An adequate logic for heterogeneous systems

Marc Aiguier
MAS Laboratory
École Centrale Paris
91 295 Châtenay-Malabry (France)
marc.aiguier@ecp.fr

Boris Golden
LIX Laboratory
École Polytechnique
91128 Palaiseau (France)
boris.golden@polytechnique.edu

Daniel Krob
LIX laboratory
École Polytechnique
91128 Palaiseau (France)
dk@polytechnique.edu

***Abstract*—We coalgebraically define a unified semantics for systems with an emphasis on the notion of time. Such a semantics intends to formalize system that underly system engineering (i.e. the discipline focusing on the integration mastery of large industrial systems). Moreover, we give a formal meaning to another important aspect of systems engineering : system requirements, constraining the expected properties of a system. To express such requirements, we define a logic that extends μ-calculus to our coalgebraic definition of systems. We establish an important property of this logic: adequacy.**

***Keywords*-System modelling; System semantics; System requirements; Timed Mealy machines; Coalgebras; μ-calculus; Adequacy**

I. Introduction

To manage the complexity of systems, a discipline called *systems engineering*, focused on the integration mastery of large industrial systems, has progressively emerged since the 50's. Systems engineering consists of a set of concepts, methods and good organizational and technical practices that the industry had to develop to be able to deal with the growing complexity of industrial systems (see [5], [15], [20], [22] for further details). However, systems engineering is just the application of a more general paradigm of thought, *systems approach*, a aim of which is to address dynamic system design homogeneously. At the heart of this paradigm is the notion of system which is generally described as a set of heterogeneous interconnected components[1], themselves recursively defined as systems, and interacting together to participate permanently to a same goal.
When mathematically apprehended, the concept of system is often defined with models coming from control theory and physics, that deal with systems as partial functions of dataflow transformation, so-called *transfer function*, of the form:

$$y = F(x, q, t)$$

where x, y and q are respectively vectors of input, output and state of the system usually defined as internal variables, and where t stands for time (considered in these approaches as continuous (see [4], [8], [21]). There is therefore a great challenge on being able to transfer to the systems engineering area, the formalization of systems taking into account the special feature of industrial systems to be software intensive, that is, providing advanced interaction with the external world through intelligent sensors and actuators based on software components. This then requires to unify in a same formal framework homogeneously and consistently mathematical methods dealing with the design of both continuous and discrete systems. To reach this purpose, we extend the approach developed in [6] where discrete and continuous times have been unified homogeneously by using techniques of non-standard analysis [9], [10], [11], [16], [17], into discrete models where steps are either infinitesimal or usual, in order to denote, respectively continuous and discrete times. The approach developed in [6] is both semantic and operational, in the sense that time scales associated to systems are subsets of the non-standard positive real numbers ${}^*\mathbb{R}^+$ (the time reference) and systems are extended Turing Machines. To address more abstractly system design, we deal with time axiomatically, that is by expressing the minimal properties that both time references and time scales have to satisfy. Hence, the key point on which our approach relies, is a common (discrete) model of time, inspired of the way time scales have been defined in [6] over a non-standard model of real numbers. This will allow us to avoid a trouble property standard in the theory of hybrid systems because based on the continuous time $\mathbb{R}$ [14]: the Zeno effect[2]. We further propose to specify system behaviors in an extension of the coalgebraic framework of Mealy machines so as it has been defined in [19]. The interest, as this has been shown strinkingly in [19], is that system behaviors can be modelled by causal transfer functions. We can then observe that transfer functions are causal functions following the sense given in [19], but taking into account different time scales between system inputs and outputs. Hence, by using standard results of the categorical theory of coalgebras [18], we will show how to define the causal function which

[1]The heterogeneity of components being mainly the result of the way they deal with dataflow, i.e. continuously or discretely, making of time a central element of their modelling.

[2]A hybrid system can change of state an infinite number of times within a finite time.

underlies any system or equivalently that the system under consideration implements. In the paper [3], to build new systems from existing ones, we have formally defined two operators that play a crucial role in systems engineering:[3]

1) *Integration operators* based on two basics operators: product and feedback
2) *Abstraction/simulation* operators.

The integration operator consists in building larger systems by making connections between inputs and outputs of more basic ones, and the abstraction/simulation operators allow to define abstractions of systems, so that they can be integrated in more global ones.

A last crucial aspect is still missing with a complete formalization of systems engineering : system requirements. In a formal approach such as defined in this paper, it is necessary to formally express such requirements to be able to address "scientifically" system correctness, i.e. checking that the system satisfies a set of requirements. Our formalism being based on an extension of Mealy machines with time scales, it is natural to define these requirements by temporal and real-time properties with the possibility to express constraints on the production time of output values from input ones. We then propose to extend a logic which subsumes most of modal and temporal logics: the μ-calculus. More precisely, we propose a variant of first-order fixed point modal logic [12], [23], and it is precisely the first order extension that will allow us to express real-time properties on the production time of output values from input ones. Of course, this logic will probably be restricted when we are interested in future works in its computational aspects such as system synthesis or the definition of model-checking algorithm. Here, being interested by theoretical results such as truth of formulas is preserved by bisimulations, the variant of first-order fixed point modal logic we propose is quite adequate.

The paper is structured as follows:

Section II gives the axiomatization of time references, and defines time scales and dataflows on which system behaviors will be defined. In Section III are defined both fundamental notions of transfer functions and systems. We will take the benefit of having formalized systems as coalgebras to extend in this section some standard results connected to the definition of a terminal model. This will then allow us to formally define system behaviors through transfer functions. We do not present in this paper our formalization of both integration and abstraction/simulation operators. Further details can be found in [3]. In Section IV, we present a logic whose interpretation will be over systems, and show that it is adequate with respect to bisimulation.

[3]In [3], the system definition is slightly different. It uses no coalgebraic notation, and manipulated input and output are not simple sets but are provided with some structures to read and write in a virtual buffer. Nevertheless, these two operators are easily definable in the formalism presented in this paper.

II. Time axiomatisation and dataflows

A. Time reference

By following an axiomatic approach to specify time carriers, we can accommodate several models of times including $\mathbb{N}$, $\mathbb{Z}^+$, $\mathbb{R}^+$, or ${}^*\mathbb{R}^+$,[4] as well as more pragmatic times such as the δ time of VHDL.

Definition 2.1 (Time reference): A **time reference** is any set T together with an internal law $+^T : T \to T$ and a pointed subset $(T^+, 0^T)$ satisfying the following conditions:

- upon T^+:
  - $\forall a, b \in T^+,\ a +^T b \in T^+$ closure (Δ_1)
  - $\forall a, b \in T^+,\ a +^T b = 0^T \Longrightarrow a = 0^T \wedge b = 0^T$ initiality (Δ_2)
  - $\forall a \in T^+,\ 0^T +^T a = a$ neutral to left (Δ_3)
- upon T:
  - $\forall a, b, c \in T,\ a +^T (b +^T c) = (a +^T b) +^T c$ associativity (Δ_4)
  - $\forall a \in T,\ a +^T 0^T = a$ neutral to right (Δ_5)
  - $\forall a, b, c \in T,\ a +^T b = a +^T c \Longrightarrow b = c$ cancelable to left (Δ_6)
  - $\forall a, b \in T,\ \exists c \in T^+,\ (a +^T c = b) \vee (b +^T c = a)$ linearity (Δ_7)

Elements of T are dates (or instants) whilst elements of T^+ are durations, or distances between instants. Any duration can be considered as an instant, by considering a conventional origin.

Using standard definitions, we note that $(T, +, 0^T)$ and $(T^+, +, 0^T)$ are monoid and submonoid, respectively, that further satisfy Δ_6 and Δ_2 for the latter. Moreover, also note that Δ_2 forbids inverse elements in T^+.

Example 2.1 (Non-standard real numbers): Following [6] which has inspired the works developed here, we can choose as time reference the set of non-standard real numbers ${}^*\mathbb{R}$ defined as the quotient of real numbers $\mathbb{R}$ under the equivalence relation $\equiv \subseteq \mathbb{R}^{\mathbb{N}} \times \mathbb{R}^{\mathbb{N}}$ defined by:

$$(a_n)_{n\geq 0} \equiv (b_n)_{n\geq 0} \Longleftrightarrow m(\{n \in \mathbb{N} | a_n = b_n\}) = 1$$

where m is an additive measure that separates between each subset of $\mathbb{N}$ and its complement, one and only one of these two sets being always of measure 1, and such that finite subsets are always of measure 0. The obvious zero element of ${}^*\mathbb{R}$ is $(0)_{n\geq 0}$, ${}^*\mathbb{R}^+$ is its positive part taken here as durations, and the internal law $+$ is defined as the usual addition on $\mathbb{R}^{\mathbb{N}}$, i.e.:

$$(a_n)_{n\geq 0} + (b_n)_{n\geq 0} = (a_n + b_n)_{n\geq 0}$$

${}^*\mathbb{R}$ being a totally ordered field that extends the set of standard reals identified to the class of the constant

[4]${}^*\mathbb{R}$ is the non-standard real numbers set [9], [16], [17]. Its definition is given in Example 2.1.

sequences by considering infinitesimal and infinite numbers, all the conditions of Definition 2.1 are satisfied. Observe also that ${}^*\mathbb{R}$ has as subset, the set of non-standard integers ${}^*\mathbb{Z}$ where infinite numbers are all ones having absolute value greater that any $n \in \mathbb{N}$.

Example 2.2 (VHDL time reference): Our axiomatisation is sufficiently broad to regard the δ time of VHDL as a possible model. The VHDL time reference [13] $\mathcal{V}$ is given by a couple of natural numbers (both sets of moments and durations are similar): the first number denotes the "real" time, the second number denotes the step number in the sequence of computations that must be performed at the same time – but still in a causal order. Such steps are called "δ-steps" in VHDL (and "micro-steps" in StateCharts). The idea is that when simulating a circuit, all independent processes must be simulated sequentially by the simulator. However, the real time (the time of the hardware) must not take these steps into account. Thus, two events e_1, e_2 at moments $(a,1),(a,2)$ respectively will be performed sequentially (e_1 before e_2) but at a same real time a. The VHDL addition is defined by the following rules:

$$(r' \neq 0) \Longrightarrow (r,d) + (r',d') = (r + r', d')$$
$$(r' = 0) \Longrightarrow (r,d) + (r',d') = (r, d + d')$$

where r, r', d and d' are natural numbers and $+$ denotes the usual addition on natural numbers. Clearly, the internal law $+$ above is not commutative, nor Archimedean: we may infinitely follow a δ-branch by successively adding δ-times[5].

The properties given upon T and T^+ are constraints that catch the intuitive view that the time elapses linearly by adding successively durations between them.

Proposition 2.1: Let us note $\preceq^T$ and $\prec^T$ the binary relations on T defined as follows:

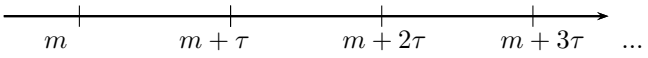
$$a \preceq^T b \Leftrightarrow \exists c \in T^+,\ b = a +^T c$$
$$a \prec^T b \Leftrightarrow \exists c \in T^+ \setminus \{0^T\},\ b = a +^T c$$

Then, $\preceq^T$ and $\prec^T$ are total orders on T.

In the following, $\preceq^T$ and $\prec^T$ will be simply noted $\preceq$ and $\prec$ when this does not raise ambiguity.

B. Time scale

Time references give the basic expected properties of any time carrier. Now, we can observe that time scales of systems are different. However, most of them can be unified as regular time scales of the form:



where τ is the step. For instance, in ${}^*\mathbb{R}$ defined in Example 2.1, by using results of non-standard analysis, continuous time scale can then be considered in a discrete way (see Example 2.3 below).

Definition 2.2 (Time scale): A **time scale** is any subset $\mathbb{T}$ of a reference time T such that:

- $\mathbb{T}$ has a minimum $m^{\mathbb{T}} \in \mathbb{T}$
- $\forall d \in T,\ \mathbb{T}_{d+} = \{d' \in \mathbb{T} \mid d' \succ d\}$ has a minimum $succ^{\mathbb{T}}(d)$
- $\forall d \in T$, when $d \succ m^{\mathbb{T}}$, the set $\mathbb{T}_{d-} = \{d' \in \mathbb{T} \mid d' \prec d\}$ has a maximum $pred^{\mathbb{T}}(d)$
- the principle of induction[6] is true on $\mathbb{T}$.

Given a timescale, let us define the mapping $\vec{_}_{\mathbb{T}} : T \to \mathbb{T}$ that from $d \in T$ yields the least $d' \in \mathbb{T}$ such that $d' \succeq d$. Hence, $d_{\mathbb{T}}$ is defined by: $\begin{cases} d & \text{if } d \in \mathbb{T} \\ succ^{\mathbb{T}}(d) & \text{otherwise} \end{cases}$

Example 2.3 (Discrete and continuous time scales): By using results of non-standard analysis, continuous time scales can then be considered in a discrete way. Following the approach developed in [6] to model continuous time by non-standard real numbers, a time scale can be ${}^*\mathbb{N}_\tau$ where $\tau \in {}^*\mathbb{R}^+$ is the step, $0 \in {}^*\mathbb{N}\tau$ and $\forall d \in {}^*\mathbb{N}\tau$, $succ^{{}^*\mathbb{N}\tau}(d) = d + \tau$. This provides a discrete time scale for modelling classical discrete time (when the step is not infinitesimal) and continuous time (when the step is infinitesimal)[7]. Indeed, by using transfer principle that claims basically that every usual property of standard real numbers holds for non-standard ones up to replacing standard sets by their non-standard equivalents, the recurrence principle can be immediately lifted to ${}^*\mathbb{R}$, up to working with ${}^*\mathbb{N}_\tau$ instead of $\mathbb{N}$.

Example 2.4 (VHDL time scale): In the VHDL time $\mathcal{V}$, the internal law induces a lexicographic ordering on $\mathbb{N} \times \mathbb{N}$. Thus, let $\mathcal{W} \subset \mathcal{V}$ such that: $\forall a \in \mathbb{N},\ \exists N_a \in \mathbb{N},\ \forall (a,b) \in \mathcal{W}, b \leq N_a$ (i.e. there are only a finite number of steps at each moment of time in $\mathcal{W}$). Then $\mathcal{W}$ is a time scale in the VHDL time.

Example 2.5 (Time scale over $\mathbb{R}$): A time scale on the time reference $\mathbb{R}^+$ can be any subset A such that: $\forall d, d' \in \mathbb{R}^+,\ |A \cap [d; d+d']|$ is finite.

Let us point out, although defined discretely, time scales are not necessarily order-isomorphic copy of natural numbers. Indeed, regular time-scales ${}^*\mathbb{N}_\tau$ in Example 2.3, when τ is infinitesimal, characterises continuous times and is noncoutable. Moreover, because discretely defined, each instant is not divisible *ad infinitum*, which avoids the Zeno effect *de facto*.

C. Dataflows

Definition 2.3 (Dataflow): Let T be a time reference. Let $\mathbb{T} \subseteq T$ be a time scale. A **$\mathbb{T}$-dataflow** over a set of values A is a mapping $f : \mathbb{T} \to A$. The set of all $\mathbb{T}$-dataflows over

[5]This is not the intended use of VHDL time, however: VHDL computations should perform a finite number of δ-steps.

[6]For $A \subset \mathbb{T}$, $\left(m^{\mathbb{T}} \in A \ \&\ \forall d \in A, succ^{\mathbb{T}}(d) \in A\right) \Rightarrow A = \mathbb{T}$.

[7]In [6] It has then been shown that every regular time scale with an infinitesimal step is continuous according to its non-standard meaning.

A is noted $A^{\mathbb{T}}$. The set of all dataflows over A with any possible time scale of T is noted $A^T = \bigcup_{\mathbb{T} \subseteq T} A^{\mathbb{T}}$.

The notion of derivative dataflow which is defined just below, will be useful to characterize the observable behaviors of systems.

Definition 2.4 (Derivative dataflow): Let T be a time reference. Let $\mathbb{T} \subseteq T$ be a time scale. Let f be a $\mathbb{T}$-dataflow over a set A. Let $d \in T$ be a date. The $\mathbb{T}$-dataflow f_d **derivative** of f **at** d is defined by:

- $\forall d' \prec d_{\mathbb{T}} \in \mathbb{T}, f'_d(d') = f(d')$
- $\forall d' \succeq d_{\mathbb{T}} \in \mathbb{T}, f_d(d') = f(succ^{\mathbb{T}}(d'))$

The next definition means that dataflows can be observed at any instant of time although their values only change at instants contained in their underlying time scale.

Definition 2.5 (Snapshots): Let T be a time reference and $\mathbb{T} \subseteq T$ be a time scale. Let f be a $\mathbb{T}$-dataflow over A and let $d \in T$ be an instant of time. The **snapshot of** f **at time** d, denoted $f :: d$, is the element $f(d_{\mathbb{T}})$ of A.

III. Transfer functions via coalgebras

Similarly to Rutten's works in [19], we show in this section that the behavior of systems can be characterized by causal functions mapping infinite input to infinite output sequences. Hence, observable behaviors of systems are given by causal transfer functions.

A. Transfer function

Observable behavior of systems will be specified through causal transfer functions. Transfer functions are timed data flow transformers satisfying the causality condition.

Definition 3.1 (transfer function): Let In and Out be two sets denoting, respectively, the values in input and in output. Let T be a time reference. Let $\mathbb{T} \subseteq T$ be a time scale giving rhythm data processing in input to yield output. A function $\mathcal{F} : In^T \to Out^{\mathbb{T}}$ is a **transfer function** if, and only if it is causal, that is:

$$\forall d \in T, \forall f, g \in In^T, (\forall d' \preceq d_{\mathbb{T}} \in T, f :: d' = g :: d') \implies \mathcal{F}(f) :: d = \mathcal{F}(g) :: d$$

Here, we define the technical notion of derivative function that will be useful to build final systems, and then to denote the observable behaviours of systems.

Definition 3.2 (Derivative function): Let $\mathcal{F} : In^T \to Out^{\mathbb{T}}$ be a transfer function. For every input $i \in In$ and every time $d \in \mathbb{T}$, we define the **derivative function** $\mathcal{F}_{(i,d)} : In^T \to Out^{\mathbb{T}}$ for every dataflow $f : \mathbb{T}' \to In$ with $\mathbb{T}' \subseteq T$ by:

$$\mathcal{F}_{(i,d)}(f) = \mathcal{F}((i,d) : f)_d$$

where $(i,d) : f : \mathbb{T}' \to In$ is the dataflow defined from f as follows:

- $\forall d' \prec d_{\mathbb{T}'} \in \mathbb{T}', (i,d) : f(d') = f(d')$
- $(i,d) : f(d_{\mathbb{T}'}) = i$
- $\forall d' \succ d_{\mathbb{T}'} \in \mathbb{T}', (i,d) : f(d') = f(pred^{\mathbb{T}'}(d'))$

Proposition 3.1: For every transfer function $\mathcal{F} : In^T \to Out^{\mathbb{T}}$, $\mathcal{F}_{(i,d)}$ is a transfer function.

Proof: Let $d_1 \in T$, let $f, g \in In^T$ such that for every $d_2 \preceq d_1 \in T$, $f :: d_2 = g :: d_2$. By construction, we also have that $((i,d) : f) :: d_2 = ((i,d) : g) :: d_2$. As $\mathcal{F}$ is causal, we can then write that $\mathcal{F}((i,d) : f) :: d_1 = \mathcal{F}((i,d) : g) :: d_1$.

Here, two cases have to be considered:

1) $d_1 \neq d$. In this case, we can directly conclude by $\mathcal{F}((i,d) : f)_{|_d} :: d_1 = \mathcal{F}((i,d) : g)_{|_d} :: d_1$.
2) $d_1 = d$. By construction, $((i,d) : f) :: d = ((i,d) : g) :: d$, and $((i,d) : f) :: succ^{\mathbb{T}}(d) = ((i,d) : g) :: succ^{\mathbb{T}}(d)$ since $f :: d_1 = g :: d_1$. We can then conclude $\mathcal{F}((i,d) : f) :: succ^{\mathbb{T}}(d) = \mathcal{F}((i,d) : g) :: succ^{\mathbb{T}}(d)$, i.e $\mathcal{F}((i,d) : f)_{|_d} :: d_1 = \mathcal{F}((i,d) : g)_{|_d} :: d_1$.

■

B. Systems as coalgebras

Definition 3.3 (Systems): Let In and Out be two sets denoting, respectively, the values in input and in output. Let T be a time reference. A **system** $\mathcal{S}$ is defined by a coalgebra (S, α) for the signature $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}} : Set \to Set$ where $\mathbb{T} \subseteq T$ is the time scale of $\mathcal{S}$, and a distinguished element q_0 denoting the intial state of the system $\mathcal{S}$.

A system $\mathcal{S}$ is called a **pre-system** when its initial state is removed.

In the following, given a system $((S, \alpha), q_o)$ over a signature $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$, we will note $\alpha(q)(i,d)_1$ (resp. $\alpha(q)(i,d)_2$) the resulting output value (resp. resulting state) of the couple $\alpha(q)(i,d)$.

Definition 3.4 (Category of systems): Let $\mathcal{S} = ((S, \alpha), q_0)$ and $\mathcal{S}' = ((S', \alpha'), q'_0)$ be two systems over $\mathcal{H}$. A **system morphism** $h : \mathcal{S} \to \mathcal{S}'$ is a coalgebra homomorphism $h : (S, \alpha) \to (S', \alpha')$ such that $h(q_0) = h(q'_0)$.

We note $Sys(\mathcal{H})$ (resp $PSys(\mathcal{H})$) the **category of systems** (resp. **of pre-systems**) over $\mathcal{H}$.

Below, we give a classical result over category of systems: the existence of a terminal system. This last point will be useful to give a trace model to systems via transfer functions.

Theorem 3.1: Let $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$ be a signature. Let Γ be the set of all transfer functions $\mathcal{F} : In^T \to Out^{\mathbb{T}}$. Let $\pi : \Gamma \to (Out \times \Gamma)^{In \times \mathbb{T}}$ defined for every $\mathcal{F} : In^T \to Out^{\mathbb{T}}$ and every $i \in In$ and every $d \in \mathbb{T}$ by $\pi(\mathcal{F})(i,d) = (\mathcal{F}((i,d) : f)(d), \mathcal{F}_{(i,d)})$ where $f \in In^T$ is arbitrary[8]. Then, the pre-system (Γ, π) is the final coalgebra in $PSys(\mathcal{H})$, that is for every pre-system (S, α) there exists a unique homomorphism $!_\alpha : (S, \alpha) \to (\Gamma, \pi)$.

Proof: For every pre-system (S, α), we define the function $!_\alpha : S \to \Gamma$ which for every $q \in S$ associates the

[8]This is correct because transfer functions are causal.

transfer function $!_\alpha(q) : In^T \to Out^{\mathbb{T}}$ defined as follows. Let $d \in \mathbb{T}$, and let $(m^{\mathbb{T}}, d_1, \ldots, d_n)$ such that for every i, $0 \leq i < n$, $d_{i+1} = succ^{\mathbb{T}}(d_i)$ with $d_0 = m^{\mathbb{T}}$ and $d_n = d$. Then, for every $f : \mathbb{T}' \to In \in In^T$, $!_\alpha(q)(f)(d)$ equals:
$\alpha(\alpha(\ldots(\alpha(\alpha(q)(f :: m^{\mathbb{T}}, m^{\mathbb{T}})_2)(f :: d_1, d_{1\mathbb{T}})_2)$
$(f :: d_2, d_{2\mathbb{T}})_2 \ldots)(f :: d_{n-1}, d_{n-1\mathbb{T}})_2)(f :: d_n, d_{n\mathbb{T}})_1$
It is not very difficult to verify that $!_\alpha(q)$ is causal, and the function $!_\alpha$ is a homomorphism which is further unique. ■

We call the transfer function $!_\alpha(q)$ above the **behaviour** of q, and then $!_\alpha(q_0)$ will be the behaviour of the system $((S, \alpha), q_0)$.

Conversely, given a transfer function $\mathcal{F} \in \Gamma$, we can build the minimal system $<\mathcal{F}>$ the behaviour of which is $\mathcal{F}$ as follow:

- $\mathcal{F}$ is the initial state.
- $<\mathcal{F}>$ is the set of transfer functions of Γ that contains $\mathcal{F}$ and is closed under transitions in (Γ, π) for any inputs and times.
- $\alpha_{\mathcal{F}} :<\mathcal{F}>\to (Out\times <\mathcal{F}>)^{In\times\mathbb{T}}$ is the mapping that associates to every $\mathcal{F}' \in<\mathcal{F}>$, every $i \in In$ and every $d \in \mathbb{T}$ the couple $\pi(\mathcal{F}')(i, d)$.

Proposition 3.2: $<\mathcal{F}>$ is the minimal system (i.e. it has the smallest number of states) such that $!_{\alpha_{\mathcal{F}}}(\mathcal{F}) = \mathcal{F}$.

Proof: The fact that $!_{\alpha_{\mathcal{F}}}(\mathcal{F}) = \mathcal{F}$ follows from the identity Id_Γ which is the unique homomorphism over Γ. To show that $<\mathcal{F}>$ is minimal, let us consider a system $((S, \alpha), q_0)$ such that $!_\alpha(q_0) = \mathcal{F}$. Let us define $<q_0>$ the subsystem $((S', \alpha'), q'_0)$ of $((S, \alpha), q_0)$ as follow:

- $q'_0 = q_0$
- S' is the set of states of S that contains q_0 and is closed under transitions in (S, α) for any inputs and times.
- $\alpha' : S' \to (Out \times S')^{In\times\mathbb{T}}$ is the mapping that associates to every $q \in S'$, every $i \in In$ and every $d \in \mathbb{T}$ the couple $\alpha(q)(i, d)$.

As $!_\alpha$ is a homomorphism, it directly follows that the image $!_\alpha(S)$ is a sub-presystem of (Γ, π). Moreover, by definition we have that $!_\alpha(S) =!_\alpha(< q_0 >) =<!_\alpha(q_0) >=< \mathcal{F} >$ whence we can conclude that $<\mathcal{F}>$ is the minimal system such that $!_{\alpha_{\mathcal{F}}}(\mathcal{F}) = \mathcal{F}$. ■

By Theorem 3.1 and Proposition 3.2, we will talk about systems and transfer functions indifferently.

Example 3.1: **[Discrete system]** First, let us observe that any deterministic Mealy machine can be represented in our formalism. Indeed, given a Mealy machine (S, α) with $\alpha : S \to (Out \times S)^{In}$, we can define the equivalent pre-system $\mathcal{S} = (S, \alpha')$ over the signature $(Out \times _)^{In\times\omega}$ by: $\forall n < \omega, \forall i \in In, \forall q \in Q, \alpha'(q)(i, n) = \alpha(q)(i)$.

Here, let us consider a more concrete example. We then propose to model a very simplified toothbrush viewed as a system, and some requirements over it within our framework. We set $\mathbb{R}$ as our time reference and work on the regular time scale $\mathbb{N}_\tau$ where the step τ stands for a hundredth of second.

The toothbrush has 2 input flows, $\mathcal{B}$ and $\mathcal{E}$, modelling respectively the button to control the toothbrush and the electricity coming by the power supply of the toothbrush.

The input $\mathcal{B}$ can take two values : 1 or 0, according to the state of the button (pressed or released). The input $\mathcal{E}$ can also take the two values 1 or 0, according to the presence or not of electricity allowing to supply the toothbrush.

Our toothbrush is modelled with one single output $\mathcal{R}$ figuring the rotation of the head designed to brush the teeth. The output $\mathcal{R}$ can take values in $\{0, 1, 2, 3, 4\}$ according to the speed of rotation (0 meaning no rotation and 4 being the highest speed of the head).

An oversimplified specification of the transfer function of the toothbrush can be the following:

- the toothbrush has 5 states : 0, 1, 2, 3, 4
- whatever the state of the system:
  - when $\mathcal{E} = 0$, then $\mathcal{R} = 0$ and the system returns to the state 0.
  - when $\mathcal{E} = 1$ and $\mathcal{B} = 0$ at any moment, then at the next step the state decreases of 1 (or 0 if it was 0), and $\mathcal{R}$ takes the value corresponding to the state.
  - when $\mathcal{E} = 1$ and $\mathcal{B} = 1$ at any moment, then at the next step the state increases of 1 (or remains to 4 if it was 4), and $\mathcal{R}$ takes the value corresponding to the state.

More formally, the system $\mathcal{S} = ((S, \alpha), q_0)$ is:

- $S = \{0, 1, 2, 3, 4\}$
- $q_0 = 0$
- $\alpha : S \to (Out \times S)^{In\times\mathbb{N}_\tau}$ with $Out = \{0, 1, 2, 3, 4\}$ and $In = \{0, 1\} \times \{0, 1\}$ is defined by: $\forall q \in \{0, 1, 2, 3, 4\}, \forall d \in \mathbb{N}_\tau$
  - $\alpha(q)((0, _), d) = (0, 0)$
  - $\alpha(q)((1, 0), d) =$
    $\begin{cases} (q-1, q-1) & \text{if } q \in \{1, 2, 3, 4\} \\ (0, 0) & \text{if } q = 0 \end{cases}$
  - $\alpha(q)((1, 1), d) =$
    $\begin{cases} (q+1, q+1) & \text{if } q \in \{0, 1, 2, 3\} \\ (4, 4) & \text{if } q = 4 \end{cases}$

Example 3.2: **[Continuous system]** It has been proved that any Hamiltonian system can be modelled within the framework introduced in [6]. As our definition of system generalizes the work of this first paperIt is out of the scope of this paper to prove it, however the proof is not difficult as one can notice that non-standard time scales defined in [6] are still time scales in our new model, and that transitions defined in [6] can be rewritten as transitions in our model., we recall here a simplified example of a Water Tank given in [6], which is a well-known example of the hybrid systems and control theory literature.

We work in the time reference ${}^*\mathbb{R}$ of nonstandard real numbers. Let us fix first some regular continuous time scale

$\mathbb{T}$ with infinitesimal time step τ (i.e. $\mathbb{T} = {}^*\mathbb{N}_\tau$). We consider a water tank where water arrives at a variable rate $w_i(d) \geq 0$ (with $d \in \mathbb{T}$) through one single pipe. The water leaves through another (output) pipe at rate $w_o(d)$ (with $d \in \mathbb{T}$) controlled by a valve whose position is given by $v(d) \in [0,1]$ (with $d \in \mathbb{T}$), 0 and 1 modelling respectively here the fact that the valve is closed or open. The water tank can be modelled as a system, taking on input the current values of the incoming water flow $w_i(d)$ and the position $v(d)$ of the valve and sending on its output the corresponding output water flow $w_o(d)$ and water level $l(d)$ according to the following equations: for every $d \in \mathbb{T} \setminus \{0\}$,

$$\begin{array}{c} w_o(0) = C\,V_0, \quad w_o(d+\tau) = C\,v(d) \\ l(0) = L_0, \quad l(d+\tau) = l(d) + (w_i(d) - w_o(d))\,\tau \end{array}$$

where C is the maximal throughout capacity of the output pipe and V_0 is the initial position of the valve at the time $m^{\mathbb{T}} = 0$.
The input and output spaces of the system are thus $In = [0,C] \times [0,1]$ and $Out = [0,C] \times [L_1, L_2]$ where $[L_1, L_2]$ is the interval in which the level of water in the tank at each time (given by $l(d)$) has to belong. Hence, the initial water tank level $L_0 \in [L_1, L_2]$.
This illustrates the modelling of a simple physical system in our framework. Modelling of more complex physical systems can be found in [6].

Example 3.3: **[Hybrid system]** Let us consider a classic hybrid system that models the physical behaviour of a lamp. Three modes are modelled here: The "Init" mode (the switch button was never touched), the "On" mode (the lamp was switched on at least once) and the "Off" mode (the lamp was switched off at least once). The states corresponding to these different modes then contain the three generic evolution mode modelled by ordinary differential equations (see below) of the continuous signal $y(d)$ that represents the output lamp energy at each moment of time d. Hence, the inputs will be $In = \{Init, On, Off\} \times \{\rho, \pi\}$ where ρ and π model respectively the fact that the button is either pressed or released, and the outputs will be $Out = \mathbb{R}^+ \times \{On, Off\}$. To model such a system, we work in the time reference ${}^*\mathbb{R}$ of nonstandard real numbers and for some regular continuous time scale $\mathbb{T}$ with infinitesimal time step τ.
The lamp can be modelled as a system, taking on input the current mode of the lamp $M(d)$ and the fact that the button is pressed or released $B(d)$, and sending on its output the corresponding output lamp energy $y(d)$ and the new mode $M(d)$ according to the following equations:

$$y(0) = 0$$

$$y(d+\tau) = \begin{cases} 0 & \text{if } M(d) = Init \text{ and } B(d) = \rho \\ \tau e - y(d)(\tau k + 1) & \text{if } (M(d) = On \text{ and } B(d) = \rho) \text{ or } (M(d) \in \{Init, Off\} \text{ and } B(d) = \pi) \\ y(d)(1 - \tau k) & \text{if } (M(d) = On \text{ and } B(d) = \pi) \text{ or } (M(d) = Off \text{ and } B(d) = \rho) \end{cases}$$

where e is the energy level produced by the lamp, and k is a real parameter to express the speed of the light compared to the state of the button.

$$M(0) = Init$$

$$M(d+\tau) = \begin{cases} Init & \text{if } M(d) = Init \text{ and } B(d) = \rho \\ On & \text{if } (M(d) = On \text{ and } B(d) = \rho) \text{ or } (M(d) \in \{Init, Off\} \text{ and } B(d) = \pi) \\ Off & \text{if } (M(d) = On \text{ and } B(d) = \pi) \text{ or } (M(d) = Off \text{ and } B(d) = \rho) \end{cases}$$

Systems can be aggregated to make larger ones. Two basic connectors (feedback and Cartesian product) have been defined in [2], [3] easily adaptable to our framework. The composition of theses basic connectors have been shown sufficient to define most other connectors such as sequential composition and synchronous product. For lack of space, we do not present these operators in this paper, and refer interested readers to our papers [2], [3].Besides, a complete description of the hybrid system in Example 3.3 would require a feedback on the output mode over the input one.

Next, we define the notion of bisimulation over which we will show the adequateness of the logic presented in the paper. Systems being defined by using coalgebraic notations, we define bisimulations for systems following notations in [1], [18]. Hence, a bisimulation between two systems is a transition structure respecting relation between sets of states.

Definition 3.5 (Bisimulation): Let $\mathcal{S}_1 = ((S_1, \alpha_1), q_0^1)$ and $\mathcal{S}_2 = ((S_2, \alpha_2), q_0^2)$ be two systems over a signature $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$. A subset $R \subseteq S_1 \times S_2$ is a **bisimulation** if, and only if $(q_0^1, q_0^2) \in R$ and there exists a mapping $\alpha_R : R \to \mathcal{H}(R)$ such that both projections from R to S_1 and S_2 are coalgebra morphisms:

$$\begin{array}{ccccc} S_1 & \xleftarrow{\pi_1} & R & \xrightarrow{\pi_2} & S_2 \\ {\scriptstyle\alpha_1}\downarrow & & \downarrow{\scriptstyle\alpha_R} & & \downarrow{\scriptstyle\alpha_2} \\ \mathcal{H}(S_1) & \xleftarrow{\mathcal{H}(\pi_1)} & \mathcal{H}(R) & \xrightarrow{\mathcal{H}(\pi_2)} & \mathcal{H}(S_2) \end{array}$$

$\mathcal{S}_1$ and $\mathcal{S}_2$ are said **bisimilar** if, and only if there exists a bisimulation between them.

All the basic facts on bisimulations remain true in our framework. Among others, the greatest bisimulation between

$\mathcal{S}_1$ and $\mathcal{S}_2$, noted $\sim_{\mathcal{S}_1,\mathcal{S}_2}$ or simply $\sim$ when the context is clear, exists and is defined as the union of all bisimulations between $\mathcal{S}_1$ and $\mathcal{S}_2$.

IV. Logic for systems

We present in this section a logic $\mathcal{L}$ whose the interpretation will be over systems. $\mathcal{L}$ is a slight extension of μ-calculus to input and output values and times. The interest of μ-calculus is its greats increase in expressive power. Indeed, it includes many of modal logics commonly used in verification of reactive and distributed systems.

A. Syntax and satisfaction

The logic $\mathcal{L}$ being an extension of μ-calculus that is known to subsume most of modal and temporal logics, it will allow to express standard properties over reactive systems such reactiveness, liveness, safety, etc. Now, time being explicit in our framework, $\mathcal{L}$ must also allow to express both real-time properties on the production time of output values from input ones, and properties on the input or output value reading from both dataflow and date. This requires a language to express time expressions. By our axiomatization, it is natural to define such expressions as first-order terms with variables over the mono-sorted first-order signature $\Sigma = (F, R)$ where the set of function names $F = \{succ^1, pred^1, +^2\} \cup \{d^0 | d \in T\}$ and the set of predicates $R = \{\prec^2, \preceq^2\}$ for T a time reference. [9] Hence, time terms will be element in the set $T_\Sigma(V)$ which is the set of all terms freely generated from the signature Σ and a set V of time variables. A model for Σ or Σ-model is any first-order structure $(\mathbb{T}, succ^{\mathbb{T}}, pred^{\mathbb{T}}, +^{\mathbb{T}}, \prec^{\mathbb{T}}, \preceq^{\mathbb{T}})$ where $\mathbb{T} \subseteq T$ is a time scale, $+^{\mathbb{T}} : (d, d') \mapsto (d +^T d')_{\mathbb{T}}$, $\prec^{\mathbb{T}} = \prec^T_{|_{\mathbb{T}}}$ and $\preceq^{\mathbb{T}} = \preceq^T_{|_{\mathbb{T}}}$. In the following, this model will simply note $\mathbb{T}$ when this does not raise ambiguity.
Given an interpretation of variables $\iota : V \to T$ and a time term $t \in T_\Sigma(V)$ variables of which are among $\{x_1, \ldots, x_n\}$, the evaluation of t for ι in $\mathbb{T}$, noted $[\![t]\!]^{\mathbb{T}}_\iota$, is the evaluation of $\iota(t)$ by interpreting $succ$, $pred$, $+$, $\prec$ and $\preceq$ by $succ^{\mathbb{T}}$, $prec^{\mathbb{T}}$, $+^{\mathbb{T}}$, $\prec^{\mathbb{T}}$ and $\preceq^{\mathbb{T}}$, respectively, and every constant d by $d_{\mathbb{T}}$. [10]

In the next definition, we need a set of supplementary variables, called fixed point variables, to express formulas in μ-calculus that denote recursion on states. To differentiate these variables with those in V, we will denote in the following variables in V by the letters $x, x', x_1, x_2, \ldots, y, y', y_1, y_2, \ldots$ whilst fixed point variables will be denoted by $\overline{x}, \overline{x}', \overline{x}_1, \overline{x}_2, \ldots, \overline{y}, \overline{y}', \overline{y}_1, \overline{y}_2, \ldots$.

Definition 4.1 (Input and output terms): Let T be a reference time.Let V be a set of time variables. Let $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$ be a signature with $\mathbb{T} \subseteq T$. **Input terms** (resp. **output terms**) over $\mathcal{H}$ are all inputs in In (resp. outputs in Out) and all expressions of the form $_ ::_{In} t$ (resp. $_ ::_{Out} t$) where $t \in T_\Sigma(V)$.

Input expressions $_ ::_{In} t$ (resp. $_ ::_{Out} t$) denote the content of input (resp. output) dataflows at the date t.

Definition 4.2 (System formulas): Let T be a reference time.Let V be a set of time variables. Let X be a set of fixed points variables. Let $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$ be a signature with $\mathbb{T} \subseteq T$. **System formulas** are defined as follows:

$$\varphi := \Theta |\ E = E' \mid i \downarrow_t o |\ \overline{x} \mid [(i,t)]\varphi | \neg\varphi | \varphi_1 \wedge \varphi_2 | \exists x.\varphi | \nu\overline{x}.\psi$$

where Θ is a first-order formula built over the signature $\Sigma = (F, R)$ and the set of variables V, E and E' are either both input terms or both output terms, $i \in In$, $t \in T_\Sigma(V)$, $x \in V$, $\overline{x} \in X$ and ψ is a formula that may contain occurrences of the variable $\overline{x}$ provided that every free occurrences of $\overline{x}$ occurs positively in ψ, i.e. within the scope of an even number of negations [11].
A formula φ is **closed** when every time variable x and every fixed point variable $\overline{x}$ are in the scope of a quantifier $\exists x$ and an operator $\nu\overline{x}$, respectively.

Intuitively, atoms of the form $E = E'$ check the content of input or output dataflows at different dates, and atoms of the form $i \downarrow_t o$ stand for output formula and check that it is possible to produce the output o after performing the input i at the date t. A formula of the form $[(i,t)]\varphi$ stands for a state formula, and states that after preforming an input i at the date t, all reachable states satisfy φ. Finally, a formula of the form $\nu\overline{x}.\psi$ stands for a formula that expresses a recursion on states and is defined semantically as a function with fixpoints. More precisely, a formula φ of the logic can be semantically defined by a function $f_\varphi : \mathcal{P}(S)^n \to \mathcal{P}(S)$ where occur freely the fixed point variables $\overline{x}_1, \ldots, \overline{x}_n$ in φ, that given n subsets $S_1, \ldots, S_n$ yields the set of states that satisfy φ. Therefore, a formula φ of the form $\nu\overline{x}.\psi$ that can be seen as a "looping", denotes the greatest fixpoint of the function $f_\varphi : \mathcal{P}(S) \to f_\psi$ (see below). It is well-known that such a fixpoint exists when f is monotonic on $\mathcal{P}(S)$. The condition that every free occurrences of $\overline{x}$ occurs positively in ψ, ensures monotonicity [7].

The least fixpoint operator μ is obtained standardly:

$$\neg\nu\overline{x}.\varphi \Leftrightarrow \mu\overline{x}.\neg\varphi'$$

where φ' is the formula obtained from φ by substituting $\neg\overline{x}$ for $\overline{x}$ in all free occurrences of $\overline{x}$ in φ.

Standardly, the universal quantifier is defined:

$$\neg\exists.\varphi \Leftrightarrow \forall x.\neg\varphi$$

[9] The exponents attached to function and predicate names indicate their arty. Hence, any date $d \in T$ is considered as a constant function.

[10] $\iota(t)$ is the term obtained from t by replacing every variable x_i by its value $\iota(x_i)$ taken as a constant.

[11] The notions of free and bound variables are usual where ν is the only binding operator.

Example 4.1: we propose to express the following requirement, insuring that the toothbrush reacts quickly to a pression on its button: *(Reactiveness) when the button is pressed and that there is electricy powering, the toothbrush modifies its output within* 0.1 *second except if it is already at the highest speed. In this case, it remains to this highest speed while the button is pressed*. This requirement can be expressed as follows in our framework:

$$\begin{array}{l}_ ::_{In}\ x = (1,1) \wedge \neg(_ ::_{Out}\ x = 4) \\ \qquad\qquad \Rightarrow \exists y.\ y \leq 10\ \wedge\ \neg(_ ::_{Out}\ (x+y) = _ ::_{Out}\ x)\end{array}$$

$$_ ::_{In}\ x = (1,1) \wedge _ ::_{Out}\ x = 4 \Rightarrow (\forall y.y \succeq x \Rightarrow \left\{\begin{array}{c} \mu\overline{x}._ ::_{In}\ y = (_,0) \\ \vee \\ ([((1,1),y)]\overline{x} \wedge _ ::_{Out}\ succ(y) = 4)\end{array}\right.$$

We propose to model another requirement, imposing that the toothbrush would stop rapidly if the electricity supply is stopped: *(Inerty) when there is no power supply, the toothbrush's head stops its rotation within 1 second*. This requirement can be expressed as follows:

$$_ ::_{In}\ x = (0,_) \Rightarrow \exists y.\ y \leq x + 100\ \wedge\ _ ::_{Out}\ y = 0$$

We introduce a last requirement constraining the speed of the toothbrush when it is working: *(Performance) when the toothbrush is working properly, constantly the speed of rotation must be* 1*,* 2*,* 3 *or* 4. This requirement can be expressed as follows:

$$_ ::_{In}\ x = (1,_) \Rightarrow \bigvee_{1 \leq i \leq 4} _ ::_{Out}\ x = i$$

These 3 requirements are typical properties expected to be verified from systems.

Definition 4.3 (Input output terms evaluation): Let $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$ with $\mathbb{T} \subseteq T$. Let V be a set of time variables. Let $\iota : V \to T$ be a time variable valuation. Let $f : \mathbb{T}' \to In$ be an input dataflow in In^T. Let E be an input term over $\mathcal{H}$. The **evaluation** of E for f, noted $[\![E]\!]^{\mathbb{T}'}_{(\iota,f)}$ is defined on the structure of E as follows:

- $[\![i]\!]^{\mathbb{T}'}_{(\iota,f)} = i$ for $i \in In$
- $[\![_ ::_{In}\ t]\!]^{\mathbb{T}'}_{(\iota,f)} = f([\![t]\!]^{\mathbb{T}'}_{\iota})$.

Let $g : \mathbb{T} \to Out$ be an output dataflow. Let E be an output term over $\mathcal{H}$. The **evaluation** of E for g, noted $[\![E]\!]^{\mathbb{T}}_{(\iota,g)}$ is defined on the structure of E as follows:

- $[\![o]\!]^{\mathbb{T}}_{(\iota,g)} = o$ for $o \in Out$
- $[\![_ ::_{Out}\ t]\!]^{\mathbb{T}}_{(\iota,g)} = g([\![t]\!]^{\mathbb{T}}_{\iota})$.

Classically, the semantics of μ-calculus formulas is standardly defined by associating to each formula φ the set of states for which φ is true[12]. This kind of semantics can be easily extended to our logic equivalently to Definition 4.4 just below. However, Definition 4.4 is a more classical definition of satisfaction $\models$ defined as a binary relation between systems and formulas.

Definition 4.4 (Satisfaction): Let $\mathcal{H} = (Out \times _)^{In \times \mathbb{T}}$ be a signature with $\mathbb{T} \subseteq T$. Let $\mathcal{S} = ((S, \alpha), q_0)$ be a system over $\mathcal{H}$. Let φ be a formula over $\mathcal{H}$. For every valuation $\lambda : X \to \mathcal{P}(S)$, every interpretation of variables $\iota : V \to T$, every state $q \in S$ and every input dataflow $f : \mathbb{T}' \to In \in In^T$. $\mathcal{S}$ **satisfies** for f, q, ι and λ the formula φ, noted $\mathcal{S} \models_{f,q,\iota,\lambda} \varphi$ if, and only if:

- if φ is a first-order formula Θ over Σ, then $\mathcal{S} \models_{f,q,\iota,\lambda} \Theta$ iff $\mathbb{T} \models_{\iota} \Theta$.
- if φ is an atom of the form $E = E'$ where E and E' are input terms, then $\mathcal{S} \models_{f,q,\iota,\lambda} E = E'$ iff $[\![E]\!]^{\mathbb{T}'}_{(\iota,f)} = [\![E']\!]^{\mathbb{T}'}_{(\iota,f)}$.
- if φ is an atom of the form $E = E'$ where E and E' are output terms, then $\mathcal{S} \models_{f,q,\iota,\lambda} E = E'$ iff $[\![E]\!]^{\mathbb{T}}_{(\iota,!_\alpha(q)(f))} = [\![E']\!]^{\mathbb{T}}_{(\iota,!_\alpha(q)(f))}$.
- if φ is an atom of the form $i \downarrow_t o$, then $\mathcal{S} \models_{f,q,\iota,\lambda} i \downarrow_t o$ iff $\alpha(q)(i, [\![t]\!]^{\mathbb{T}}_{\iota})_1 = o$. [13]
- if φ is a fixed point variable $\overline{x}$, then $\mathcal{S} \models_{f,q,\iota,\lambda} \overline{x}$ iff $q \in \lambda(\overline{x})$.
- if $\varphi = [(i,t)]\varphi'$, then $\mathcal{S} \models_{f,q,\iota,\lambda} [(i,t)]\varphi'$ iff $\mathcal{S} \models_{(i,[\![t]\!]^{\mathbb{T}'}_{\iota}):f,q',\iota,\lambda} \varphi'$ and $q' = \alpha(q)(i, [\![t]\!]^{\mathbb{T}}_{\iota})_2$.
- if $\varphi = \nu\overline{x}.\psi$, then $\mathcal{S} \models_{f,q,\iota,\lambda} \nu\overline{x}.\psi$ iff $\exists S' \subseteq S$, $q \in S'$ and $\forall q' \in S'$, $\mathcal{S} \models_{f,q',\iota,\lambda[S'/\overline{x}]} \psi$.
  Here, $\lambda[S'/\overline{x}]$ is the valuation such that $\lambda[S'/\overline{x}](\overline{x}) = S'$ and $\lambda[S'/\overline{x}](\overline{x}') = \lambda(\overline{x}')$ for every $\overline{x}' \neq \overline{x}$.
- propositional connectors and first-order quantifier are handled as usual.

$\mathcal{S}$ **satisfies** a formula φ, noted $\mathcal{S} \models \varphi$, if, and only if for every $f \in In^T$, every $\iota : V \to T$ and every valuation $\lambda : X \to \mathcal{P}(S)$, $\mathcal{S} \models_{f,q_0,\iota,\lambda} \varphi$.

From Definition 4.4, it is obvious to show that for every closed formula φ, every state $q \in S$ and every input dataflow $f \in In^T$,

$$\forall \lambda : X \to \mathcal{P}(S), \forall \iota : V \to T, \mathcal{S} \models_{f,q,\iota,\lambda} \varphi \Leftrightarrow \mathcal{S} \models_{f,q,\emptyset} \varphi$$

where $\emptyset : X \to \mathcal{P}(S)$ is the valuation that associates to every $\overline{x} \in X$ the emptyset $\emptyset$.

Let us show that $\mathcal{L}$ is expressive enough to characterise bisimilarity, that is two systems $\mathcal{S}_1$ and $\mathcal{S}_2$ are bisimilar when they are elementary equivalent and vice versa, where **elementary equivalence** means that:

$$\forall \varphi, \mathcal{S}_1 \models \varphi \Leftrightarrow \mathcal{S}_2 \models \varphi$$

Theorem 4.1: Let $\mathcal{S}_1 = ((S_1, \alpha_1), q_0^1)$ ad $\mathcal{S}_2 = ((S_2, \alpha_2), q_0^2)$ be two systems over $(Out \times _)^{In \times \mathbb{T}}$. Then,

[12] returning equally to define a function $f_\varphi : \mathcal{P}(S)^n \to \mathcal{P}(S)$ as previously where n is the set number of free fixed point variables in φ.

[13] Similarly, we could also write that $\mathcal{S} \models_{f,q,\iota,\lambda} \varphi$ iff $!_\alpha(q)((i, [\![t]\!]^{\mathbb{T}'}_{\iota}) : f)([\![t]\!]^{\mathbb{T}}_{\iota}) = o$.

$\mathcal{S}_1$ and $\mathcal{S}_2$ are elementary equivalent if, and only if they are bisimilar.

Proof: To prove the **only if** implication, let us suppose that $q_0^1 \sim q_0^2$. Let $\lambda_2 : X \to \mathcal{P}(S_2)$. Let us define $\lambda_1 : X \to \mathcal{P}(S_1)$ by:

$$\lambda_1(\overline{x}) = \{q_1 | \exists q_2 \in \lambda_2(\overline{x}), q_1 \sim q_2\}$$

It is quite obvious to show by structural induction on formulas that for every φ,

$$\mathcal{S}_1 \models_{f,q_0^1,\iota,\lambda_1} \varphi \Leftrightarrow \mathcal{S}_2 \models_{f,q_0^2,\iota,\lambda_2} \varphi$$

We can apply the same reasoning from any valuation $\lambda_1 : X \to \mathcal{P}(S_1)$.

For the converse (the **if part**), let us define the relation $\equiv \subseteq S_1 \times S_2$ as follows: $q \equiv q'$ iff for every $f \in In^T$, every $\iota : V \to T$ and every $\lambda : X \to \mathcal{P}(S_1)$,

$$\forall \varphi, \mathcal{S}_1 \models_{f,q,\iota,\lambda} \varphi \Leftrightarrow \mathcal{S}_2 \models_{f,q',\iota,\lambda'} \varphi$$

where $\lambda' : X \to \mathcal{P}(S_2)$ is the mapping that associates the set $\{q' | \exists q \in \lambda(\overline{x}), q \equiv q'\}$ to each $\overline{x} \in X$. Let us show that $\equiv \subseteq \sim$. Let us suppose that $q \equiv q'$. By definition, this means for every $i \in In$ and every $d \in T$ that $\alpha_1(q)(i, d_{\mathbb{T}})_1 = \alpha_2(q')(i, d_{\mathbb{T}})_1$. It remains to prove that $\alpha_1(q)(i, d_{\mathbb{T}})_2 \equiv \alpha_2(q')(i, d_{\mathbb{T}})_2$. Let us suppose the opposite. This means there exists a formula ψ, a dataflow $f : \mathbb{T}' \to In \in In^T$, a variable interpretation $\iota : V \to T$ and a valuation $\lambda : X \to \mathcal{P}(S)$ such that $\mathcal{S}_1 \models_{f,\alpha_1(q)(i,d_{\mathbb{T}})_2,\iota,\lambda} \psi$ and $\mathcal{S}_2 \not\models_{f,\alpha_2(q')(i,d_{\mathbb{T}})_2,\iota,\lambda'} \psi$. By definition of satisfaction, ψ can be considered as a formula that does not contain first-order formulas and atoms of the form $E = E'$ because their satisfaction does not bring into play states. Therefore, we can write equivalently that $\mathcal{S}_1 \models_{(i,d_{\mathbb{T}'}):f,\alpha_1(q)(i,d_{\mathbb{T}})_2,\iota,\lambda} \psi$ and $\mathcal{S}_2 \not\models_{(i,d_{\mathbb{T}'}):f,\alpha_2(q')(i,d_{\mathbb{T}})_2,\iota,\lambda'} \psi$, whence we conclude $\mathcal{S}_1 \models_{f,q,\iota,\lambda} [i, d]\psi$ and $\mathcal{S}_2 \not\models_{f,q',\iota,\lambda'} [i, d]\psi$ what is not possible as $q \equiv q'$. The same reasoning can be carried out for $\equiv^{-1}$. ∎

When bisimulations rest on the same system, we have further the following result:

Theorem 4.2 (Characterization): Let $\mathcal{S} = ((S, \alpha), q_0)$ be a system over $(Out \times _)^{In \times \mathbb{T}}$ with $\mathbb{T} \subseteq T$ and such that the set S is finite. Then, there exists for every $q \in S$, a set of closed formulas Γ_q such that:

$$\forall q' \in S, q \sim q' \Longleftrightarrow (\forall \varphi \in \Gamma_q, \forall f \in In^T, \mathcal{S} \models_{f,q',\emptyset} \varphi)$$

Proof: Let us associate to any state $q \in S$, the variable $\overline{x}_q \in X$, and let us define the set

$$\overline{\Gamma}_q = \{\nu \overline{x}_q.[(i, d)]\overline{x}_{q'} \wedge i \downarrow_d o | i \in In, d \in T, \alpha(q)(i, d) = (o, q')\}$$

Let us define Γ_q as the set of closed formulas obtained from $\overline{\Gamma}_q$ by recursively replacing in each formula $\nu \overline{x}_q.[(i, d)]\overline{x}_{q'} \wedge i \downarrow_d o$ the variable $\overline{x}_{q'}$ by every formula of the form $[(i', succ(d))]\overline{x}_{q''} \wedge i' \downarrow_{succ(d)} o'$ in $\overline{\Gamma}_{q'}$, and starting again this process on every free fixed point variable until every fixed point variable is within in the scope of an operator ν. S being finite, this process will terminate.

Let us suppose that $q \sim q'$. Then, for every formula $\varphi \in \Gamma_q$, we can show by induction on the number of nested occurrences of ν-formulas in φ that $\mathcal{S} \models_{f,q,\emptyset} \varphi$ for every $f \in In^T$. Let us suppose that this number is one. This means that φ is of the form $\nu \overline{x}_q.[(i, d)]\overline{x}_q \wedge i \downarrow_d o$. It is obvious that in this case $\mathcal{S} \models_{f,q,\emptyset} \varphi$. It is sufficient to choose $S' = \{q\}$. Let us suppose that the number of nested occurrences of ν-formulas is greater than one. Therefore, this means that φ is of the form $\nu \overline{x}_q.[(i, d)]\varphi' \wedge i \downarrow_d o$ where φ' is a closed formula except maybe for the variable $\overline{x}_q$. By definition, this means there exists $\overline{q} \in S$ such that $\alpha(q)(i, d) = (o, \overline{q})$, and φ' is of the form $\nu \overline{x}_{\overline{q}}.[(i, succ(d))]\varphi'' \wedge i' \downarrow_{succ(d)} o'$. By induction hypothesis, we have that $\mathcal{S} \models_{f,\overline{q},\emptyset} \varphi'$, and by hypothesis we have $\mathcal{S} \models_{f,q,\emptyset} i \downarrow_d o$. By definition, φ' is a closed formula except maybe for the variable $\overline{x}_q$. Therefore, we can write $\mathcal{S} \models_{f,\overline{q},\emptyset[\overline{x}_q/\{q\}]} [(i, d)]\varphi'$, whence we can conclude $\mathcal{S} \models_{f,q,\emptyset} \varphi$. By Theorem 4.1, since $q \sim q'$, we then have $\mathcal{S} \models_{f,q',\emptyset} \varphi$.

Conversely, let us define the binary relation $\equiv$ on S by:

$$q \equiv q' \Longleftrightarrow (\forall \varphi \in \Gamma_q, \forall f \in In^T, \mathcal{S} \models_{f,q',\emptyset} \varphi)$$

Let us show that $\equiv \subseteq \sim$. Let us suppose that $q \equiv q'$. Let $i \in In$ and $d \in T$ be an input and a date such that $\alpha(q)(i, d) = (o, \overline{q})$. By definition, for every $f \in In^T$, $\mathcal{S} \models_{f,q',\emptyset} i \downarrow_d o$. It remains to prove that $\alpha(q)(i, d_{\mathbb{T}})_{|_2} \equiv \alpha(q')(i, d_{\mathbb{T}})_{|_2}$. Let us suppose the contrary. This means there are $\psi \in \Gamma_{\alpha(q)(i,d_{\mathbb{T}})_{|_2}}$ and $f \in In^T$ such that $\mathcal{S} \models_{f,\alpha(q)(i,d_{\mathbb{T}})_{|_2},\emptyset} \psi$ and $\mathcal{S} \not\models_{f,\alpha(q')(i,d_{\mathbb{T}})_{|_2},\emptyset} \psi$. By definition, there exist a formula $\varphi \in \Gamma_q$ and a formula $\varphi' \in \overline{\Gamma}_q$ such that $\varphi[\overline{x}_q/\varphi'] = \nu \overline{x}_q.[(i, d)]\psi \wedge i \downarrow_d o$ where $\varphi[\overline{x}_q/\varphi']$ is the formula obtained from φ by substituting every occurrence of $\overline{x}_q$ by φ'. Hence, we then have that $\mathcal{S} \models_{f,q,\emptyset} \varphi[\overline{x}_q/\varphi']$ and $\mathcal{S} \not\models_{f,q,\emptyset} \varphi[\overline{x}_q/\varphi']$. As φ is a closed formula, we also have that $\mathcal{S} \models_{f,q,\emptyset} \varphi$ and $\mathcal{S} \not\models_{f,q,\emptyset} \varphi$ what is impossible since $q \equiv q'$. The same reasoning can be carried out for $\equiv^{-1}$. ∎

V. Conclusion

In this paper, we have defined a logic dedicated to express properties over systems specified in a timed extension of the coalgebraic framework of Mealy machines. The formalism thus defined then allows to consider in a same framework discrete, continuous and hybrid systems.

The logic has been defined as an extension of μ-calculus. Hence, besides standard temporal properties, it further expresses real-time properties and constraints on the production time of output values from input ones. Moreover, we have established the important property of adequacy with

respect to bisimulation as well as the characterisation of bisimulation by sets of closed formulas.

We are currently studying conditions to preserve properties along our integration and abstraction/simulation operators.

REFERENCES

[1] P. Aczel and N. Mendler. A final coalgebra theorem. In D.-H. Pitt, D.-E. Ryeheard, P. Dybjer, A.-M. Pitts, and A. Poigne, editors, *Proceedings category in computer science*, Lecture Notes in Computer Science, pages 357–365. Springer-Verlag, 1989.

[2] M. Aiguier, F. Boulanger, and B. Kanso. A formal abstract framework for modeling and testing complex software systems. *Theoretical Computer Science*, 455:66–97, 2011.

[3] M. Aiguier, B. Golden, and D. Krob. Modeling of complex systems ii: A minimalist and unified semantics for heterogeneous integrated systems. *Applied Mathematics and Computation*, 218(16), 2012.

[4] E. Alesken and R. Belcher. *Systems Engineering*. Prentice Hall, 1992.

[5] B.-S. Blanchard and W.-J. Fabrycky. *Systems engineering and analysis*. Prentice Hall, 1998.

[6] S. Bliudze and D. Krob. Modeling of complex systems - systems as data-flow machines. *Fundamenta Informaticae*, 91:1–24, 2009.

[7] J. Bradfield and C. Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *Handbook of Modal Logic*, pages 721–756. Elsevier, 2007.

[8] D. Cha, J. Rosenberg, and C. Dym. *Fundmantals of Modeling and Analysing Engineering Systems*. Cambridge University Press, 2000.

[9] F. Diener and G. Reeb. *Analyse Non Standard*. Hermann, 1989.

[10] J. Harthong. éléments pour une théorie du continu. *Astérisque*, 109/110:235–244, 1983.

[11] J. Harthong. une théorie du continu. In H. Barreau and J. Harthong, editors, *La mathématique non standard*, pages 307–329. Éditions du CNRS, 1989.

[12] R. Kashima and K. Okamoto. General models and completeness of first-order modal μ-calculus. *Journal of Logic and Computation*, 18(4):497–507, 2008.

[13] C. Delgado Kloos and P.-T. Breuer, editors. *Formal Semantics for VHDL*. Kluwer Academics Publishers, 1995.

[14] J. Lygeros. Lecture notes on hybrid systems. Technical report, ENSIETA Worshop, 2004.

[15] M.-W. Maier and E. Rechtin. *The art of system architecturing*. CRC Press, 2002.

[16] E. Nelson. Internal set theory: a new approach to nonstandard analysis. *Bulletin of the American Mathematical Society*, 83:1165–1198, 1977.

[17] A. Robinson. *Non-standard analysis*. American Elsevier, 2nd. ed. edition, 1974.

[18] J.-M.-M. Rutten. Universal coalgebra: a theory of systems. *Theor. Comput. Sci.*, 249(1):3–80, 2000.

[19] J.-M.-M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. In *International Workshop on Formal Aspects of Component Software (FACS 2005)*, volume 160 of *Electronic Notes in Computer Science*, pages 305–319. Elsevier, 2006.

[20] A.-P. Sage and J.-E. Amstrong. *Introduction to system engineering*. John Wiley, 2000.

[21] E. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, volume 6 of *Textbooks in Applied Mathematics*. Springer-Verlag, 1998.

[22] W.-C. Turner, J.-H. Mize, K.-E. Case, and J.-W. Nazemeth. *Introduction to industrial and systems engineering*. Prentice Hall, 1993.

[23] M. Viswanathan and R. Viswanathan. A higher order modal fixed point logic. In Ph. Gardner and N. Yoshida, editors, *CONCUR 2004 - Concurrency Theory*, volume 3170 of *Lecture Notes in Computer Science*, pages 512–528. Springer-Verlag, 2004.