic series is presented in [8].

The semiring $\mathcal{E}[\![\delta]\!]$ is presented in [5],[6],[11],[13].

The semiring $\mathcal{T}[\![\gamma]\!]$ is introduced in [14].

All these algebraic structures are also recalled and detailed in the PhD thesis of Johannes Trunk.

Proposition 1 *In semiring $\mathcal{O}$, operators γ^n, δ^t, μ_m, β_b, Δ_T satisfy:*

$$\begin{array}{rclrclrcl}
\gamma^1\delta^1 & = & \delta^1\gamma^1 & \gamma^n\gamma^{n'} & = & \gamma^{n+n'} & \delta^t\delta^{t'} & = & \delta^{t+t'} \quad (f1) \\
\gamma^n \oplus \gamma^{n'} & = & \gamma^{\min(n,n')} & \delta^t \oplus \delta^{t'} & = & \delta^{\max(t,t')} & & & \quad (f2) \\
\mu_m\delta^1 & = & \delta^1\mu_m & \beta_b\delta^1 & = & \delta^1\beta_b & \beta_m\mu_m & = & e \quad (f3) \\
\Delta_T\gamma^1 & = & \gamma^1\Delta_T & \Delta_T\delta^T & = & \delta^T\Delta_T & & & \quad (f4) \\
\mu_m\mu_{m'} & = & \mu_{m\times m'} & \beta_b\beta_{b'} & = & \beta_{b\times b'} & & & \quad (f5) \\
\mu_m\gamma^1 & = & \gamma^m\mu_m & \gamma^1\beta_b & = & \beta_b\gamma^b & & & \quad (f6)
\end{array}$$

3 MODELLING WITH ETVO

ETVO has been developed in order to assist ourself in the computation of the behaviour of DES. This section is devoted to the practical way calculations should be organized with ETVO.

3.1 TIMED EVENT GRAPHS

The formal series handled in ETVO are well suited to describe the behaviour of some subclasses of timed Petri nets. First, we give here a very short description of the subclass of Timed Event Graphs. This graphical model is presented in more detail in other references such as [1, chap.2],[3], [10, chap.7],[9].

A Timed Event Graph (TEG) is a timed Petri net - with P the set of places, T the set of transitions and $A \subset (P \times T) \cup (T \times P)$ the set of edges - such that *each place has exactly one upstream and one downstream transition*.

Each place $p_k \in P$ can have a positive holding time value $\tau \in \mathbb{N}$ and an initial marking denoted $M_0(p_k) \in \mathbb{N}$. The holding time is the minimal time a token must stay in the place before being able to cross the downstream transition.

We denote by $p^\bullet$ (resp. ${}^\bullet p$) the downstream (resp. upstream) transition of place p, and we denote by $t^\bullet$ (resp. ${}^\bullet t$) the set of downstream (resp. upstream) places of transition t. When a TEG runs according to the *earliest firing rule*[1], a transition t_j fires as soon as each place $p_l \in {}^\bullet t_j$ contains at least n ($n > 0$) available tokens. Then n tokens are removed from each place p_l, and n tokens are added to each place $p_k \in t_j^\bullet$.

Timed Event Graphs provide a graphical representation of different dynamical phenomena. For a TEG, signals (sequence, counters, daters) are attached to transitions. A transition is linked to an event type and each of its firings is an occurrence of that event type.

In Fig.8 we see the main phenomena arising in TEGs. A transition with two upstream places describes the synchronization of events. A place with a holding time leads to a time-shift between the upstream and the downstream transitions. Finally, the initial marking acts as a shift in the event numbering.

1 also called As Soon As Possible (ASAP) functioning

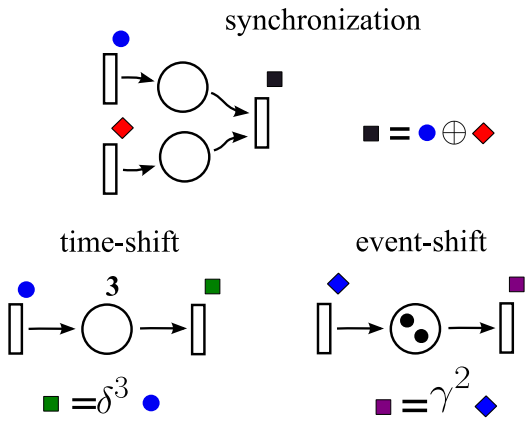

Figure 8: Elementary Timed Event Graphs

Therefore, the structure of a TEG can be translated into a block-diagram where only γ^ν and δ^τ operators are involved, as well as the synchronization of signals (denoted $\oplus$ in the block-diagram). For TEGs, the semiring considered is $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma, \delta]\!]$ and the computations can be made by the MinMaxGD library included into ETVO.

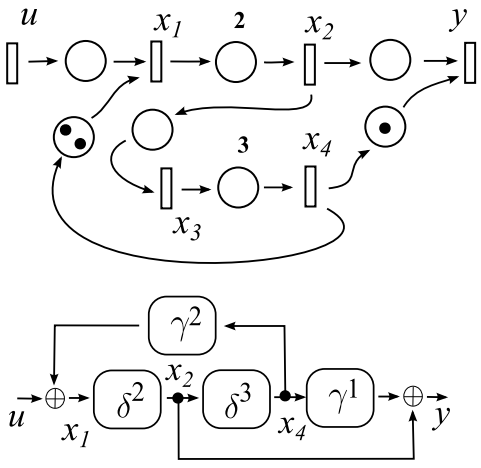

Figure 9: TEG decomposition into basic operators

In Fig.9, a TEG is depicted. Its decomposition in γ^{ν} and δ^{τ} operators is given as a block-diagram. The input corresponds to the signal u and the output is given by the signal y. All the signals are related to each other by the next relations,

$$\begin{aligned} x_1 &= u \oplus \gamma^2 x_4, \\ x_2 &= \delta^2 x_1, \\ x_3 &= x_2, \\ x_4 &= \delta^3 x_3, \\ y &= x_2 \oplus \gamma^1 x_4. \end{aligned}$$

We can describe this system in a matrix form where $x = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}'$ is a vector of signals, and A, B and C are matrices whose entries are operators

$$\begin{cases} x &= Ax \oplus Bu \\ y &= Cx \end{cases}$$

with

$$A = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \gamma^2 \\ \delta^2 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \delta^3 & \varepsilon \end{pmatrix}, B = \begin{pmatrix} e \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, C = \begin{pmatrix} \varepsilon & e & \varepsilon & \gamma^1 \end{pmatrix}.$$

Theorem 2 *In a complete idempotent semiring, equation $x = ax \oplus b$ admits $x = a^*b = (e \oplus a \oplus a^2 \oplus a^3 \oplus \ldots)b$ as least solution.*

Since $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma, \delta]\!]$ is a complete idempotent semiring and according to Th.2, the input-output behaviour of this system can be described by

$$y = CA^*Bu = (CB \oplus CAB \oplus CA^2B \ldots)u.$$

For this example and with the help of ETVO/MinMaxGD, the computation gives

$$y = (\delta^2 \oplus \gamma^1\delta^5)(\gamma^2\delta^5)^*u.$$

This rational expression is an operator that describes how the input signal is transformed into the ouput signal by the system. By analogy with the classical system theory, we say that this operator is the transfer function of the TEG.

For this short example, the C++ script using ETVO is given below.

Listing 3.1: TEG transfer with ETVO/C++

```
#include "etvo.h"
using namespace std;   // namespace for cout object
using namespace etvo;  // namespace for ETVO classes
int main(){
    // series is a type for MinMaxGD series
        matrix<series> A(4,4), B(4, 1), C(1, 4);
        // all entries of A,B,C are initially set to epsilon
        B(0,0)= gd(0,0); //g0.d0=e
        C(0,1)= gd(0,0);
        C(0,3)= gd(1,0);        //g1.d0=g1
        A(0,3)= gd(2,0);
        A(1,0)= gd(0,2);        //g0.d2=d2
        A(2,1)= gd(0,0);
        A(3,2)= gd(0,3);

        matrix<series>  H = C * A.star() * B;
        cout << H(0,0) << endl;
        // output : (g0.d2+g1.d5).[g2.d5]*
}
```

ETVO interpreter/calculator

ETVO also comes with a rudimentary interpreter/calculator. It is a program in which series can be entered in the form of rational expressions. The intermediate results are always reduced to an ultimate periodic form. No programming skills are required to use this rudimentary tool. Another advantage of this tool is that the results provided also satisfy the grammatical rules of the interpreter. The results can therefore be saved in text files and used again later in the calculator [2].

With this tool, the script is as follows:

Listing 3.2: ETVO calculator

```
SA=eps(4,4)
SB=eps(4,1)
SC=eps(1,4)
SA(0,3)=g2
SA(1,0)=d2
SA(2,1)=g0
```

2 Let us note that a version of this interpreter is also available via a web page.

```
SA(3,2)=d3
SB(0,0)=g0
SC(0,1)=g0
SC(0,3)=g1
SH=SC.[SA]*.SB
[output] SH(0,0)=(g0.d2+g1.d5).[g2.d5]*
```

3.2 WEIGHTED TIMED EVENT GRAPHS

A Weighted Timed Event Graph is a Timed Event Graph the edges of which have an integer weight (see [7]). For $p_k \in P$ a place, the edge $t_i \rightarrow p_k$ (resp. $p_k \rightarrow t_o$) is valued by a strictly positive integer denoted $w_i(p_k)$ (resp. $w_o(p_k)$) (the weights of the edges). In order to avoid confusion with holding times, weights of edges are denoted between brackets, e.g. $\langle 2 \rangle$. Moreover, $t_i \rightarrow p_k \rightarrow t_o$ defines an elementary path denoted π_k the gain of which is given by $\Gamma(\pi_k) \triangleq w_i(p_k)/w_o(p_k) \in \mathbb{Q}$.

The weights describe how many tokens are consumed/produced by each transition firing. When one considers the *earliest firing rule*, a transition t_j fires as soon as each input place p_l of t_j contains at least $w_o(p_l)$ available token(s). Then $w_o(p_l)$ token(s) is(are) removed from each input place p_l of t_j, and $w_i(p_k)$ token(s) is(are) added to each output place p_k of t_j.

Fig.10 considers weights as basic operators. A weight on the output edge of a transition describes a multiplication of events. The μ_m operator can model this phenomenon. Conversely, a weight on the input edge of a transition describes a batch operation which is modeled by a β_b operator.

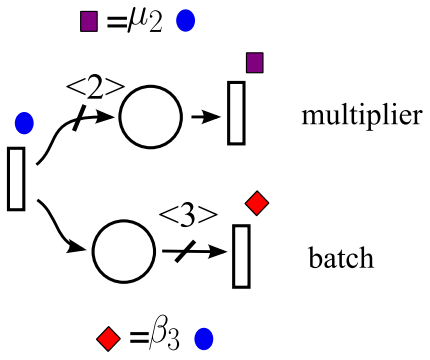

Figure 10: Elementary Weighted Timed Event Graphs

In Fig.11, a Weighted TEG is depicted. Its decomposition in basic operators is given as a block-diagram. All the relation between signals are expressed below :

$$\begin{aligned} x_1 &= \beta_2 u \oplus \beta_3 \gamma^5 \mu_2 x_4 \\ x_2 &= \delta^2 \mu_3 x_1 \\ x_3 &= x_2 \\ x_4 &= \beta_2 \delta^5 x_3 \\ y &= \beta_2 x_2 \oplus \gamma^1 x_4 \end{aligned}$$

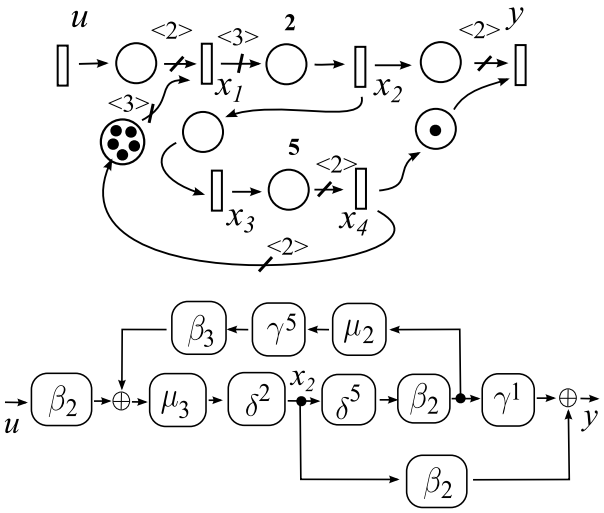

Figure 11: WTEG decomposition into basic operators

We can describe this system in a matrix form where $x = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}'$,

$$\begin{cases} x &= Ax \oplus Bu \\ y &= Cx \end{cases}$$

with

$$A = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \beta_3\gamma^5\mu_2 \\ \delta^2\mu_3 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \beta_2\delta^5 & \varepsilon \end{pmatrix}, B = \begin{pmatrix} \beta_2 \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, C = \begin{pmatrix} \varepsilon & \beta_2 & \varepsilon & \gamma^1 \end{pmatrix}$$

The input-output behaviour of this system can be described by $y = CA^*Bu$. For this example, the computation with the ETVO library gives

$$y = \left((\mu_3\beta_4\gamma^2 \oplus \gamma^1\mu_3\beta_4)\delta^2 \oplus (\gamma^1\mu_3\beta_4\gamma^2 \oplus \gamma^2\mu_3\beta_4)\delta^7\right)(\gamma^2\delta^7)^*u$$

For this short example, the C++ script is given below.

Listing 3.3: WTEG transfer with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main(){
        matrix<seriesEd> A(4,4), B(4,1), C(1,4);
        B(0, 0) = eb(2); //b2
        C(0, 1) = eb(2);
        C(0, 3) = eg(1); //g1
        A(0, 3) = eb(3)*eg(5)*em(2); //b3.g5.m2
        A(1, 0) = em(3)*ed(2); //m3.d2
        A(2, 1) = eg(0); //g0=e
        A(3, 2) = eb(2)*ed(5); //b2.d5
        matrix<seriesEd> H=C*A.star()*B;
        H(0, 0).toRight();
        cout << H(0,0) << endl;
//output=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
}
```

In ETVO/C++, the basic operators are coded by some global functions: $eg(n)$ for γ^n, $em(w)$ for μ_w, $eb(v)$ for β_v and $ed(t)$ for δ^t.

The same computation with the interpreter/calculator is given below.

Listing 3.4: WTEG transfer with ETVO/calculator

```
EA=eps(4,4)
EB=eps(4,1)
EC=eps(1,4)
EA(0,3)=b3.g5.m2
EA(1,0)=m3.d2
EA(2,1)=g0
EA(3,2)=b2.d5
EB(0,0)=b2
EC(0,1)=b2
EC(0,3)=g1
```

```
EH=EC.[EA]*.EB
eH=right(EH(0,0))
[output]eH=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
```

3.3 TIMED EVENT GRAPHS WITH PERIODIC HOLDING TIMES

ETVO can handle the basic T-operators denoted δ^t, ν_v, ω_w and $\Delta_T = \nu_T\omega_T$. As said before, the Δ_T operator describes a synchronization on dates in $T\mathbb{Z}$. With this operator, we can model delays changing as time changes. For instance, the Δ_3 operator can be considered as a delay δ^τ the value of which is time-variant. The Δ_3 operator adds no delay for input events occuring at dates in $3\mathbb{Z}$. For instance, $\Delta_3\{(a_0,0),(a_1,6)\} = \{(b_0,0),(b_1,6)\}$. For an input event occuring at dates in $3\mathbb{Z}+1$, the delay is 2 time units : $\Delta_3\{(a_0,1),(a_1,1),(a_2,4)\} = \{(b_0,3),(b_1,3),(b_2,6)\}$. And the delay is only 1 time unit for input events occuring at dates in $3\mathbb{Z}+2$. We summarize this with the notation $\Delta_3 = \delta^{\langle 0,2,1\rangle}$. The time-variant delays thus obtained are necessarily with a periodic sequence of values.

By considering DES with time-variant delays, we can model Timed Event Graphs with periodic holding times. This class of models is called Periodic Time-variant Event Graphs (PTEGs) in the thesis of J.Trunk. Each periodic time-variant delay can be obtained as a finite composition of fixed delays δ^τ and Δ_T operators.

In Fig.12, a PTEG is depicted as well as its decomposition in time-variant operators. As said before, $\delta^{\langle 0,2,1\rangle} = \Delta_3$. For the others time-variant holding times, we have the equivalence:

$$\begin{aligned} \delta^{\langle 1,0\rangle} &= \delta^1\Delta_2\delta^{-1} \\ \delta^{\langle 2,3,2\rangle} &= \delta^2\Delta_3\delta^{-2} \oplus \delta^1\Delta_3 \end{aligned}$$

We can describe this system in a matrix form where $x = \begin{pmatrix} x_1 & x_2 & x_3 & x_4 \end{pmatrix}'$,

$$\begin{cases} x &= Ax \oplus Bu \\ y &= Cx \end{cases}$$

with

$$A = \begin{pmatrix} \varepsilon & \varepsilon & \varepsilon & \gamma^2 \\ \delta^1 & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & e & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \delta^{\langle 2,3,2\rangle} & \varepsilon \end{pmatrix}, B = \begin{pmatrix} \Delta_3 \\ \varepsilon \\ \varepsilon \\ \varepsilon \end{pmatrix}, C = \begin{pmatrix} \varepsilon & \delta^{\langle 1,0\rangle} & \varepsilon & \gamma^1 \end{pmatrix}.$$

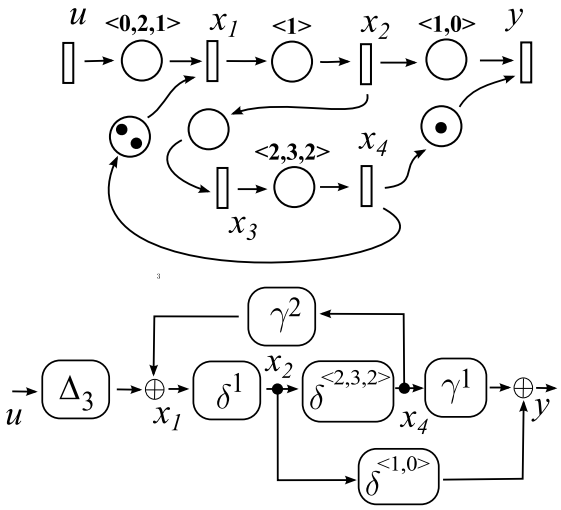

Figure 12: PTEG decomposition into operators

The behaviour of this system can be described by a rational expression $y = Gu = CA^*Bu$. For this example, the computation with the ETVO library gives

$$y = \delta^{\langle 1,4,3,2,3,2\rangle}\gamma^0 \oplus (\delta^{\langle 4,6,5\rangle}\gamma^1 \oplus \delta^{\langle 5,8,7,6,7,6\rangle}\gamma^2)(\delta^3\gamma^2)^* u.$$

For this short example, the C++ script is given below.

Listing 3.5: PTEG transfer with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;
int main(){
        matrix<seriesTg> A(4, 4), B(4, 1), C(1, 4);
        B(0, 0) = tD(3); //D3=d<0,2,1>
        C(0, 1) = td({1,0}); //d<1,0>
        C(0, 3) = tg(1); //g1
        A(0, 3) = tg(2);
        A(1, 0) = td(1); //d1
        A(2, 1) = tg(0);
        A(3, 2) = td({2,3,2});
        matrix<seriesTg>  G=C*A.star()*B;
        G(0,0).toRight();
```

```
        cout << G(0,0).toStringAsDeltaVar() << endl;
//output=((d<1,4,3,2,3,2>.g0))+(d<4,6,5>.g1+d<5,8,7,6,7,6>.g2).[d3.g2]*
}
```

Listing 3.6: PTEG tranfser with the calculator

```
TA=eps(4,4)
TB=eps(4,1)
TC=eps(1,4)
TA(0,3)=g2
TA(1,0)=d1
TA(2,1)=g0
TA(3,2)=d<2,3,2>
TB(0,0)=v3.w3
TC(0,1)=d<1,0>
TC(0,3)=g1
TG=TC.[TA]*.TB
tG=right(TG(0,0))
asDeltaVar(tG)
[output] ((d<1,4,3,2,3,2>.g0))+(d<4,6,5>.g1+d<5,8,7,6,7,6>.g2).[d3.g2]*
```

3.4 CYCLOWEIGHTED TIMED EVENT GRAPHS

In [6], a model of Weighted TEGs with variable (cyclic) weights is introduced. This model is directly inspired by the Cyclostatic Synchronous Dataflow model [2]. One must then introduce E-variant operators capable of describing variable-weight multiplication and division operators. These operators actually have an equivalent expression with fixed weight operators. Said differently, this does not allow to describe additional dynamic phenomena. On the other hand, the notion of cyclic routing is then expressed quite simply.

In Figure 13, the behavior of the two operators $\mu_{<1,3,2>}$ and $\beta_{<1,2>}$ is illustrated on a sequence of events. The ETVO tool integrates the management of these operators and translates them into a combination of μ and β operators with fixed weight.

Listing 3.7: Cycloweighted operators in ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;
```

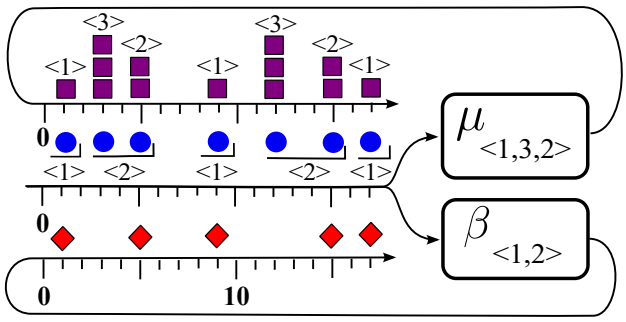

Figure 13: Cyclo-weighted operators

```
int main()
{
        seriesEd mvar,bvar;
        mvar=em({1,3,2}); // m<1,3,2>
        bvar=eb({1,2});   // b<1,2>
        cout << "mvar=" << mvar << endl;
        cout << "bvar=" << bvar << endl;
        //output mvar=((m6.b3.g2+g1.m6.b3.g1+g4.m6.b3).d0)
        //output bvar=((m2.b3.g2+g1.m2.b3).d0)
}
```

The same calculation performed this time within the interpreter gives:

Listing 3.8: Cycloweighted operators in the interpreter/calculator

```
->eMvar=m<1,3,2>
eMvar=((m6.b3.g2+g1.m6.b3.g1+g4.m6.b3).d0)
->eBvar=b<1,2>
eBvar=((m2.b3.g2+g1.m2.b3).d0)
```

By attaching variable weights to the transitions of a TEG, we obtain a Cycloweighted TEG for which we obtain an operator-based model. An example from [6] is given in Figure 14.

Listing 3.9: Computation with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
```

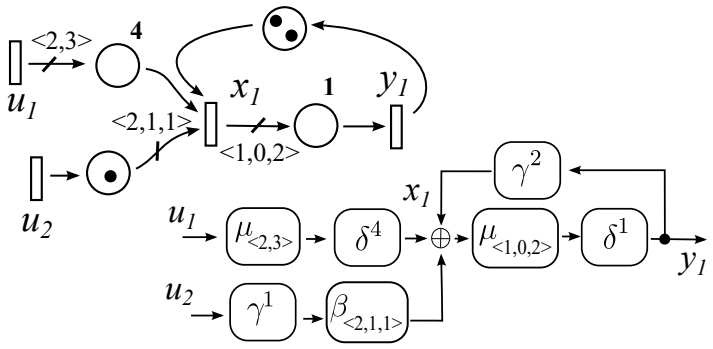

Figure 14: Cycloweighted TEG and its block diagram description

```
matrix<seriesEd> A(2,2), B(2,2),C(1,2),H(1,2);
A(0,1)=eg(2);
A(1,0)=ed(1)*em({1,0,2});
B(0,0)=ed(4)*em({2,3});
B(0,1)=eb({2,1,1})*eg(1);
C(0,1)=eg(0);
H=C*A.star()*B;
H(0,0).toRight();
H(0,1).toRight();
cout << H << endl;
/*output

[0,0]=((m15.b6.g5+g1.m15.b6.g4+g4.m15.b6.g3+g7.m15.b6.g2+g10.m15.
   b6.g1+g12.m15.b6).d5+(g1.m15.b6.g5+g3.m15.b6.g4+g6.m15.b6.g3+g9
   .m15.b6.g2+g12.m15.b6.g1+g13.m15.b6).d6+(g3.m15.b6.g5+g4.m15.b6
   .g4+g7.m15.b6.g3+g10.m15.b6.g2+g13.m15.b6.g1+g15.m15.b6).d7+(g4
   .m15.b6.g5+g6.m15.b6.g4+g9.m15.b6.g3+g12.m15.b6.g2+g15.m15.b6.
   g1+g16.m15.b6).d8+(g6.m15.b6.g5+g7.m15.b6.g4+g10.m15.b6.g3+g13.
   m15.b6.g2+g16.m15.b6.g1+g18.m15.b6).d9+(g7.m15.b6.g5+g9.m15.b6.
   g4+g12.m15.b6.g3+g15.m15.b6.g2+g18.m15.b6.g1+g19.m15.b6).d10+(
   g9.m15.b6.g5+g10.m15.b6.g4+g13.m15.b6.g3+g16.m15.b6.g2+g19.m15.
   b6.g1+g21.m15.b6).d11+(g10.m15.b6.g5+g12.m15.b6.g4+g15.m15.b6.
   g3+g18.m15.b6.g2+g21.m15.b6.g1+g22.m15.b6).d12+(g12.m15.b6.g5+
   g13.m15.b6.g4+g16.m15.b6.g3+g19.m15.b6.g2+g22.m15.b6.g1+g24.m15
   .b6).d13+(g13.m15.b6.g5+g15.m15.b6.g4+g18.m15.b6.g3+g21.m15.b6.
   g2+g24.m15.b6.g1+g25.m15.b6).d14).[g6.d10]*
[0,1]=((m3.b4.g3+g1.m3.b4.g1).d1).[g2.d1]*
*/
```

```
}
```

The same example, this time calculated in the interpreter. In order to be more concise, the result is expressed here using the cycloweighted operators.

Listing 3.10: Computation with the calculator

```
EA=eps(2,2)
EB=eps(2,2)
EC=eps(1,2)
EA(0,1)=g2
EA(1,0)=d1.m<1,0,2>
EB(0,0)=d4.m<2,3>
EB(0,1)=b<2,1,1>.g1
EC(0,1)=g0
EH=EC.[EA]*.EB
eH1=right(EH(0,0))
eH2=right(EH(0,1))
asMuVar(eH1)
asMuVar(eH2)
[output]eH1=(g0.m<1,3,3,3,2,3>.d5+g1.m<2,3,3,3,1,3>.d6+g3.m<1,3,3,3,2,3>.
    d7+g4.m<2,3,3,3,1,3>.d8+g6.m<1,3,3,3,2,3>.d9+g7.m<2,3,3,3,1,3>.d10+g9.m
    <1,3,3,3,2,3>.d11+g10.m<2,3,3,3,1,3>.d12+g12.m<1,3,3,3,2,3>.d13+g13.m
    <2,3,3,3,1,3>.d14).[g6.d10]*
[output] eH2=(g0.m<1,0,2,0>.d1).[g2.d1]*
```

Thanks to the cycloweighted operators, it is quite natural to express cyclic routings. We take another example from [6] where demuliplexing and multiplexing operators allow to describe a routing in parallel branches of a production workshop. In this example, the inputs are processed by the H_1 and H_2 systems according to the following cyclicity: $[H_1, H_2, H_1, H_2, H_2]$.

This example is treated only with the help of the interpreter. Each branch of the system is a linear (max,+) system described by a rational expression. The two branches are synchronized using operators describing demultiplexing and multiplexing.

Listing 3.11: Cycloweighted operators with the calculator

```
eH1=d1.d3.[g2.d3]*
eH2=d4.[g4.d4]*.d1.[g2.d1]*
eG=b<1,0,1,0,0>.eH1.m<1,0,1,0,0>+b<0,1,0,1,1>.eH2.m<0,1,0,1,1>
asMuVar(eG)
[output]
eH1=(g0.d4).[g2.d3]*
eH2=(g0.d5+g2.d6).[g4.d4]*
```

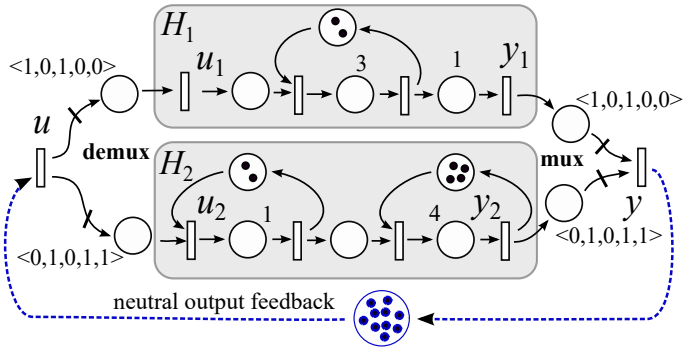

Figure 15: Demux/Mux operations

```
eG=[g20.d12]*.(g0.d4+(g1.m5.b5.g3+g3.m5.b5.g1+g4.m5.b5).d5+(g4.m5.b5.g3+g6
    .m5.b5.g1+g8.m5.b5).d6+(g5.m5.b5.g4+g7.m5.b5.g2+g9.m5.b5.g1).d7+(g8.m5.
    b5.g3+g9.m5.b5.g1+g11.m5.b5).d9+g10.d10+(g14.m5.b5.g3+g16.m5.b5.g1+g18.
    m5.b5).d13+(g18.m5.b5.g3+g19.m5.b5.g1+g21.m5.b5).d14)
[expressed with cycloweighted operators]
[g20.d12]*.(g0.m<1>.d4+g1.m<0,2,0,1,2>.d5+g4.m<0,2,0,2,1>.d6+g5.m
    <2,0,2,1,0>.d7+g8.m<0,1,0,2,2>.d9+g10.m<1>.d10+g14.m<0,2,0,2,1>.d13+g18
    .m<0,1,0,2,2>.d14)
```

3.5 MULTI-CLOCK TIMED EVENT GRAPH

T-operators are also used to describe time basis changes. In the example Figure 16, the two subsystems do not use the same time unit value. Between the two systems, the operator $\nu_2\omega_3$ indicates that the time unit is changed at the junction of the systems.

ETVO can be used to express the transfer function of such a system. In this case, there is a temporal gain between the input and the output. The time basis to describe the input events is then not the same as the one to describe the output events.

Listing 3.12: T-variant operators with the calculator

```
tH1=d<0,2,1>.d1.d1.[g1.d1]*
tH2=d1.d4.[g2.d4]*
tG=tH2.v2.w3.tH1
[output] tG=(d7.v2.w3.d-1.g0+d7.v2.w3.g1).[d6.g2]*
```

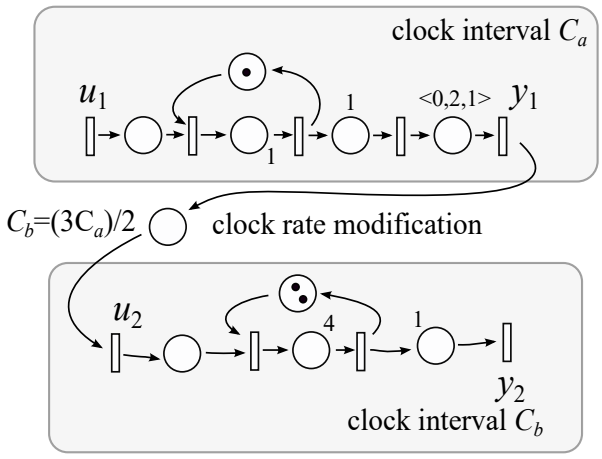

Figure 16: TEG with two different time units (clock rate change)

4 SYSTEM RESPONSE AND CONTROL

4.1 SYSTEM RESPONSE

An input-output model obtained with ETVO describes the dynamics of a system. It can also be used to know the response of a system to which an input signal is submitted since $y = Hu$ where u is an input signal and y the associate response.

4.1.1 Response of a TEG

Let's go back to the example discussed in the section 3.1. It is a TEG whose the transfer function is given by $H = (\delta^2 \oplus \gamma^1\delta^5)(\gamma^2\delta^5)^*$.

We encode an input signal in a series of $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$. For instance, for a signal given by

$$\{u\} = \{(u_0,4),(u_1,8),(u_2,8),(u_3,8),(u_4,8),(u_5,11),(u_6,+\infty)\},$$

its description in $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$ is

$$u = \gamma^0\delta^4 \oplus \gamma^1\delta^8 \oplus \gamma^2\delta^8 \oplus \gamma^3\delta^8 \oplus \gamma^4\delta^8 \oplus \gamma^5\delta^{11} \oplus \gamma^6\delta^{+\infty}.$$

In $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$, it is equivalent to

$$u = \gamma^0\delta^4 \oplus \gamma^1\delta^8 \oplus \gamma^5\delta^{11} \oplus \gamma^6\delta^{+\infty}.$$

This signal describes a finite sequence of input events. Nevertheless, ETVO does not handle infinite values to describe the absence of the u_6 event. As an alternative, we can describe this by a date of arbitrarily large value, compared to the other dates. The following C++ script provides the result of this calculation. The interpretation of the result gives:

$$y = Hu = \gamma^0\delta^6 \oplus \gamma^1\delta^{10} \oplus \gamma^2\delta^{13} \oplus \gamma^3\delta^{15} \oplus \gamma^4\delta^{18} \oplus \gamma^5\delta^{20} \oplus \gamma^6\delta^{+\infty}$$

Listing 4.1: Computation with ETVO/C++

```
#include "etvo.h"
```

```
using namespace std;
using namespace etvo;

int main()
{
 series H;
 H=series(gd(0,2)+gd(1,5))*series(gd(2,5)).star();
 series U=gd(0,4)+gd(1,8)+gd(5,11)+gd(6,10000);
 cout << H << endl;
 series Y=H*U;
 cout << Y << endl;
 /* output H=(g0.d2+g1.d5).[g2.d5]*
   output Y=g0.d6+g1.d10+g2.d13+g3.d15+g4.d18+g5.d20
                +(g6.d10002+g7.d10005).[g2.d5]*
   = g0.d6+g1.d10+g2.d13+g3.d15+g4.d18
                +g5.d20+g6.d10002+g7.d10005 + ....
 */
}
```

The same calculation can be executed with the interpreter/calculator.

Listing 4.2: Computation with the interpreter

```
sH=(g0.d2+g1.d5).[g2.d5]*
sU=g0.d4+g1.d8+g5.d11+g6.d10000
sY=sH.sU
[output]
 sH=(g0.d2+g1.d5).[g2.d5]*
 sU=g0.d4+g1.d8+g5.d11+(g6.d10000)
 sY=g0.d6+g1.d10+g2.d13+g3.d15+g4.d18+g5.d20+(g6.d10002+g7.d10005).[g2.d5
    ]*
```

4.1.2 Response of a Weighted TEG

For WTEGs, the result is less immediate (see the thesis of J.Trunk for further explanations). Let's consider the example discussed in the section 3.2. It is a Weighted TEG whose the transfer function is given by $H = \left((\mu_3\beta_4\gamma^2 \oplus \gamma^1\mu_3\beta_4)\delta^2 \oplus (\gamma^1\mu_3\beta_4\gamma^2 \oplus \gamma^2\mu_3\beta_4)\delta^7\right)(\gamma^2\delta^7)^*$.

Once again, one can describe an input signal by a series in $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$, for instance

$$u = \gamma^0\delta^4 \oplus \gamma^1\delta^8 \oplus \gamma^5\delta^{11} \oplus \gamma^6\delta^{13} \oplus \gamma^7\delta^{15} \oplus \gamma^8\delta^{16} \oplus \gamma^9\delta^{+\infty}.$$

To obtain the associate response, we use the zero-slice mapping introduced in [12, section 3.2]. Formally, the corresponding output $y \in \mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$ is obtained by

$$y = \Psi_{3|4}(Hu) \qquad (\text{where } Hu \in \mathcal{E}[\![\delta]\!]).$$

The computation with ETVO gives

$$y = \gamma^0\delta^{10} \oplus \gamma^1\delta^{17} \oplus \gamma^2\delta^{22} \oplus \gamma^3\delta^{24} \oplus \gamma^4\delta^{31} \oplus \gamma^5\delta^{36} \oplus \gamma^6\delta^{+\infty}.$$

Let us note that in this case, the gain is $(3/4)$, in average 4 input events lead to 3 output events.

Listing 4.3: Computation with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
        seriesEd H;
        H=Ed(0,3,4,2,2)+Ed(1,3,4,0,2)+Ed(1,3,4,2,7)+Ed(2,3,4,0,7);
        H=H*seriesEd(Ed(0,1,1,2,7)).star();
        series U=gd(0,4)+gd(1,8)+gd(5,11)+gd(6,13)+gd(7,15)
                +gd(8,16)+gd(9,100000);
        seriesEd Um=seriesEd::toSeriesEd(U);
        seriesEd Y=H*Um;
        series y = Y.toSeries();

        cout << "H=" <<H << endl;
        cout << "y=" <<y << endl;
        /* output
        H=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
        y=g0.d10+g1.d17+g2.d22+g3.d24+g4.d31+g5.d36+(g6.d100002+ ...
        */

}
```

The same calculation can be executed with the interpreter/calculator.

Listing 4.4: Computation with the interpreter

```
eH=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
eU=g0.d4+g1.d8+g5.d11+g6.d13+g7.d15+g8.d16+g9.d100000
sY=EdToMM(eH.eU)
[output]
```

```
        eH=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
        eU=(g0.d4+g1.d8+g5.d11+g6.d13+g7.d15+g8.d16+g9.d100000)
        sY=g0.d10+g1.d17+g2.d22+g3.d24+g4.d31+g5.d36+g6.d100002+ ...
```

4.1.3 Response of a TEG with periodic holding times

Let's consider the example discussed in the section 3.3. It is a TEG with periodic holding times whose the transfer function is given by

$$H = \delta^{\langle 1,4,3,2,3,2\rangle}\gamma^0 \oplus (\delta^{\langle 4,6,5\rangle}\gamma^1 \oplus \delta^{\langle 5,8,7,6,7,6\rangle}\gamma^2)(\delta^3\gamma^2)^* u.$$

We describe first the input signal by a series in $\mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$. For instance, we choose here

$$u = \gamma^0\delta^4 \oplus \gamma^1\delta^8 \oplus \gamma^5\delta^{11} \oplus \gamma^6\delta^{13} \oplus \gamma^7\delta^{15} \oplus \gamma^8\delta^{16} \oplus \gamma^9\delta^{+\infty}.$$

Then, to obtain the associate response, we use the zero-slice mapping introduced in [12, section 4.4]. Formally, the corresponding output $y \in \mathcal{M}_{\text{in}}^{\text{ax}}[\![\gamma,\delta]\!]$ is obtained by

$$y = \Psi_{6|6}(Hu) \qquad (\text{where } Hu \in \mathcal{T}[\![\gamma]\!]).$$

The computation with ETVO gives

$$y = \gamma^0\delta^7 \oplus \gamma^1\delta^{11} \oplus \gamma^2\delta^{13} \oplus \gamma^3\delta^{15} \oplus \gamma^4\delta^{16} \oplus \gamma^5\delta^{17} \oplus \gamma^6\delta^{19} \oplus \gamma^7\delta^{21} \oplus \gamma^8\delta^{22} \oplus \gamma^9\delta^{+\infty}.$$

Listing 4.5: Computation with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
        seriesTg H;
        H=(td({4,6,5})*tg(1)+td({5,8,7,6,7,6})*tg(2))*(seriesTg(Tg(3,2)).
            star());
        H=H+td({1,4,3,2,3,2});
        series u=gd(0,4)+gd(1,8)+gd(5,11)+gd(6,13)+gd(7,15)
                +gd(8,16)+gd(9,100000);
        seriesTg Um=seriesTg::toSeriesTg(u);
        seriesTg Y=H*Um;
```

```
        series y = Y.toSeries();
        H.toRight();
        cout << "H=" << H << endl;
        cout << "H=" << H.toStringAsDeltaVar() << endl;
        cout << "u=" << u << endl;
        cout << "y=" << y << endl;

        /* output
        H= (((d-1.D6+d1.D6.d-3).g0))+(d4.D3.g1+(d3.D6+d5.D6.d-3).g2).[d3.
            g2]*
        H=((d<1,4,3,2,3,2>.g0))+(d<4,6,5>.g1+d<5,8,7,6,7,6>.g2).[d3.g2]*
        u=g0.d4+g1.d8+g5.d11+g6.d13+g7.d15+g8.d16+(g9.d100000)
        y=g0.d7+g1.d11+g2.d13+g3.d15+g4.d16+g5.d17+g6.d19+g7.d21+g8.d22+g9
            .d100003+...
        */
}
```

The same calculation can be executed with the interpreter/calculator.

Listing 4.6: Computation with the interpreter

```
tH=((d<1,4,3,2,3,2>.g0))+(d<4,6,5>.g1+d<5,8,7,6,7,6>.g2).[d3.g2]*
tH=right(tH)
tu=g0.d4+g1.d8+g5.d11+g6.d13+g7.d15+g8.d16+(g9.d100000)
sy=TgToMM(tH.tu)
[output]
        tH=(((d-1.v6.w6+d1.v6.w6.d-3).g0))+(d4.v3.w3.g1+(d3.v6.w6+d5.v6.w6
            .d-3).g2).[d3.g2]*
        tu=(d4.g0+d8.g1+d11.g5+d13.g6+d15.g7+d16.g8+d100000.g9)
        sy=g0.d7+g1.d11+g2.d13+g3.d15+g4.d16+g5.d17+g6.d19+g7.d21+g8.d22+
            g9.d100003+...
```

4.2 OPTIMAL CONTROL

The first control problem solved for the (max,+) systems consisted in searching for the input signal u allowing the output of the system, $y = Hu$, to be as close as possible to a reference signal denoted z. Thanks to the residuation theory, we know that for a given signal z, the following equation has an optimal solution,

$$Hu \preceq z \iff u \preceq H \backslash z,$$

where $H\backslash z = \bigoplus\{x \text{ s.t. } Hx \preceq z\}$. The optimality means here that the input events are as late as possible while, for the associated output, the events y are before the reference output z. We say that this input is optimal according to the "just-in-time" criterion.

Both operations $a\backslash b = \bigoplus\{x \text{ s.t. } ax \preceq b\}$ and $b/a = \bigoplus\{x \text{ s.t. } xa \preceq b\}$ are implemented in ETVO.

4.2.1 Optimal control for TEGs

Let's consider again the example discussed in the section 3.1 where $H = (\delta^2 \oplus \gamma^1\delta^5)(\gamma^2\delta^5)^*$.

We need to choose an output reference, for instance

$$z = \gamma^0\delta^{10} \oplus \gamma^3\delta^{13} \oplus \gamma^5\delta^{19} \oplus \gamma^6\delta^{+\infty}.$$

Again, to compensate for the lack of infinite value management in ETVO, we must code this trajectory as an ultimate periodic series (with the same slope as H) as follows

$$z = \gamma^0\delta^{10} \oplus \gamma^3\delta^{13} \oplus \gamma^5\delta^{19} \oplus \left(\gamma^6\delta^{largeValue}(\gamma^2\delta^5)^*\right).$$

The optimal output is then

$$y_{opt} = Hu_{opt} = \gamma^0\delta^3 \oplus \gamma^1\delta^6 \oplus \gamma^2\delta^8 \oplus \gamma^3\delta^{11} \oplus \gamma^4\delta^{13} \oplus \gamma^5\delta^{19} \oplus \gamma^6\delta^{+\infty}.$$

Events 4 and 5 perfectly match the reference and the other outputs are before the reference.

Listing 4.7: Computation with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
        series H;
        H=series(gd(0,2)+gd(1,5))*series(gd(2,5)).star();
        series z(poly::Epsilon(),gd(6,10000),gd(2,5));
        z=z+gd(0,10)+gd(3,13)+gd(5,19);
        series uopt=z.frac(H);
        series yopt=H*uopt;
        cout << "uopt=" << uopt << endl;
        cout << "yopt=" << yopt << endl;
        //uopt=g0.d1+g1.d3+g2.d6+g3.d8+g4.d11+g5.d17+(g6.d9995+...
        //yopt=g0.d3+g1.d6+g2.d8+g3.d11+g4.d13+g5.d19+(g6.d9997+...
}
```

The same calculation can be executed with the interpreter/calculator.

Listing 4.8: Computation with the interpreter

```
sH=(g0.d2+g1.d5).[g2.d5]*
sZ=g0.d10+g3.d13+g5.d19+g6.d10000.[g2.d5]*
sUopt=frac(sZ,sH)
sYopt=sH.sUopt
[output]
        sH=(g0.d2+g1.d5).[g2.d5]*
        sZ=g0.d10+g3.d13+g5.d19+(g6.d10000).[g2.d5]*
        sUopt=g0.d1+g1.d3+g2.d6+g3.d8+g4.d11+g5.d17+g6.d9995+...
        sYopt=g0.d3+g1.d6+g2.d8+g3.d11+g4.d13+g5.d19+g6.d9997+...
```

4.3 CONTROLLER SYNTHESIS

We focus here on two types of control. The precompressor control, and the output feedback control.

For a given system H, there is a greatest neutral precompensator and a greatest neutral output feedback. Based on the theory of residuation, we obtain $\hat{P} = H\backslash H$ and $\hat{F} = H\backslash H/H$. $\hat{P}$ is the greatest solution to $Hx = H$ and $\hat{F}$ is the greatest solution to $H(xH)^* = H$.

To obtain a realizable system, we must project the result in the set of causal series.

4.3.1 Controller synthesis for a TEG

Let's consider again the example discussed in the section 3.1 where $H = (\delta^2 \oplus \gamma^1\delta^5)(\gamma^2\delta^5)^*$. Using the interpreter, we obtain the expression of the greatest precompensator. We also check that the system with the control keeps the same transfer as the system without control.

Listing 4.9: Computation with the interpreter

```
sH=(g0.d2+g1.d5).[g2.d5]*
sPopt=frac(sH,sH)
sFopt=pr+(frac(frac(sH,sH),sH))
sC1=sH.sPopt
sC2=sH.[sFopt.sH]*
[output]
        sH=(g0.d2+g1.d5).[g2.d5]*
        sPopt=(g0.d0+g1.d2).[g2.d5]*
```

```
        sFopt=(g1.d0+g2.d2).[g2.d5]*
        sC1=(g0.d2+g1.d5).[g2.d5]*
        sC2=(g0.d2+g1.d5).[g2.d5]*
```

4.3.2 Controller synthesis for a Weighted TEG

Let's consider the example discussed in the section 3.2 where the transfer function is given by $H = \left((\mu_3\beta_4\gamma^2 \oplus \gamma^1\mu_3\beta_4)\delta^2 \oplus (\gamma^1\mu_3\beta_4\gamma^2 \oplus \gamma^2\mu_3\beta_4)\delta^7\right)(\gamma^2\delta^7)^*$.

Listing 4.10: Computation with the interpreter

```
eH=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
ePopt=lfrac(eH,eH)
eFopt=pr+(rfrac(lfrac(eH,eH),eH))
eCL1=eH.ePopt
eCL2=right(eH.[eFopt.eH]*)
[output]
        eH=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
        ePopt=[g2.d7]*.(m2.b2.d0)
        eFopt=[g2.d7]*.((g2.m4.b3.g2+g4.m4.b3).d0+(g2.m4.b3.g2+g4.m4.b3.g1
            ).d5)
        eCL1=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
        eCL2=((m3.b4.g2+g1.m3.b4).d2+(g1.m3.b4.g2+g2.m3.b4).d7).[g2.d7]*
```

5 EXAMPLES

In this part, we propose to treat some examples using the ETVO tool.

5.1 EXAMPLE 1

For the example in Figure 17, each machine M_1/M_2 can be described by a transfer function.

$$x_2 = \delta^6(\gamma^5\delta^6)^*\delta^4 u,$$

$$x_4 = \delta^5(\gamma^4\delta^5)^*\delta^1 x_2.$$

Globally, the input-output transfer is $y = \delta^4\left(\delta^5(\gamma^4\delta^5)^*\delta^1\right)\left(\delta^6(\gamma^5\delta^6)^*\delta^4\right)u$. With ETVO, we obtain the transfer function $y = Hu$ with:

$$\begin{aligned} H \;=\; & \delta^{20} \oplus \gamma^4\delta^{25} \oplus \gamma^5\delta^{26} \oplus \gamma^8\delta^{30} \oplus \gamma^9\delta^{31} \oplus \gamma^{10}\delta^{32} \\ & \oplus \left(\gamma^{12}\delta^{35} \oplus \gamma^{13}\delta^{36} \oplus \gamma^{14}\delta^{37} \oplus \gamma^{15}\delta^{38}\right)(\gamma^4\delta^5)^* \end{aligned}$$

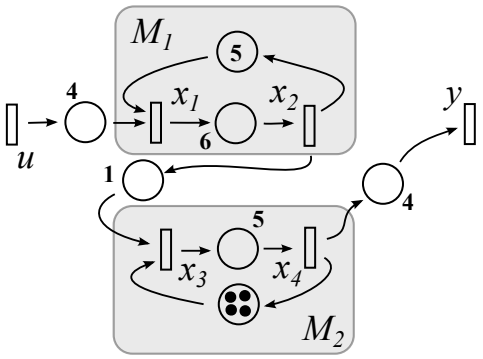

Figure 17: TEG (example 1)

Listing 5.1: Example 1 with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
        series H;
        series M1(poly::Epsilon(),gd(0,6),gd(5,6));
        series M2(poly::Epsilon(),gd(0,5),gd(4,5));
        H=series(gd(0,4))*M2*gd(0,1)*M1*gd(0,4);
        cout << "H=" <<  H << endl;
        // H=g0.d20+g4.d25+g5.d26+g8.d30+g9.d31+g10.d32+(g12.d35+g13.d36+
           g14.d37+g15.d38).[g4.d5]*
}
```

Listing 5.2: Example 1 with ETVO/interpreter

```
sM1=d6.[g5.d6]*
sM2=d5.[g4.d5]*
sH=d4.sM2.d1.sM1.d4
```

5.2 EXAMPLE 2

This is the same example, this time we are interested in the synthesis of an output feedback. The greatest neutral output feedback is given by

$$\hat{F} = H \backslash H / H.$$

It is the greatest feedback such that the closed-loop behavior be the same as the open-loop one, say $H = H(\hat{F}H)^*$.

For the system taken in the example 1, the ETVO tool computes the transfer function of the greatest feedback.

$$F = \left(\gamma^{16} \oplus \gamma^{17}\delta^{1} \oplus \gamma^{18}\delta^{2} \oplus \boldsymbol{\gamma^{19}\delta^{3}}\right)(\gamma^{4}\delta^{5})^*.$$

Note that, for example, a feedback with 19 tokens and a delay of 3 time units is also neutral.

Listing 5.3: Example 2 with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;
int main()
{
        series H, F;
        series M1(poly::Epsilon(),gd(0,6),gd(5,6));
        series M2(poly::Epsilon(),gd(0,5),gd(4,5));
        H=series(gd(0,4))*M2*gd(0,1)*M1*gd(0,4);

        // F=H\H/H
        F=H.frac(H).frac(H);
        // causal projection
        F=F.prcaus();
        cout << "F=" << F << endl;
        series Fs=series(gd(19,3));
        cout << "Fs=" << Fs << endl;
        cout << "H.(Fs.H)*=" << H*(Fs*H).star() << endl;
        /* output
        F=(g16.d0+g17.d1+g18.d2+g19.d3).[g4.d5]*
        Fs=(g19.d3)
        H.(Fs.H)*=g0.d20+g4.d25+g5.d26+g8.d30+g9.d31+g10.d32+(g12.d35+g13.
            d36+g14.d37+g15.d38).[g4.d5]*
        */
}
```

Listing 5.4: Example 2 with ETVO/interpreter

```
sM1=d6.[g5.d6]*
sM2=d5.[g4.d5]*
sH=d4.sM2.d1.sM1.d4
sF=pr+(frac(frac(sH,sH),sH))
sCL=sH.[g19.d3.sH]*
```

5.3 EXAMPLE 3

For the example in Figure 19, each machine M_1 / M_2 can be described by a transfer function,

$$H_1 = \delta^2(\gamma^1\delta^2)^*,$$

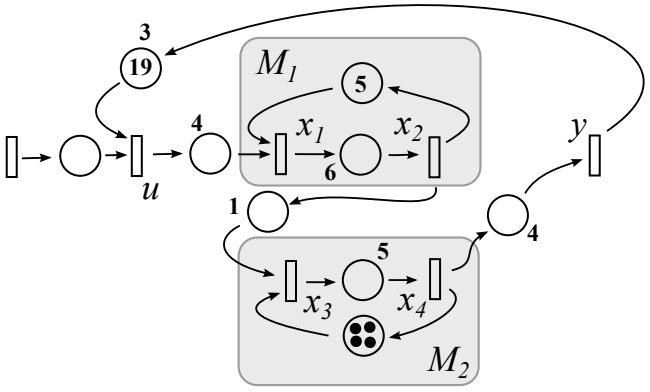

Figure 18: TEG (example 2) with an output feedback

$$H_2 = \delta^1(\gamma^2\delta^1)^*.$$

Globally, it is a weighted TEG, some E-variant operators are involved. The input-output transfer is given by $y = Gu$,

$$y = \delta^1\left(\delta^2\mu_3 H_1\beta_2\delta^2 \oplus \beta_2\gamma^1\delta^1 H_2\delta^2\mu_3\delta^1\right)\delta^2 u.$$

Using the ETVO tool, we obtain the following ultimate periodic expression:

$$\begin{aligned} G \;=\; & \mu_3\beta_2\delta^9 \oplus (\gamma^2\mu_3\beta_2\gamma^1 \oplus \gamma^3\mu_3\beta_2)\delta^{10} \oplus \gamma^3\mu_3\beta_2\delta^{11} \\ & \oplus(\gamma^4\mu_3\beta_2\gamma^1 \oplus \gamma^6\mu_3\beta_2)\delta^{12} \oplus (\gamma^5\mu_3\beta_2\gamma^1 \oplus \gamma^6\mu_3\beta_2)\delta^{13} \\ & \oplus(\gamma^1\delta^1)^*(\gamma^6\mu_3\beta_2\gamma^1 \oplus \gamma^8\mu_3\beta_2)\delta^{14} \end{aligned}$$

Listing 5.5: Example 3 with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
 seriesEd G;
 series H1(poly::Epsilon(),gd(0,2),gd(1,2));
 series H2(poly::Epsilon(),gd(0,1),gd(2,1));
```

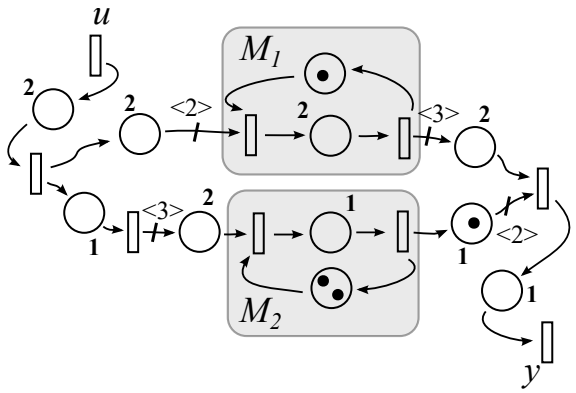

Figure 19: Weighted TEG (example 3)

```
 G=ed(2)*em(3)*seriesEd::toSeriesEd(H1)*eb(2)*ed(2);
 G=G+eb(2)*eg(1)*ed(1)*seriesEd::toSeriesEd(H2)*ed(2)*em(3)*ed(1);
 G=ed(1)*G*ed(2);
 cout << "G=" << G << endl;
/* output
        G=((m3.b2.d9+(g2.m3.b2.g1+g3.m3.b2).d10+g3.m3.b2.d11
        +(g4.m3.b2.g1+g6.m3.b2).d12+(g5.m3.b2.g1+g6.m3.b2).d13))
        +[g1.d1]*.((g6.m3.b2.g1+g8.m3.b2).d14)   */
}
```

Listing 5.6: Example 3 with ETVO/interpreter

```
eH1=d2.[g1.d2]*
eH2=d1.[g2.d1]*
eG=d1.(d2.m3.eH1.b2.d2+b2.g1.d1.eH2.d2.m3.d1).d2
```

5.4 EXAMPLE 4

For the example in Figure 20, we can decompose into subsystems.

$$\begin{aligned} H_1 &= \delta^1(\gamma^2\delta^1)^* \\ H_2 &= \delta^3(\gamma^2\delta^3)^* \\ H_3 &= \delta^4(\gamma^3\delta^4)^* \end{aligned}$$

Then, the global transfer function is written $y = Gu$ with

$$G = \delta^1 \left(\beta_{<1,0,0,1,0>} H_2 \delta^2 \mu_{<0,1,0,1,0>} \oplus \beta_{<0,1,1,0,1>} H_3 \delta^2 \mu_{<1,0,1,0,1>} \right) \delta^3 H_1 \delta^3.$$

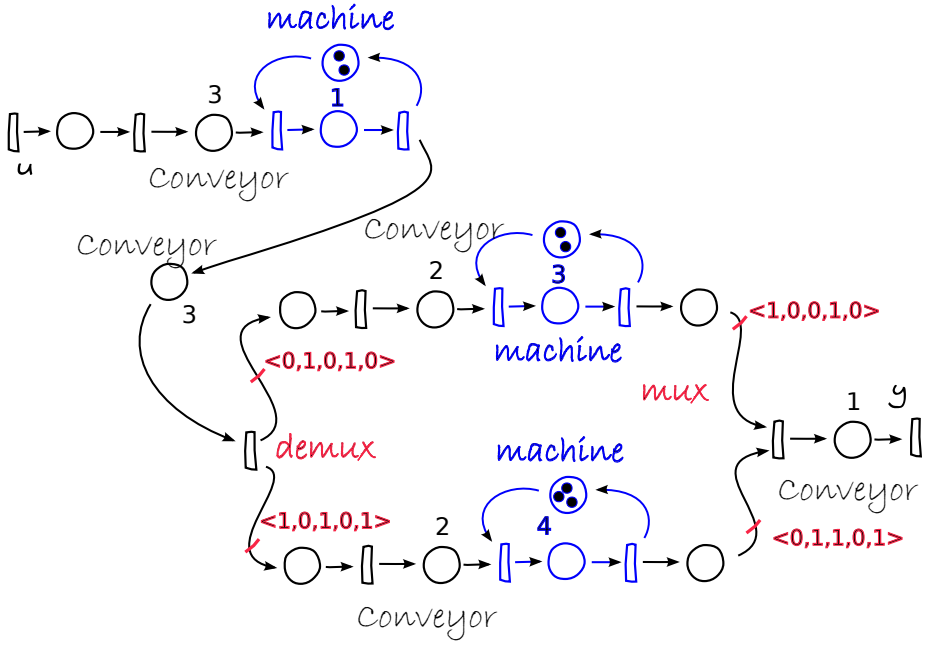

Figure 20: Weighted TEG (example 4) with mux/demux

Listing 5.7: Example 4 with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
 seriesEd G;
 series H1(poly::Epsilon(),gd(0,1),gd(2,1));
 series H2(poly::Epsilon(),gd(0,3),gd(2,3));
 series H3(poly::Epsilon(),gd(0,4),gd(3,4));

 G=eb({1,0,0,1,0})*seriesEd::toSeriesEd(H2)*ed(2)*em({0,1,0,1,0});
```

```
 G=G+eb({0,1,1,0,1})*seriesEd::toSeriesEd(H3)*ed(2)*em({1,0,1,0,1});
 G=ed(1)*G*ed(3)*seriesEd::toSeriesEd(H1)*ed(3);
 cout << "G=" << G << endl;
 cout << "G=" << G.toStringAsMuVar() << endl;
 /* output
 G=(((m5.b5.g3+g2.m5.b5.g2+g3.m5.b5.g1+g4.m5.b5).d13+(g1.m5.b5.g4+g2.m5.b5
     .g2+g4.m5.b5).d14
 +(g2.m5.b5.g4+g4.m5.b5.g2+g6.m5.b5.g1).d15+(g4.m5.b5.g4+g5.m5.b5.g3+g7.m5
     .b5.g1).d16))
 +[g5.d4]*.((g6.m5.b5.g4+g7.m5.b5.g2+g9.m5.b5).d18+(g7.m5.b5.g4+g9.m5.b5.
     g2+g11.m5.b5.g1).d19
 +(g9.m5.b5.g4+g11.m5.b5.g3+g12.m5.b5.g1).d20)
 G=((g0.m<0,2,1,1,1>.d13+g1.m<1,0,2,0,2>.d14+g2.m<2,0,2,1,0>.d15+g4.m
     <1,2,0,2,0>.d16))
 +[g5.d4]*.(g6.m<1,0,2,0,2>.d18+g7.m<2,0,2,1,0>.d19+g9.m<2,1,0,2,0>.d20)
 */
}
```

Listing 5.8: Example 4 with ETVO/interpreter

```
eH1=d1.[g2.d1]*
eH2=d3.[g2.d3]*
eH3=d4.[g3.d4]*
eG=d1.(b<1,0,0,1,0>.eH2.d2.m<0,1,0,1,0>+b<0,1,1,0,1>.eH3.d2.m<1,0,1,0,1>).
    d3.eH1.d3
asMuVar(eG)
```

5.5 EXAMPLE 5

For the example in Figure 21, we can decompose into subsystems.

$$\begin{aligned} H_1 &= \delta^6(\gamma^5\delta^6)^* \\ H_2 &= \delta^5(\gamma^4\delta^5)^* \\ H_3 &= \delta^7(\gamma^1\delta^7)^* \\ H_4 &= \delta^8(\gamma^2\delta^8)^* \end{aligned}$$

Then, the global transfer function is written $y = Gu$ with

$$G = \delta^1 \left(\beta_{<0,1>}\delta^1\mu_2 H_3\delta^1\mu_{<0,1,0>} \oplus \beta_{<1,0>}\delta^1 H_4\delta^1\mu_{<1,0,1>}\right)\beta_2\delta^3 H_2\delta^1 H_1\delta^1.$$

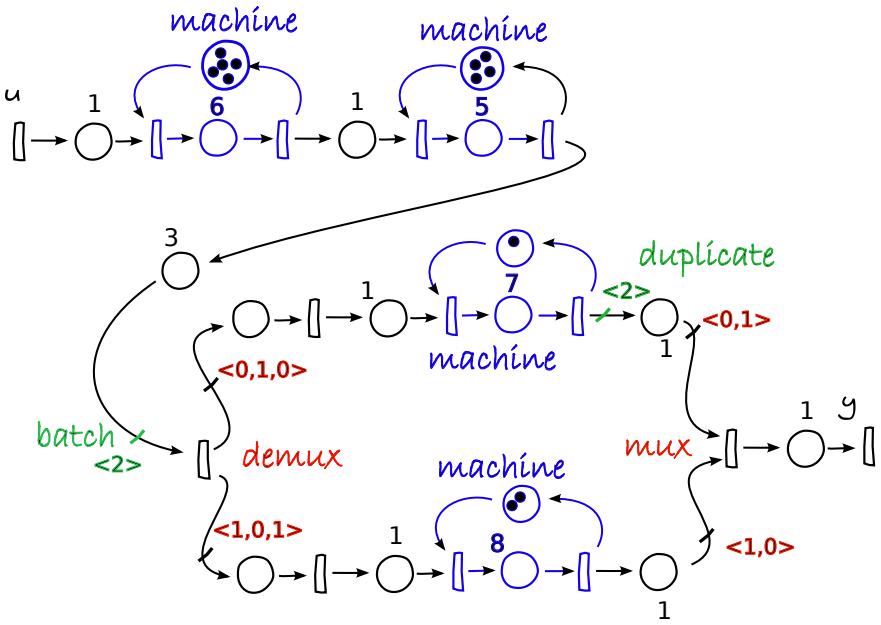

Figure 21: Weighted TEG (example 5)

Listing 5.9: Example 5 with ETVO/C++

```
#include "etvo.h"
using namespace std;
using namespace etvo;

int main()
{
 seriesEd G;
 series H1(poly::Epsilon(),gd(0,6),gd(5,6));
 series H2(poly::Epsilon(),gd(0,5),gd(4,5));
 series H3(poly::Epsilon(),gd(0,7),gd(1,7));
 series H4(poly::Epsilon(),gd(0,8),gd(2,8));

 G=eb({0,1})*ed(1)*em(2)*seriesEd::toSeriesEd(H3)*ed(1)*em({0,1,0});
 G=G+eb({1,0})*ed(1)*seriesEd::toSeriesEd(H4)*ed(1)*em({1,0,1});
 G=ed(1)*G*eb(2)*ed(3)*seriesEd::toSeriesEd(H2);
 G=G*ed(1)*seriesEd::toSeriesEd(H1)*ed(1);
 cout << "G=" << G << endl;
 cout << "G=" << G.toStringAsMuVar() << endl;
 /* partial output
```

```
 ((g0.m<0,1,0,1,0,2>.d26+g0.m<0,2,0,0,0,2>.d27+g2.m<0,2,0,1,0,1>.d31
 +g2.m<0,2,0,1,1,0>.d32+g2.m<2,0,1,1,0,0>.d33+g4.m<0,2,0,0,0,2>.d35
 +g5.m<0,1,0,2,0,1>.d36+g5.m<1,0,0,2,0,1>.d37+g6.m<0,0,2,0,1,1>.d38
 +g6.m<0,2,0,1,1,0>.d39+g6.m<0,2,0,2,0,0>.d40+g6.m<2,0,1,1,0,0>.d41
 +g8.m<0,1,1,0,0,2>.d42+g8.m<0,2,0,0,0,2>.d43+g9.m<1,0,0,2,0,1>.d44
 +g10.m<0,0,0,2,0,2>.d45+g10.m<0,0,2,0,1,1>.d46+g10.m<0,2,0,1,1,0>.d47
 +g10.m<0,2,0,2,0,0>.d48+g10.m<2,0,2,0,0,0>.d49+g12.m<0,2,0,0,0,2>.d51
 +g13.m<1,0,0,2,0,1>.d52))
 +[g4.d8]*.(g14.m<0,0,0,2,0,2>.d53+g14.m<0,0,2,0,2,0>.d54
                 +g14.m<0,2,0,2,0,0>.d56+g14.m<2,0,2,0,0,0>.d57
                 +g16.m<0,2,0,0,0,2>.d59)
 */
}
```

Listing 5.10: Example 5 with ETVO/interpreter

```
eH1=d6.[g5.d6]*
eH2=d5.[g4.d5]*
eH3=d7.[g1.d7]*
eH4=d8.[g2.d8]*
eG=d1.(b<0,1>.d1.m2.eH3.d1.m<0,1,0>+b<1,0>.d1.eH4.d1.m<1,0,1>)
eG=eG.b2.d3.eH2.d1.eH1.d1
asMuVar(eG)
```

5.6 EXAMPLE 6

The example in Figure 22 is a TEG with a periodic holding time. First, we can isolate the transfer functions of the two machines.

$$\begin{aligned} H_1 &= \delta^6(\gamma^5\delta^6)^* \\ H_2 &= \delta^5(\gamma^4\delta^5)^* \end{aligned}$$

Then, the transfer function becomes $y = Gu$ with

$$G = \delta^3 H_2 \delta^1 \delta^{<2,1,0,0,1>} \delta^1 H_1 \delta^1.$$

Listing 5.11: Example 6 with ETVO/C++

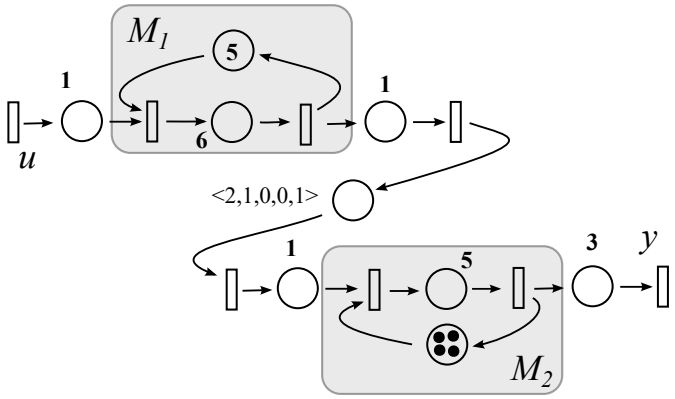

Figure 22: TEG with a periodic holding time

```
#include "etvo.h"
using namespace std;
using namespace etvo;
int main()
{
 seriesTg G;
 series H1(poly::Epsilon(),gd(0,6),gd(5,6));
 series H2(poly::Epsilon(),gd(0,5),gd(4,5));

 G=td(3)*seriesTg::toSeriesTg(H2)*td(1);
 G=G*td({2,1,0,0,1})*td(1)*seriesTg::toSeriesTg(H1)*td(1);
 cout << "G=" << G << endl;
 cout << "G=" << G.toStringAsDeltaVar() << endl;
 /* partial output
 G=((d<17,18,19,18,17>.g0+d<22,23,24,23,22>.g4+d<24,25,24,23,23>.g5
 +d<27,28,29,28,27>.g8+d<29,30,29,28,28>.g9+d<31,30,29,29,30>.g10))
 +[d5.g4]*.(d<32,33,34,33,32>.g12
 +d<34,35,34,33,33>.g13+d<36,35,34,34,35>.g14+d<36,35,35,36,37>.g15)
 */
}
```

Listing 5.12: Example 6 with ETVO/interpreter

```
tH1=d6.[g5.d6]*
tH2=d5.[g4.d5]*
tG=d3.tH2.d1.d<2,1,0,0,1>.d1.tH1.d1
asDeltaVar(tG)
```

5.7 EXAMPLE 7

The example in the Figure 23 combines Event-variant and Time-variant operators. This model describes the behavior of a system where two paths cross and the crossing is managed by a traffic light.

$$\begin{aligned} H_1 &= \delta^2(\gamma^3\delta^2)^* \\ H_2 &= \delta^4(\gamma^3\delta^4)^* \\ H_3 &= \delta^3(\gamma^2\delta^3)^* \\ H_4 &= \delta^4(\gamma^3\delta^4)^* \end{aligned}$$

Then, the transfer function becomes $y = \delta^1 G \delta^2 u$ with

$$G = \left(\beta_{<0,1>} H_3 \delta^1 \delta^{<0,0,2,1>} \delta^1 H_2 \mu_{<0,1>} \oplus \beta_{<1,0>} H_4 \delta^1 \delta^{<2,1,0,0>} \delta^1 H_1 \mu_{<1,0>}\right)$$

Using the ETVO tool, we can find another expression for this transfer function.

Listing 5.13: Example 7 with ETVO/interpreter

```
sH1=d2.[g3.d2]*
sH2=d4.[g3.d4]*
sH3=d3.[g2.d3]*
sH4=d4.[g3.d4]*
sG=d1.(b<0,1>.sH3.d1.d<0,0,2,1>.d1.sH2.m<0,1>+b<1,0>.sH4.d1.d<2,1,0,0>.d1.
    sH1.m<1,0>).d2
```

The transfer function can be written, for example, in the following form

$$G = H_a \mu_2 \beta_2 \gamma^1 \oplus H_b \mu_2 \beta_2,$$

with $H_a, H_b \in \mathcal{T}[\![\gamma]\!]$.

$$\begin{aligned} H_a &= (((\delta^8\Delta_4\delta^{-3} \oplus \delta^7\Delta_4) \oplus (\delta^9\Delta_4\delta^{-3} \oplus \delta^8\Delta_4)\gamma^1)) \\ &\quad \oplus [\delta^3\gamma^4]^*((\delta^{12}\Delta_4\delta^{-3} \oplus \delta^{11}\Delta_4)\gamma^5 + (\delta^{13}\Delta_4\delta^{-3} \oplus \delta^{12}\Delta_4)\gamma^7) \\ H_b &= (((\delta^9\Delta_4\delta^{-3} \oplus \delta^8\Delta_4)\gamma^1)) \oplus [\delta^3\gamma^4]^*((\delta^{12}\Delta_4\delta^{-3} \oplus \delta^{11}\Delta_4)\gamma^5 \oplus (\delta^{13}\Delta_4\delta^{-3} \oplus \delta^{12}\Delta_4)\gamma^7) \end{aligned}$$

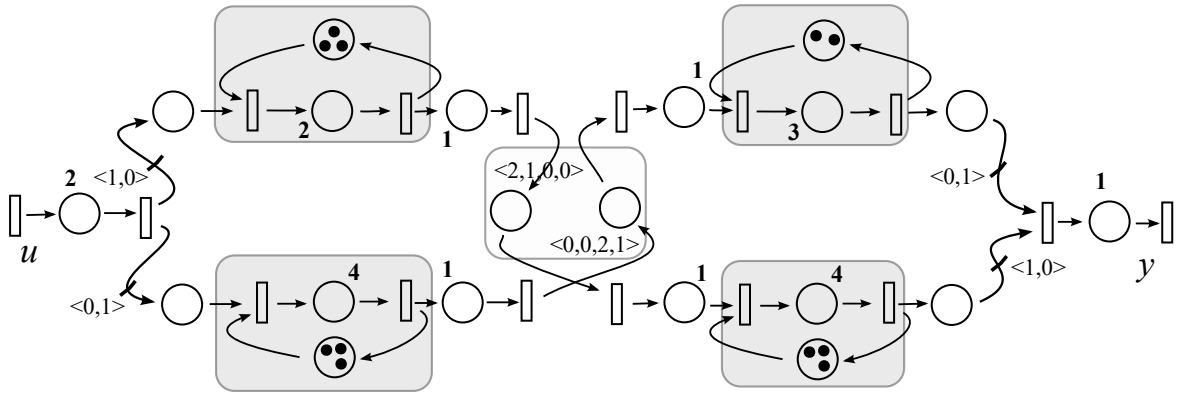

Figure 23: Weighted TEG with periodic holding times.

5.8 EXAMPLE 8

This last example is based on a more complex system. It is a production cell (see Figure 24) with a conveyor system. There is a periodic routing system in order to process the parts in two different workplaces (M_1 and M_2). In addition, the crossing of intersections C_1 and C_2 is only allowed within time windows (comparable to a traffic light).

The flow of parts is as follows:

- from A to B in 8 time units (the number of parts between A and the crossing C_1 is limited to 8).
- at B, parts can cross C_1 (1 time unit) only on dates in $4\mathbb{Z}$ and $4\mathbb{Z}+1$.
- from C_1 to C in 13 time units
- at C, the first part is assigned to M_1, the next two to M_2, this assignment is then repeated cyclically. If a part is assigned to M_1, the path is $C \to D \to E \to G$. If a part is assigned to M_2, the path is $C \to E \to F \to G$.
- The parts going from C to E (M_1) cross the parts going from D to G (M_2) at C_2. 2 time units are assigned to each path.
- at G, parts from M_1 and M_2 are recombined with a multiplexer.
- at H, parts can cross C_1 (1 time unit) only on dates in $4\mathbb{Z}+2$ and $4\mathbb{Z}+3$.

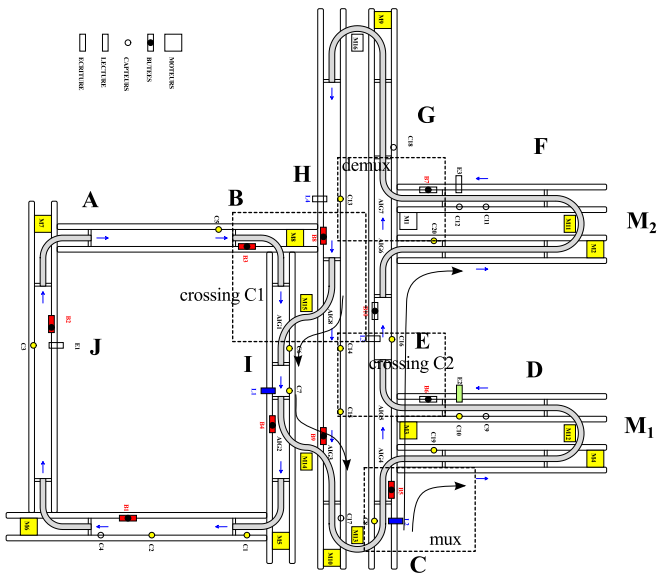

Figure 24: Production cell (example 8)

- from I to J in 10 time units

The system as a whole is both E-variant and T-variant. We can decompose into subsystems as follows:

$$
\begin{aligned}
H_{CA} &= \delta^{13}\left((\delta^{<3,2,1,1>}\gamma^{1})^{*}\delta^{<3,2,1,1>}\delta^{8}[\gamma^{8}(\delta^{<3,2,1,1>}\gamma^{1})^{*}\delta^{<3,2,1,1>}\delta^{8}]^{*}\right)\\
H_{GDC} &= \beta_{<1,0,0>}\delta^{7}\delta^{<0,0,2,1>}\delta^{4}\delta^{8}(\gamma^{1}\delta^{8})^{*}\delta^{10}\mu_{<1,0,0>}\\
H_{GFC} &= \beta_{<0,1,1>}\delta^{4}\delta^{5}(\gamma^{1}\delta^{5})^{*}\delta^{12}\delta^{<2,1,0,0>}\delta^{3}\mu_{<0,1,1>}\\
H_{JG} &= \delta^{10}(\delta^{<1,1,3,2>}\gamma^{1})^{*}\delta^{<1,1,3,2>}\delta^{4}
\end{aligned}
$$

Overall, we obtain the transfer function $y = Gu$ with

$$G = H_{JG}(H_{GFC} \oplus H_{GDC})H_{CA}.$$

Listing 5.14: Example 8 with ETVO/interpreter

```
sJG=d10.[d<1,1,3,2>.g1]*.d<1,1,3,2>.d4
sGDC=b<1,0,0>.d7.d<0,0,2,1>.d4.d8.[g1.d8]*.d10.m<1,0,0>
```

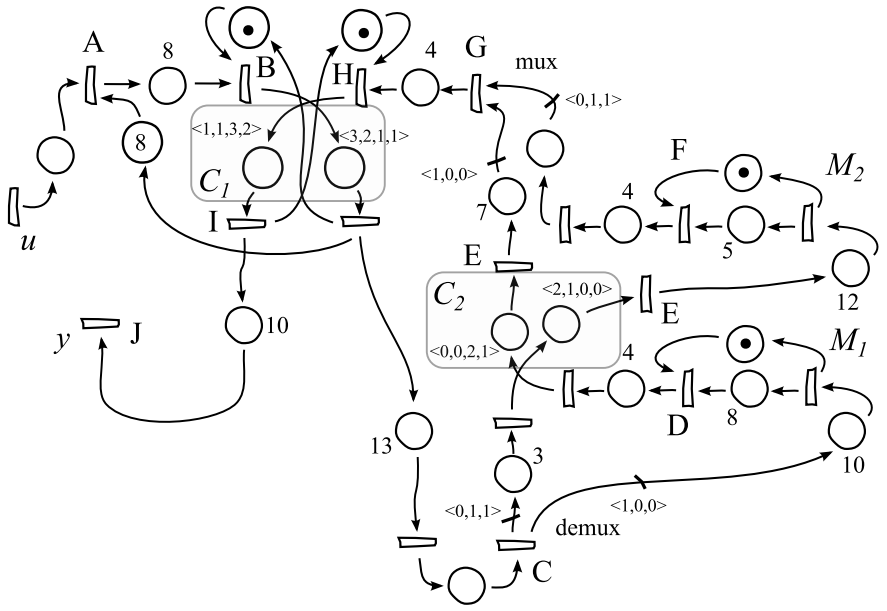

Figure 25

```
sGFC=b<0,1,1>.d4.[d5.g1]*.d5.d12.d<2,1,0,0>.d3.m<0,1,1>
sCA=d13.[d<3,2,1,1>.g1]*.d<3,2,1,1>.d8.[g8.[d<3,2,1,1>.g1]*.d<3,2,1,1>.d8
    ]*
sG=sJG.(sGDC+sGFC).sCA
```

BIBLIOGRAPHY

[1] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat. *Synchronization and Linearity: An Algebra for Discrete Event Systems*. John Wiley and Sons, New York, 1992.

[2] G. Bilsen, M. Engels, R. Lauwereins, and J. Peperstraete. Cycle-static dataflow. *IEEE Trans. on Signal Processing*, 44(2):397–408, 1996.

[3] G. Cohen. Analisis y control de sistemas de eventos discretos: de redes de petri temporizadas al algebra, 2001.

[4] G. Cohen, P. Moller, J.P. Quadrat, and M. Viot. Algebraic Tools for the Performance Evaluation of Discrete Event Systems. *IEEE Proceedings: Special issue on Discrete Event Systems*, 77(1):39–58, January 1989.

[5] B. Cottenceau, L. Hardouin, and J.-L. Boimond. Modeling and Control of Weight-Balanced Timed Event Graphs in Dioids. *IEEE Trans. on Autom. Cont.*, vol. 59:1219–1231, May 2014.

[6] Bertrand Cottenceau, Laurent Hardouin, and Johannes Trunk. Weight-balanced timed event graphs to model periodic phenomena in manufacturing systems. *IEEE Transactions on Automation Science and Engineering*, 14(4):1731–1742, 2017.

[7] E. Teruel, P. Chrzastowski, J.M. Colom, and M. Silva. On weighted T-systems. *LNCS 616*, pages 348–367, 1992.

[8] L. Hardouin, B. Cottenceau, and M.Lhommeau. Minmaxgd, a toolbox to handle periodic series in semiring minmax[[g,d]]., 2013.

[9] Laurent Hardouin, Bertrand Cottenceau, Ying Shang, Jörg Raisch, et al. Control and state estimation for max-plus linear systems. *Foundations and Trends® in Systems and Control*, 6(1):1–116, 2018.

[10] B. Heidergott, G.J. Olsder, and J.van der Woude. *Max Plus at Work - Modelling and Analysis of Synchronized Systems - A Course on Max-Plus Algebra and Its Applications*. Princeton University Press, Princeton, 2006.

[11] J. Trunk, B. Cottenceau, L. Hardouin, and J. Raisch. Model decomposition of weight-balanced timed event graphs in dioids: Application to control synthesis. In *IFAC World Congress*, Toulouse, France, 2017.

[12] Johannes Trunk. *On the modeling and control of extended Timed Event Graphs in dioids*. PhD thesis, Angers, 2019.

[13] Johannes Trunk, Bertrand Cottenceau, Laurent Hardouin, and Jörg Raisch. Output reference control for weight-balanced timed event graphs. In *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pages 4839–4846. IEEE, 2017.

[14] Johannes Trunk, Bertrand Cottenceau, Laurent Hardouin, and Jörg Raisch. Model decomposition of timed event graphs under partial synchronization in dioids. *IFAC-PapersOnLine*, 51(7):198–205, 2018.