Journal of Logical and Algebraic Methods in Programming 85 (2016) 1057–1085

Contents lists available at ScienceDirect

Journal of Logical and Algebraic Methods in Programming

www.elsevier.com/locate/jlamp

Continuity as a computational effect

Renato Neves [a,*], Luis S. Barbosa [a], Dirk Hofmann [b], Manuel A. Martins [b]

[a] INESC TEC (HASLab) & Universidade do Minho, Portugal
[b] CIDMA – Dep. of Mathematics, Universidade de Aveiro, Portugal

ARTICLE INFO

Article history:
Received 2 July 2015
Received in revised form 28 May 2016
Accepted 31 May 2016
Available online 8 June 2016

Keywords:
Monads
Components
Hybrid systems
Control theory

ABSTRACT

The original purpose of component-based development was to provide techniques to master complex software, through composition, reuse and parametrisation. However, such systems are rapidly moving towards a level in which software becomes prevalently intertwined with (continuous) physical processes. A possible way to accommodate the latter in component calculi relies on a suitable encoding of continuous behaviour as (yet another) computational effect.

This paper introduces such an encoding through a monad which, in the compositional development of hybrid systems, may play a role similar to the one played by $1+$, powerset, and distribution monads in the characterisation of partial, nondeterministic and probabilistic components, respectively. This monad and its Kleisli category provide a universe in which the effects of continuity over (different forms of) composition can be suitably studied.

1. Introduction

1.1. Motivation and objectives

Component-based software development is often explained through a visual metaphor: a palette of computational units, and a blank canvas in which they are dropped and interconnected by drawing wires abstracting different composition and synchronisation mechanisms. More and more, however, components are not limited to traditional information processing units, but encapsulate some form of interaction with physical processes. The resulting systems, referred to as *hybrid* [1,2], exhibit a complex dynamics in which computations, coordination, and physical processes interact, become mutually constrained, and cooperate to achieve specific goals.

One generic way of looking at components, proposed in [3], emphasises an *observational* semantics, through a signature of observers and methods, that makes them amenable to a *coalgebraic* characterisation as (generalisations of) abstract Mealy machines. The resulting calculus is parametric on whatever behavioural model underlies a component specification. This captures, for example, partial, nondeterministic or probabilistic behaviour of a component's dynamics by encoding such behavioural effects as *strong monads* [4] – a pervasive mathematical structure with surprising applications in different areas of Computer Science (see *e.g.*, [5–9]).

Indeed, each monad captures a specific type of behaviour, which is then reflected in the corresponding component calculus. For example, *maybe* monad $(1+)$ introduces *partial* components; the *powerset* $(\mathcal{P})$ monad *nondeterministic* ones;

* Corresponding author.
E-mail addresses: rjneves@inescporto.pt (R. Neves), lsb@di.uminho.pt (L.S. Barbosa), dirk@ua.pt (D. Hofmann), martins@ua.pt (M.A. Martins).

http://dx.doi.org/10.1016/j.jlamp.2016.05.005
2352-2208/

and *distribution* monad ($\mathcal{D}$) brings (discrete) *probabilistic* evolution into the scene. Can *continuous behaviour*, prevalent in hybrid systems and control theory, be encoded in a similar way, as (yet another) computational effect? Such is the question addressed in this paper.

Monads first came in contact to Computer Science in the 80's, when E. Moggi proposed their use to structure the denotational semantics of programming languages [10,5]. Later the concept was introduced in programming practice by P. Wadler [6], leading to a rigorous style of combining purely functional programs that mimic impure (side-)effects. The key idea is that monads encode in abstract terms several kinds of computational effects, such as exceptions, state updating, nondeterminism or continuations. Such effects are represented by a type constructor $\mathcal{T}$ (an endofunctor over a suitable category) so that computations producing values of type O are regarded as terms of type $\mathcal{T}O$. In this way *values* and *computations* are explicitly distinguished and programs can be thought of as arrows $I \to \mathcal{T}O$ representing the computation of values of type O from values of type I, while producing some effect described by $\mathcal{T}$. Or, putting it in a different way, output values are encapsulated (or embedded) in the effect specified by $\mathcal{T}$. A monad comes equipped with an identity and an associative multiplication which, from a computational point of view, builds a (trivial) computation from a value, and flattens nested effects, respectively. Furthermore, if $\mathcal{T}$ is *strong* [6] additional machinery is available to distribute the computations' effect over context. The monad structure allows program composition by handling the underlying computational effect through functor $\mathcal{T}$ and the flattening operation. Actually, each monad gives rise to a so called Kleisli category in which one may study the effects of the behavioural type (as specified by the monad) over different forms of composition; ultimately, this leads to rich component calculi (as discussed in [3]).

The current paper introduces a (strong) monad $\mathcal{H}$ that subsumes the typical continuous behaviour of dynamical, and hybrid systems. Intuitively, the type effect of $\mathcal{H}$ (*i.e.*, the underlying endofunctor) represents the (continuous) *evolution* over time of some value in O; the identity defines a trivial evolution (*i.e.*, with duration zero), and the flattening operation allows the control of an evolution to be passed along different systems.

Moreover, the paper explores the corresponding Kleisli category as the mathematical space in which the underlying (continuous) behaviour can be isolated and its effect over different forms of composition suitably studied. As we will see in the sequel, such a category gives rise to several forms of composition operators (*e.g.*, sequential, parallel execution), *wiring* mechanisms, and *synchronisation* techniques. Again this parallels the role that the categories of partial functions, relations and stochastic matrices have as reasoning universes for component composition under the behavioural model provided, respectively, by monads $1+$, $\mathcal{P}$ and $\mathcal{D}$ [11,12]. Similarly, this work paves the way to the development of a coalgebraic calculus of *hybrid components* in the spirit of [3].

1.2. A tribute to José Nuno Oliveira

The idea of regarding *continuity* as a computational effect, or more rigorously, a *physical* one, entailing a suitable notion of composition and a reasoning universe, in the form of a Kleisli category, owes much to the way José helped us to approach computational phenomena.

Building on the role of monads in functional programming and program calculi, as monadic inductive and coinductive schemes [13,14], José introduced us to monads both as a powerful structuring mechanism and a source of equally powerful genericity. An obsession for patterns and a sharp intuition for generic, conceptually reusable structures remain, after all, the hallmark of his illuminating, Socratic teaching.

In the late 1990's, José supervised the PhD work of the second author on the coalgebraic calculus of state-based components mentioned above [3]. This emerged from the conjunction of two key ideas; first, that a 'black-box' characterisation of software components favoured an *observational*, essentially coalgebraic, semantics; second, that the envisaged calculus had to be *generic*, in the sense that it should not depend on a particular notion of component behaviour. Monads, actually strong monads, were quickly identified as a source of such a genericity, the whole work boiling down to a calculus of *monadic* Mealy machines. Software components were thus studied as coalgebras (in a suitable category) typed as

$$S \longrightarrow \mathcal{T}(S \times O)^I$$

where S represents the (internal) state space, and I, O are respectively the input and output spaces. $\mathcal{T}$ is a strong monad that captures the intended behavioural effect.

Being generic entailed the need for an equally generic reasoning framework. By then, the adoption of a *pointfree*, essentially equational, calculational proof style, thus avoiding the somehow more standard coinductive proofs through the explicit construction of bisimulations, was understood as the price to be paid for genericity, as component laws were to be verified without fixing the working monad completely. Generic proofs performed in this style are clear and easy to follow, even if often long due to the systematic recording of almost all elementary steps.

For José, however, the way proofs are written is not a technicality. Proofs, as he taught us every day, are basically honest explanations, bearing evidence in a fixed formal context, and therefore must be conveyed in a crisp, clear, easily reproducible style, letting the underlying structure to emerge and helping to build the correct intuitions. Years later, in the context of a joint research project [15], José championed the use of calculational, pointfree reasoning as a way of reinvigorating the role of proof in elementary mathematical education. The pointfree style adopted in many proofs of this paper is also intended as a tribute to this view.

For José being generic does not mean to seek refuge in some sort of formal ivory tower, of stylised constructions polished ahead of any meaningful intuition. This explains why, being a devoted functional programmer, who resorts to Haskell as a pocket calculator, José soon started to focus his attention on the rich universes of specific monadic computations – their Kleisli categories. If pure functions are computations for the identity monad, relations and matrices play a similar role in such richer contexts. To be added, of course, and in a very concrete way, to the relevant calculator. His systematic, calculational, 'syntax-driven' work on relation algebra [16,17], as a framework for nondeterministic computations, and linear algebra [18,19], for probabilistic ones, was responsible for a fresh understanding of the Kleisli categories of two fundamental monads, and lead to a number of new results and simpler, elegant renderings of old ones. Having introduced a monad for continuity, this paper initiates the unravelling of the corresponding Kleisli category, as the reasoning universe for continuous processes, thus, and once again, pursuing a path José will certainly cheer.

1.3. Document structure

After a brief detour on preliminaries and notation in Section 2, the *continuous evolution* monad ($\mathcal{H}$) is introduced in Section 3. In Section 4, we explore the corresponding Kleisli category: as we will see, its arrows define *continuous systems* $I \to \mathcal{H}O$ (technically, preliminary versions of dynamical, and hybrid systems) and (Kleisli) composition makes possible for a component to execute after another, starting its evolution when the preceding one finishes its own. In Section 5, we take advantage of the so called Kleisli adjunction to define wiring mechanisms and characterise (co)limits. The latter give rise to new forms of component composition and corresponding laws. In order to add synchronisation techniques to our (monadic) framework, Section 6 provides extra structure to the underlying functor of monad $\mathcal{H}$. After this we suggest a feedback operator. In Section 7, we show that monad $\mathcal{H}$ is strong; this brings us closer to hybrid systems as coalgebraic components (in the spirit of [3]) whose behavioural effect is captured by $\mathcal{H}$. Formally, coalgebras typed as

$$S \longrightarrow \mathcal{H}(S \times O)^I.$$

Finally, Section 8 discusses related work, provides possible research directions, and presents concluding remarks.

In order to illustrate the developments of the ensuing sections, a number of classical examples of continuous and hybrid systems will be explored under the light of the framework reported in this paper.

2. Preliminaries

2.1. Continuous systems

Technically, we qualify as *continuous* a system whose output, for any given input, is a (continuous) evolution over time; *i.e.*, an arrow typed as

$$I \longrightarrow \coprod_{d \in [0,\infty]} O^{T_d}$$

where I, O are, respectively, input and output spaces, O^{T_d} the space of continuous functions $T_d \to O$ (the *evolutions*), and T_d stands for $\{r \in \mathbb{R}_{\geq 0} \mid r \leq d\}$. Actually, this definition includes the family of *continuous dynamical systems* that interpret the non-negative reals (*i.e.*, $\mathbb{R}_{\geq 0}$, here denoted by letter T) as a time domain (*cf.* [20,21]). Formally, the latter are characterised as functions,

$$\frac{\Phi : X \times \mathsf{T} \to X}{\lambda\Phi : X \to X^{\mathsf{T}}}$$

such that for any $t \in \mathsf{T}$, $x \in X$

$$\Phi\ (x, 0) = x \tag{1}$$

$$\Phi\ (x, t_1 + t_2) = \Phi(\Phi(x, t_1),\ t_2) \tag{2}$$

From a monadic perspective, continuous dynamical systems (in the form $\lambda\Phi : X \to X^{\mathsf{T}}$) may be seen as programs whose behavioural effect subsumes some form of continuous evolution over time. Indeed, as we will see later in the paper, such systems are part of a broader family of arrows that live in the Kleisli category of monad $\mathcal{H}$ ($\mathbf{Top}_{\mathcal{H}}$). In general, law (1) will be an important part in the characterisation of Kleisli composition. We will also see that the traditional view of hybrid systems – as a family of dynamical (or continuous) systems indexed by a (discrete) state space – coincides with ours; and, moreover, that such systems also live in $\mathbf{Top}_{\mathcal{H}}$ (due to the machinery that makes $\mathcal{H}$ strong).

$$\frac{f : X \to Y,\ g : Y \to Z}{g \cdot f : X \to Z}\ (\cdot) \qquad \frac{f : X \times Y \to Z}{\lambda f : X \to Z^Y}\ (\lambda)$$

$$\frac{f : X \to Y_1,\ g : X \to Y_2}{\langle f, g\rangle : X \to Y_1 \times Y_2}\ (\times) \qquad \frac{f : X_1 \to Y,\ g : X_2 \to Y}{[f, g] : X_1 + X_2 \to Y}\ (+)$$

$$\frac{f : X \to Y,\ A \subseteq X}{f_A : A \to Y}\ (\downarrow_l) \quad \text{with } f_A = f \cdot \iota \text{ (for } \iota : A \hookrightarrow X) \qquad \frac{f : X \to Y,\ \mathrm{img}\ f \subseteq B}{f^B : X \to B}\ (\downarrow_r) \quad \text{with } \iota \cdot f^B = f \text{ (for } \iota : B \hookrightarrow Y)$$

Fig. 1. Continuity rules in **Top**.

2.2. Notation

The key role that continuity takes in this work, suggests the category **Top** of topological spaces and continuous functions as a suitable working environment for developing the envisaged results.

In the sequel, whenever the context is clear, a topological space will be denoted by its underlying set. Topological spaces $X \times Y$, $X + Y$ correspond to the canonical product and coproduct of X, Y, respectively. Also, for any $X \subseteq Y$, assume that X has the subspace topology induced by Y. Finally, whenever Y is *core-compact* (*cf.* [22]), space X^Y has the exponential topology.

Category **Top** is (co)complete; this allows to take advantage of isomorphisms $\alpha : (X \times Y) \times Z \cong X \times (Y \times Z)$, and $sw : X \times Y \cong Y \times X$. **Top** also provides a set of useful rules for showing continuity; Fig. 1 sums up the ones used in the paper. In rule (λ), Y must be core-compact so that the evaluation function $ev : X^Y \times Y \to X$ is well defined (*cf.* [22]).

Universal arrows $X \to 1$ to the final object in **Top** are denoted by !, and a function constantly yielding a value x by $\underline{x}$. Given two functions $f, g : X \to Y$, and a predicate p, we introduce a conditional expression $f \lhd p \rhd g : X \to Y$, defined by,

$$(f \lhd p \rhd g)\, x \mathrel{\widehat{=}} \begin{cases} f\, x & \text{if } p\, x \\ g\, x & \text{otherwise} \end{cases}$$

Whenever found relevant, and no ambiguities arise, we will denote expression $(f \lhd p \rhd g)\, x$ by $(f\, x \lhd p\, x \rhd g\, x)$. The continuous functions *minimum* $\curlywedge : \mathsf{T} \times [0,\infty] \to \mathsf{T}$ and *truncated subtraction* $\ominus : \mathsf{T} \times [0,\infty] \to \mathsf{T}$ play a key role in some proofs. They are defined by the following equations

$$\curlywedge \mathrel{\widehat{=}} \pi_1 \lhd (\leq) \rhd \pi_2$$

$$\ominus \mathrel{\widehat{=}} (-) \lhd (>) \rhd \underline{0}$$

where $\leq, >$ are the usual ordering relations over the reals with infinity.

As usual, functions $\pi_1 : X \times Y \to X$, $\pi_2 : X \times Y \to Y$ correspond to the projections associated with any binary product, and $i_1 : X \to X + Y$, $i_2 : Y \to X + Y$ the coprojections associated with any binary coproduct. Moreover, symbol $\star$ is used to denote the element of a singleton set, and $|\mathbf{C}|$ to represent the class of objects of a category $\mathbf{C}$. Finally, to avoid a burdened notation, we will often drop the subscript in a component of a natural transformation.

3. The continuous evolution monad

As mentioned above, we regard continuous systems as arrows of type

$$I \longrightarrow \coprod_{d \in [0,\infty]} O^{T_d}.$$

In order to define them in **Top**, we need to equip the target object with a suitable topology. A first choice would be the coproduct topology (as suggested by the expression above), but this is not suitable, since in many cases such a topology forbids the system to change the duration of its evolutions along different inputs.

Let us thus explore an alternative topology; the strategy will be similar to the one used in the definition of a *Moore path category* where, given a topological space X, arrows are paths (*i.e.*, evolutions) $[0, d] \to X$ and composition corresponds to the concatenation of those paths (*cf.* [23]). Actually, the flattening operation of monad $\mathcal{H}$, discussed below, can be seen as *a more general version* of path concatenation.

Consider, with no loss of generality, that all evolutions have domain T. Such is possible when one notices that T_d (for some $d \in [0,\infty]$) is a *retract* of T through the *truncation* function (the retraction)

$$\curlywedge_d : \mathsf{T} \longrightarrow T_d$$

$\curlywedge_d \mathrel{\widehat{=}} id \lhd (\leq_d) \rhd \underline{d}$ and considers just those functions $f \in O^{\mathsf{T}}$ that become constant after time instant d, *i.e.* $f \cdot \curlywedge_d = f$. This gives a family of bijections $\{f \in O^{\mathsf{T}} \mid f \cdot \curlywedge_d = f\} \cong O^{T_d}$ indexed by durations $d \in [0, \infty]$. Continuous systems thus become arrows typed as,

$$I \longrightarrow \{\, (f, d) \in O^{\mathsf{T}} \times [0, \infty] \mid f \cdot \curlywedge_d = f \,\}$$

where the target object comes equipped with the canonical topology. This leads to the following definition for the underlying functor of monad $\mathcal{H}$.

Definition 1. $\mathcal{H} : \mathbf{Top} \to \mathbf{Top}$ is a mapping such that for any objects $X, Y \in |\mathbf{Top}|$ and any continuous function $g : X \to Y$,

$$\mathcal{H}X \mathrel{\widehat{=}} \{\, (f, d) \in X^{\mathsf{T}} \times D \mid f \cdot \curlywedge_d = f \,\}$$
$$\mathcal{H}g \mathrel{\widehat{=}} g^{\mathsf{T}} \times id$$

where $\mathsf{D} = [0, \infty]$ is the one-point compactification of $\mathbb{R}_{\geq 0}$ (*cf.* [24]), and $g^{\mathsf{T}}\, f = g \cdot f$.

Theorem 1. *$\mathcal{H}$ is a functor.*

Proof. We need to show that for any continuous functions $g : X \to Y, h : Y \to Z$, $\mathcal{H}g : \mathcal{H}X \to \mathcal{H}Y$ is continuous, and $\mathcal{H}(h \cdot g) = \mathcal{H}h \cdot \mathcal{H}g$

Since $\mathcal{H}g = g^{\mathsf{T}} \times id$, and g is continuous, then $\mathcal{H}g$ must be as well. Distributivity of composition follows from property

$$\iota \cdot \mathcal{H}g = \big((\_ \times \mathsf{D}) \cdot (\_)^{\mathsf{T}}\, g\big) \cdot \iota$$

where ι is the inclusion map $\mathcal{H}X \hookrightarrow (X^{\mathsf{T}} \times \mathsf{D})$, $(\_ \times \mathsf{D})$ is the (D) product functor, and $(\_)^{\mathsf{T}}$ the (T) exponential functor. □

Let us explore some examples of continuous systems characterised as arrows $I \to \mathcal{H}O$.

Example 1. *Signal generators* are classical examples of continuous systems that can generate sinusoidal waves as output. They can be regarded as arrows $s : \mathbb{R} \to \mathcal{H}\mathbb{R}$ such that $s\, r \mathrel{\widehat{=}} (r + (sin\ \_),\ \infty)$.

Note that, in contrast to the coproduct topology (in the target object), the topology chosen for $\mathcal{H}$ allows durations to change, and thus captures a wider range of behaviours. For example,

Example 2. Consider a *thermostat* $c : \mathbb{R} \to \mathcal{H}\mathbb{R}$ that, given the current temperature, *linearly* raises it to, say, 20 °C. Such a behaviour can be expressed as $c\, r \mathrel{\widehat{=}} ((r + \_),\ 20 \ominus r)$ where $\ominus : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is the truncated subtraction, *i.e.*, $\ominus = (-) \lhd (>) \rhd \underline{0}$.

The execution time of system c is thus inversely proportional to the current temperature (which is given as input).

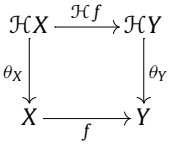
One may also consider another component that takes action after c, and whose functionality is, for instance, to maintain the current temperature. The result is a composed system that can raise temperatures to a desired level and then maintain them – we will explore this specific case in the next section. Of course, analogous behaviour can also be found in *e.g.*, cruise control systems, water level regulators, and production lines. For example, imagine a component of a cruise control system that gives control of the car's velocity to another component whenever an obstacle is detected, or the emergency mode becomes active. As we will see in the sequel, Kleisli composition (for monad $\mathcal{H}$) caters for this sort of action.

The following definition will help in the development of monad $\mathcal{H}$.

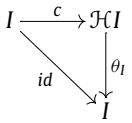
Definition 2. For any given topological space $X \in |\mathbf{Top}|$, define continuous function $\theta_X : \mathcal{H}X \to X$ such that

$$\theta_X\, (f, d) \mathrel{\widehat{=}} f\ 0.$$

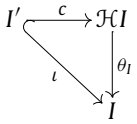
Actually, we can canonically extend $\theta_X : \mathcal{H}X \to X$ to a natural transformation $\theta : \mathcal{H} \to Id$, since it is straightforward to show that the following diagram commutes for any continuous function $f : X \to Y$.



Moreover, it becomes possible to express the first law of continuous dynamical systems (recall the previous section) in a concise, diagramatic manner: simply by saying that system $c : I \to \mathcal{H}I$ obeys the first law ((1) above) iff the diagram below commutes.



Actually, we can generalise the diagram to



where $\iota : I' \hookrightarrow I$ is the inclusion map $I' \subseteq I$. We qualify as *pre-dynamical* any system that follows this generalised condition. Note that both examples above (1 and 2) concern pre-dynamical systems.

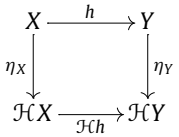
We shall now discuss how to equip $\mathcal{H}$ with the structure of a monad. As already mentioned, in programming semantics a monad captures a behavioural effect and provides mechanisms to wrap a value into such an effect and to flatten two effects into a single one. Technically, they are referred to as the monad identity $\eta : Id \to \mathcal{H}$, and its multiplication $\mu : \mathcal{H}\mathcal{H} \to \mathcal{H}$, respectively. Let us start by defining the unit operation $\eta : Id \to \mathcal{H}$, which will denote trivial evolutions.

Definition 3. Given a space $X \in |\mathbf{Top}|$, function $\eta_X : X \to \mathcal{H}X$ is defined by

$$\eta_X\, x \mathrel{\widehat{=}} (\underline{x},\, 0).$$

Intuitively, arrow $\eta_X : X \to \mathcal{H}X$ defines a system whose outputs are always trivial evolutions, *i.e.*, with duration zero. For this reason we will refer to η_X as $copy_X$, and often omit the subscript.

Lemma 1. *The mapping $\eta : Id \to \mathcal{H}$ is a natural transformation,* i.e.*, for any topological space X, $\eta_X : X \to \mathcal{H}X$ is a continuous function, and, moreover, the diagram below commutes*

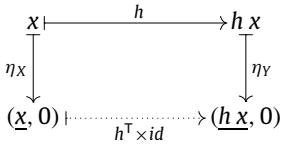


for any continuous function $h : X \to Y$.

Proof. To see that η_X is continuous, observe first that $\eta_X = \langle \lambda\pi_1, \underline{0} \rangle$. Then,

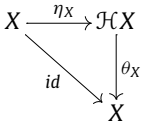
$$\dfrac{\dfrac{\dfrac{\pi_1 : X \times \mathsf{T} \to X}{\lambda\pi_1 : X \to X^{\mathsf{T}}}\ (\lambda)}{\langle \lambda\pi_1, \underline{0} \rangle : X \to X^{\mathsf{T}} \times \mathsf{D}}\ (\times)}{\langle \lambda\pi_1, \underline{0} \rangle : X \to \mathcal{H}X}\ (\downarrow_r)$$

It remains to show the naturality of $\eta : Id \to \mathcal{H}$. Consider the diagram



where $h : X \to Y$ is an arbitrary continuous function. Property $h \cdot \underline{x} = \underline{h\, x}$ entails its commutativity. □

It is also simple to see that, for any topological space $X \in |\mathbf{Top}|$, the following diagram commutes



(*i.e.*, that η_X is pre-dynamical). Actually, this is one of two laws that characterise θ_X as an *Eilenberg-Moore* $\mathcal{H}$-algebra [25], a notion we will visit later in the paper.

The next step is to define multiplication $\mu : \mathcal{H}\mathcal{H} \to \mathcal{H}$. We start with an (auxiliary) definition of evolution (or path) concatenation.

Definition 4. Given any elements $(f, d), (g, e) \in \mathcal{H}X$, define

$$(f, d) +\!\!+ (g, e) \mathrel{\widehat{=}} (f +\!\!+_d g, d + e)$$

where $f +\!\!+_d g \mathrel{\widehat{=}} f \lhd (\leq_d) \rhd g\,(\_ - d)$.

Let us omit the subscript in $+\!\!+_d$. Note that $f +\!\!+ g$ is continuous whenever the endpoint of f and the startpoint of g coincide. We will show that this condition is always met for the case of multiplication.

Definition 5. Given any topological space $X \in |\mathbf{Top}|$, define

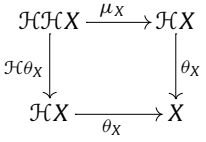
$$\mu_X\,(f, d) \mathrel{\widehat{=}} \begin{cases} (\theta \cdot f, d) +\!\!+ (f\ d) & \text{if } d \neq \infty \\ (\theta \cdot f,\ \infty) & \text{otherwise} \end{cases}$$

Intuitively, multiplication will serve to concatenate the resulting evolutions of two components.

Lemma 2. *The family of mappings μ defines a natural transformation.*

Proof. In Appendix. □

Lemma 3. *For every topological space $X \in |\mathbf{Top}|$, the diagram below commutes*



Proof. Consider a pair $(f, d) \in \mathcal{H}\mathcal{H}X$, where d is finite. Then,

$$\begin{aligned}
& \theta \cdot \mu\,(f, d) \\
= \quad & \{\ \text{Definition of } \mu\ \} \\
& \theta\,((\theta \cdot f, d) +\!\!+ (f\ d)) \\
= \quad & \{\ \text{Definition of } +\!\!+ \text{ on point 0}\ \} \\
& \theta\,((\theta \cdot f, d)) \\
= \quad & \{\ \text{Definition of } \mathcal{H}\ \} \\
& \theta \cdot \mathcal{H}\theta\,(f, d)
\end{aligned}$$

Proof for the case in which d is infinite is achieved via an analogous reasoning process. □

This property, together with the fact that $\theta_X \cdot \eta_X = id$ (discussed above), entail that $\theta_X : \mathcal{H}X \to X$ is an Eilenberg-Moore $\mathcal{H}$-algebra. In words, an algebra of functor $\mathcal{H}$ that is compatible with the monadic structure defined above. This notion will be rather useful in the sequel.

Theorem 2. *$\langle \mathcal{H}, \eta, \mu \rangle$ forms a monad.*

Proof. In Appendix. □

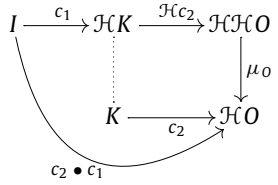
4. ...and its Kleisli category ($\mathbf{Top}_{\mathcal{H}}$)

If a monad abstracts a computational effect, its Kleisli category, represents the universe of computations encapsulated in such an effect. Hence, in the case of monad $\mathcal{H}$, the associated Kleisli category of $\mathcal{H}$ ($\mathbf{Top}_{\mathcal{H}}$) provides an interesting setting to study the requirements placed by continuity over different forms of composition. Actually, the envisaged calculus of continuous, and hybrid components is essentially its calculus.

This section studies the Kleisli composition of $\mathbf{Top}_{\mathcal{H}}$, and illustrates its application to the specification of continuous systems – the hybrid ones will be discussed later in the paper. We start with the definition of $\mathbf{Top}_{\mathcal{H}}$.

Definition 6. Category $\mathbf{Top}_{\mathcal{H}}$ is defined as follows:

- $|\mathbf{Top}_{\mathcal{H}}| = |\mathbf{Top}|$,
- for any objects $I, O \in |\mathbf{Top}_{\mathcal{H}}|$, $\mathbf{Top}_{\mathcal{H}}(I, O) = \mathbf{Top}(I, \mathcal{H}O)$, and for any object $I \in |\mathbf{Top}_{\mathcal{H}}|$, η_I is its identity.
- Given two arrows $c_1 : I \to \mathcal{H}K$, $c_2 : K \to \mathcal{H}O$ their composition, denoted by $c_2 \bullet c_1$, is given by $\mu_O \cdot \mathcal{H}c_2 \cdot c_1$. Diagrammatically,



Whenever found suitable, we will denote an arrow $c : I \to \mathcal{H}O$ as $c : I \nrightarrow O$, and $\pi_1 \cdot c$ as $f_c : I \to O^{\mathsf{T}}$.

Recall that arrows $c : I \nrightarrow O$ are here interpreted as continuous components, which means that the Kleisli composition of $\mathcal{H}$ can be seen as a component operator. Let us explore its behaviour: consider two systems

$$c_1 : I \nrightarrow K, \quad c_2 : K \nrightarrow O.$$

For a given input $x \in I$, compute the execution time of $c_2 \bullet c_1$,

$$\begin{aligned}
& \pi_2 \cdot (c_2 \bullet c_1)\ (x) \\
= \quad & \{\ \text{Kleisli composition}\ \} \\
& \pi_2 \cdot \mu \cdot \mathcal{H}c_2 \cdot c_1\ (x) \\
= \quad & \{\ \text{Definition of } \mathcal{H}, \text{ let } d = \pi_2 \cdot c_1\ (x)\ \} \\
& \pi_2 \cdot \mu (c_2 \cdot (f_{c_1}\ x), d) \\
= \quad & \{\ \text{Definition of } \mu\ \} \\
& d + \pi_2 (c_2 \cdot (f_{c_1}\ x)\ d) \\
= \quad & \{\ \text{Composition}\ \} \\
& d + \pi_2 (c_2\ (f_{c_1}\ x\, d))
\end{aligned}$$

This means that the execution time of $c_2 \bullet c_1$ is the sum of the execution times of c_1 (for input x) and c_2 (which receives value $f_{c_1}\, x\, d$ as input). On the other hand,

$$\begin{aligned}
& \pi_1 \cdot (c_2 \bullet c_1)\ (x) \\
= \quad & \{\ \text{Kleisli composition}\ \} \\
& \pi_1 \cdot \mu \cdot \mathcal{H}c_2 \cdot c_1\ (x) \\
= \quad & \{\ \text{Definition of } \mathcal{H}, \text{ let } d = \pi_2 \cdot c_1\ (x)\ \} \\
& \pi_1 \cdot \mu\ (c_2 \cdot (f_{c_1}\ x), d) \\
= \quad & \{\ \text{Definition of } \mu\ \} \\
& \theta \cdot c_2 \cdot (f_{c_1}\ x) \mathbin{+\!\!+} (f_{c_2}\ (f_{c_1}\ x\, d)) \\
= \quad & \{\ \text{Definition of } +\!\!+\ \} \\
& \theta \cdot c_2\ (f_{c_1}\ x\ _) \lhd (\leq_d) \rhd f_{c_2}\ (f_{c_1}\ x\, d)\ (_ - d)
\end{aligned}$$

Hence, if c_2 is pre-dynamical,

$$f_{(c_2 \bullet c_1)}\, x = (f_{c_1}\, x\ \_) \lhd (\leq_d) \rhd f_{c_2}\, (f_{c_1}\, x\, d)\, (\_ - d)$$

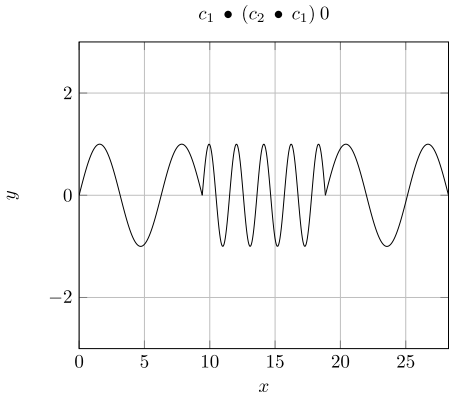
The last expression tells that for the duration of $c_1\, x$, $c_2 \bullet c_1\, x$ evolves first according to c_1, and then, on its termination, according to c_2 which receives as input the endpoint of $f_{c_1}\, x$. Clearly, this is the expected behaviour according to the definition of operation μ, which 'concatenates' evolutions. Intuitively, $c_2 \bullet c_1$ may also be described as mentioned in Section 1: component c_1 acts and then, at instant d, gives control of its evolution to c_2.

If, however, c_2 is not pre-dynamical, then up to completion of interval $[0, d]$, c_2 *'alters'* the evolution of c_1; then it proceeds according to its own evolution. These notions are illustrated in the following examples.

Example 3. Given two signal generators $c_1, c_2 : \mathbb{R} \nrightarrow \mathbb{R}$ defined as

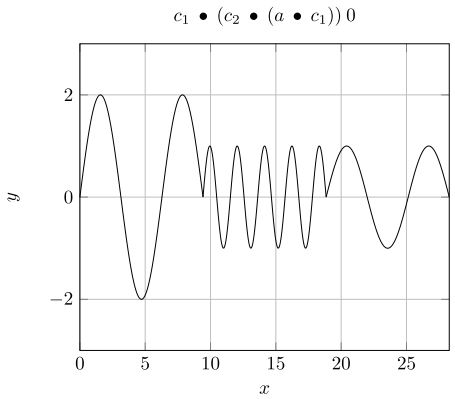
$$c_1\, r \mathrel{\widehat{=}} (r + (sin\ \_), 3\pi), \quad c_2\, r \mathrel{\widehat{=}} (r + sin\, (3 \times \_), 3\pi)$$

the evolution $c_1 \bullet (c_2 \bullet c_1)\, 0$ is represented by the plot below.



This type of signal is commonly seen in *frequency modulation*: the varying frequency is used to encode information for electromagnetic transmission. Note that c_1 gives control for some time to c_2, and then 'takes it back'.

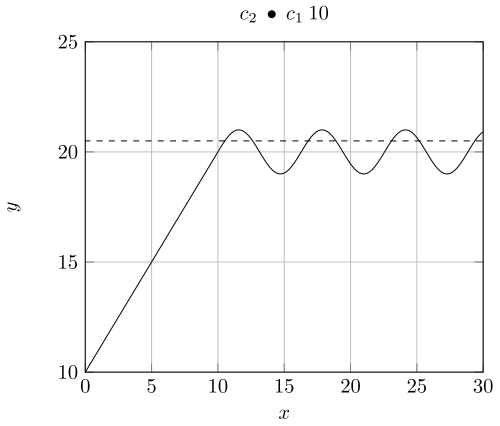
In order to amplify signals, one can use component $a : \mathbb{R} \nrightarrow \mathbb{R}$, where $a\, r \mathrel{\widehat{=}} (\underline{r \times 2}, 0)$ (note that since system a is not pre-dynamical it can alter evolutions of other components). Given input 0, system $c_1 \bullet (c_2 \bullet (a \bullet c_1))$, returns the following evolution.



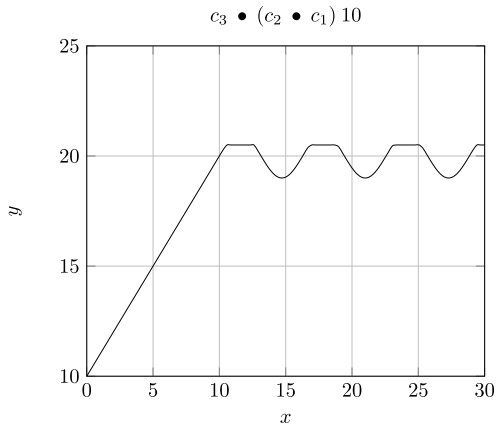
Example 4. Suppose the temperature of a room is to be regulated according to the following discipline: starting at 10 °C, seek to reach and maintain 20 °C, but in no case surpass 20.5 °C. To realise such a system, three elementary components have to work together: c_1 to raise the temperature to 20 °C, component c_2 to maintain a given temperature, and component c_3 to ensure the temperature never goes over 20.5 °C. Formally,

$$c_1\, x = (\,(x + \_)\,,\; 20 \ominus x\,)$$
$$c_2\, x = (\,x + (\sin \_)\,,\; \infty\,)$$
$$c_3\, x = (\,\underline{x} \lhd (x \leq 20.5) \rhd \underline{20.5}\,,\, 0\,)$$

In a first try one may compose c_2, c_1 into $c_2 \bullet c_1$. This results in a component able to read the current temperature, raise it to 20 °C, and then keep it stable, as exemplified by the plot below.



If, however, temperatures over 20.5 °C occur, composition $c_3 \bullet (c_2 \bullet c_1)$ puts the system back into the right track as illustrated in the following plot.



Clearly, c_3 can be regarded as a supervisor system that, for the sake of efficiency, only acts when temperatures exceed the threshold, using just enough power to keep the temperate below the limit. Actually, note that c_3 is able to play a supervisory role precisely because it is non pre-dynamical. Of course in this specific case, we assume that c_3 has an idealised behaviour, which, despite pedagogical, is quite unrealistic.

The examples above hint at an interesting property of evolutions with infinite duration.

Theorem 3. *Consider two arrows $c_1 : I \nrightarrow O$, $c_2 : O \nrightarrow O$. If system c_2 is pre-dynamical and* $\text{img}\,(\pi_2 \cdot c_1 \cdot \iota) \subseteq \{\infty\}$ *for some embedding $\iota : I' \hookrightarrow I$, then*

$$(c_2 \bullet c_1) \cdot \iota = c_1 \cdot \iota$$

Proof.

$$\begin{aligned} & (c_2 \bullet c_1) \cdot \iota \\ = \quad & \{\ \text{Kleisli composition, } \text{img}\,(\pi_2 \cdot c_1 \cdot \iota) \subseteq \{\infty\}\ \} \end{aligned}$$

$$
\begin{array}{ll}
& \mu\,(c_2 \cdot (f_{c_1} \cdot \iota), \infty) \\
= & \quad \{ \text{ Definition of } \mu \ \} \\
& (\theta \cdot c_2 \cdot (f_{c_1} \cdot \iota), \infty) \\
= & \quad \{ \text{ System } c_2 \text{ is pre-dynamical } \} \\
& (f_{c_1} \cdot \iota, \infty) \\
= & \quad \{ \text{ Notation } \} \\
& c_1 \cdot \iota \qquad \square
\end{array}
$$

Corollary 1. *If c_2 is pre-dynamical and* $\mathrm{img}\,(\pi_2 \cdot c_1) \subseteq \{\infty\}$*, then* $c_2 \bullet c_1 = c_1$.

This means that if evolutions of the first component always exhibit an infinite duration, the second one, if pre-dynamical, will never have the chance to execute.

In general, $\mathcal{H}$-Kleisli composition provides the basic composition mechanism for continuous components; the structure of $\mathbf{Top}_{\mathcal{H}}$ yields its basic laws. To be more concrete, take *copy* as the trivial system that outputs its input with duration zero (*i.e.*, the unit of monad $\mathcal{H}$). Then, given systems c_1, c_2, c_3

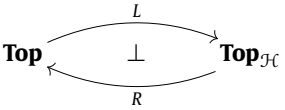
$$copy \bullet c_1 = c_1 \tag{3}$$

$$c_1 \bullet copy = c_1 \tag{4}$$

$$(c_3 \bullet c_2) \bullet c_1 = c_3 \bullet (c_2 \bullet c_1) \tag{5}$$

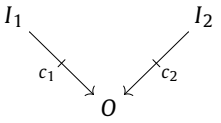
5. Wiring mechanisms and (additional) composition operators

In a category, (co)limits are a main tool to 'build new arrows from old ones', which in the case of $\mathbf{Top}_{\mathcal{H}}$ translates to new forms of component composition. Actually, coproducts are easy to obtain through the canonical adjunction between **Top** and $\mathbf{Top}_{\mathcal{H}}$,

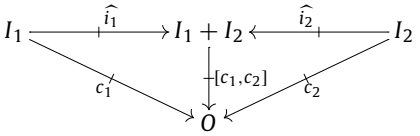


which entails that $\mathbf{Top}_{\mathcal{H}}$ inherits colimits of **Top** through L. For notational simplicity, given a continuous function $f : X \to Y$, we will denote system $Lf = \eta \cdot f : X \nrightarrow Y$ by $\widehat{f}$.

In $\mathbf{Top}_{\mathcal{H}}$, the coproduct (also known as a *choice* operator) is inherited as follows: given two components



define component $[c_1, c_2] : I_1 + I_2 \nrightarrow O$ which makes the following diagram to commute.



Intuitively, $[c_1, c_2]$ behaves as c_1 whenever input I_1 is chosen, and as c_2 otherwise. Such a mechanism is useful to aggregate systems with the same codomain; the result being a *singular* system with different modes of operation (corresponding to the respective subcomponents), chosen according to the input received. As usual, a functorial *sum* operator is easily defined.

Definition 7. Consider components $c_1 : I_1 \nrightarrow O_1$, $c_2 : I_2 \nrightarrow O_2$. Then define component $c_1 \boxplus c_2 : I_1 + I_2 \nrightarrow O_1 + O_2$ as

$$c_1 \boxplus c_2 \mathrel{\widehat{=}} [\widehat{i_1} \bullet c_1, \widehat{i_2} \bullet c_2]$$

The definition of operator choice as the coproduct universal arrow in $\mathbf{Top}_{\mathcal{H}}$, yields a number of useful laws for free.

$$c_3 \bullet [c_1, c_2] = [c_3 \bullet c_1, c_3 \bullet c_2] \tag{6}$$

$$(c_1 \boxplus c_2) \bullet \widehat{i_1} = \widehat{i_1} \bullet c_1 \tag{7}$$

$$(c_1 \boxplus c_2) \bullet \widehat{i_2} = \widehat{i_2} \bullet c_2 \tag{8}$$

$$copy_X \boxplus copy_Y = copy_{X+Y} \tag{9}$$

$$(d_1 \boxplus d_2) \bullet (c_1 \boxplus c_2) = (d_1 \bullet c_1) \boxplus (d_2 \bullet c_2) \tag{10}$$

$$[d_1, d_2] \bullet (c_1 \boxplus c_2) = [d_1 \bullet c_1, d_2 \bullet c_2] \tag{11}$$

Moreover,

Lemma 4. *For any continuous functions $f : X_1 \to Y_1$, $g : X_2 \to Y_2$, the following equation holds*

$$\widehat{f} \boxplus \widehat{g} = \widehat{f + g} \tag{12}$$

Proof.

$$\begin{aligned}
&\widehat{f} \boxplus \widehat{g} \\
=\quad &\{\ \text{Definition of } \boxplus\ \} \\
&[\widehat{i_1} \bullet \widehat{f}, \widehat{i_2} \bullet \widehat{g}] \\
=\quad &\{\ L \text{ is a functor}\ \} \\
&[\widehat{i_1 \cdot f}, \widehat{i_2 \cdot g}] \\
=\quad &\{\ \text{Definition of } L\ \} \\
&[copy \cdot i_1 \cdot f, copy \cdot i_2 \cdot g] \\
=\quad &\{\ \text{Universal property of coproduct}\ \} \\
&copy \cdot [i_1 \cdot f, i_2 \cdot g] \\
=\quad &\{\ \text{Definition of } +, \text{ definition of } L\ \} \\
&\widehat{f + g} \qquad \square
\end{aligned}$$

The left adjoint is also useful to lift functions to the universe of $\mathbf{Top}_{\mathcal{H}}$. This provides a number of interesting operations and wiring mechanisms. For example, recall the diagonal function $\triangle : X \to X \times X$ which duplicates the input value; the corresponding lifted version $\widehat{\triangle} : X \nrightarrow X \times X$ duplicates evolutions. Take now the scalar multiplication $*_s : \mathbb{R} \to \mathbb{R}$; operation $\widehat{*_s} : \mathbb{R} \nrightarrow \mathbb{R}$ can be used to amplify signals, a ubiquitous procedure both in signal and control theory. Another example is $\widehat{\pi_1} : X \times Y \nrightarrow X$ (resp. $\widehat{\pi_2} : X \times Y \nrightarrow X$) which eliminates the right (resp. left) side of 'paired' evolutions. Finally, $\widehat{sw} : X \times Y \nrightarrow Y \times X$ swaps the order of evolutions, a functionality graphically represented by wire swapping.

Since L is a functor, the following laws also come for free

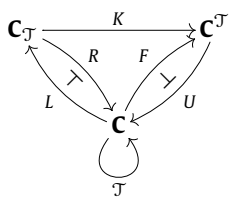
$$\widehat{id} = copy \tag{13}$$

$$\widehat{g} \bullet \widehat{f} = \widehat{g \cdot f} \tag{14}$$

Finding limits in a Kleisli category through left adjoint L is often more difficult. However, under specific conditions, L also preserves limits. The following theorem makes such conditions precise.

Theorem 4. *Consider the Kleisli adjunction $L \dashv R$ of a given monad $\langle \mathcal{T}, \eta, \mu \rangle$. Functor L preserves whatever limits $\mathcal{T}$ does.*

Proof. Observe the diagram



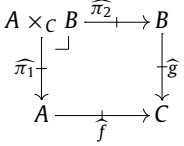
where $\mathbf{C}^{\mathcal{T}}$ is the Eilenberg-Moore category for monad $\mathcal{T}$ [25], and K the corresponding (*fully faithful*) functor such that $\mathcal{T} = UKL$. Then, consider a limit $\lim_{\leftarrow} D$ in $\mathbf{C}$ and assume that $\mathcal{T}$ preserves it. This means that $\mathcal{T}(\lim_{\leftarrow} D)$ is the limit of $\mathcal{T}D$, and equivalently, $UKL(\lim_{\leftarrow} D)$ is the limit of $UKLD$. Since both U and K reflect limits, $L(\lim_{\leftarrow} D)$ must be the limit of LD. □

Note that the theorem above was stated in general terms and is thus applicable to any monad. Even though easily proved, its consequences are quite useful. For example, in the case of $\mathcal{H}$ it provides pullbacks in $\mathbf{Top}_{\mathcal{H}}$, as

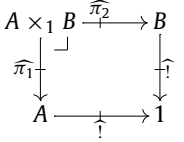
Theorem 5. *Functor $\mathcal{H}$ preserves pullbacks.*

Proof. In the appendix. □

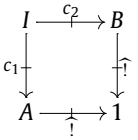
More concretely, Theorems 4 and 5 assert that any cospan $A \xrightarrow{f} C \xleftarrow{g} B$ in **Top** gives rise to a pullback in $\mathbf{Top}_{\mathcal{H}}$, diagrammatically described as



One interesting cospan, worthy of special attention, is $A \xrightarrow{!} 1 \xleftarrow{!} B$, which induces the pullback



Indeed, such a construction brings parallelism up front, and moreover, makes possible to combine evolutions. More concretely, the diagram states that whenever two systems are *compatible* – in the sense that for any input they produce evolutions with equal duration – a new component that encapsulates their parallel composition can be defined. Formally, two systems $c_1 : I \nrightarrow A$, $c_2 : I \nrightarrow B$ are called compatible when the diagram



commutes (note that this is not trivially true, because 1 is not a final object in $\mathbf{Top}_{\mathcal{H}}$). Then, let E denote set $\{((f,d),(g,e)) \in \mathcal{H}A \times \mathcal{H}B \mid d = e\}$. When the two systems are compatible, a new component $\langle\langle c_1, c_2\rangle\rangle : I \nrightarrow (A \times_1 B)$ comes forward through the mediating arrow (of the pullback), as follows

$$\langle\langle c_1, c_2\rangle\rangle \mathrel{\widehat{=}} \gamma \cdot \langle c_1, c_2\rangle$$

where $I \xrightarrow{\langle c_1, c_2\rangle} E \xrightarrow{\gamma} \mathcal{H}(A \times_1 B)$, $\gamma\,((f,d),(g,d)) \mathrel{\widehat{=}} (\langle f, g\rangle, d)$.

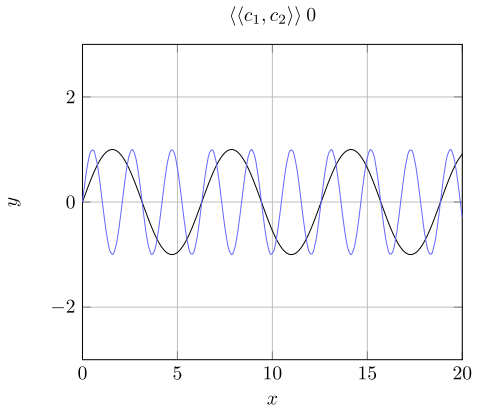
Note that $\mathrm{img}\,\langle c_1, c_2\rangle \subseteq E$ precisely because of the assumption of compatibility between components (*cf.* proof of Theorem 5). In order to keep notation simple, we will omit the 1 in the subscript of $(A \times_1 B)$.

We call $\langle\langle c_1, c_2\rangle\rangle$ the *strict parallel* composition of c_1 and c_2. Let us illustrate its behaviour through a number of examples.

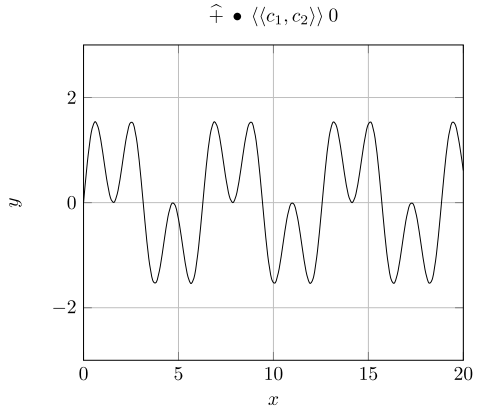
Example 5. Consider two signal generators,

$$c_1\,x = (\,x + (\sin\_),\ 20\,),\ \ c_2\,x = (\,x + \sin(3 \times \_),\ 20\,)$$

For input 0, system $\langle\langle c_1, c_2\rangle\rangle$ exhibits the following behaviour



Consider now component $\widehat{+} : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ which adds incoming signals. Then, for input 0, the composed system $\widehat{+} \bullet \langle\langle c_1, c_2\rangle\rangle$ yields the following signal.



Since strict parallelism comes from a pullback, the following operator arises in a canonical way.

Definition 8. Consider two continuous systems $c_1 : I_1 \nrightarrow O_1$, $c_2 : I_2 \nrightarrow O_2$ such that $c_1 \bullet \widehat{\pi_1}$ and $c_2 \bullet \widehat{\pi_2}$ are compatible. Then, define $c_1 \boxtimes c_2 : I_1 \times I_2 \nrightarrow O_1 \times O_2$ as

$$c_1 \boxtimes c_2 \mathrel{\widehat{=}} \langle\langle c_1 \bullet \widehat{\pi_1}, c_2 \bullet \widehat{\pi_2}\rangle\rangle$$

Moreover, the following laws come for free, further contributing to an emerging calculus of continuous and hybrid components: in each equation below, assume that both its sides are well defined (*i.e.* that the compatibility conditions are respected). Then, we have,

$$\langle\langle c_1, c_2\rangle\rangle \bullet d = \langle\langle c_1 \bullet d, c_2 \bullet d\rangle\rangle \tag{15}$$

$$\widehat{\pi_1} \bullet (c_1 \boxtimes c_2) = c_1 \bullet \widehat{\pi_1} \tag{16}$$

$$\widehat{\pi_2} \bullet (c_1 \boxtimes c_2) = c_2 \bullet \widehat{\pi_2} \tag{17}$$

$$\langle\langle c_1, c_2\rangle\rangle = (c_1 \boxtimes c_2) \bullet \widehat{\Delta} \tag{18}$$

$$copy_X \boxtimes copy_Y = copy_{X \times Y} \tag{19}$$

$$(d_1 \boxtimes d_2) \bullet (c_1 \boxtimes c_2) = (d_1 \bullet c_1) \boxtimes (d_2 \bullet c_2) \tag{20}$$

$$(d_1 \boxtimes d_2) \bullet \langle\langle c_1, c_2\rangle\rangle = \langle\langle d_1 \bullet c_1, d_2 \bullet c_2\rangle\rangle \tag{21}$$

Strict parallelism yields a result dual to Lemma 4.

Lemma 5. *For any continuous functions $f : X_1 \to Y_1$, $g : X_2 \to Y_2$, the following equation holds*

$$\widehat{f} \boxtimes \widehat{g} = \widehat{f \times g} \tag{22}$$

Proof.

$$\begin{aligned}
& \widehat{f} \boxtimes \widehat{g} \\
= & \quad \{ \text{ Definition of } \boxtimes \} \\
& \langle \widehat{f} \bullet \widehat{\pi_1}, \widehat{g} \bullet \widehat{\pi_2} \rangle \rangle \\
= & \quad \{ \text{ } L \text{ is a functor } \} \\
& \langle \widehat{f \cdot \pi_1}, \widehat{g \cdot \pi_2} \rangle \rangle \\
= & \quad \{ \text{ Definition of } L, \times_1 \text{ (in } \mathbf{Top}_{\mathcal{H}}) \} \\
& \gamma \cdot \langle \eta \cdot f \cdot \pi_1, \eta \cdot g \cdot \pi_2 \rangle \\
= & \quad \{ \text{ Universal property of product (in } \mathbf{Top}) \} \\
& \gamma \cdot (\eta \times \eta) \cdot \langle f \cdot \pi_1, g \cdot \pi_2 \rangle \\
= & \quad \{ \text{ } \gamma \cdot (\eta_{Y_1} \times \eta_{Y_2}) = \eta_{Y_1 \times Y_2}, \text{ definition of } \times \text{ (in } \mathbf{Top}) \} \\
& \eta \cdot (f \times g) \\
= & \quad \{ \text{ Definition of } L \} \\
& \widehat{f \times g} \qquad \square
\end{aligned}$$

In some cases, however, putting two components in strict parallel may be too restrictive or not enough to meet the system's design requirements. The next section introduces a more relaxed version of parallelism where synchronisation comes into play. Mathematically, our construction explores the monoidal nature of functor $\mathcal{H}$.

6. Synchronised product and feedback

Synchronised parallelism is a form of composition in which components no longer need to be compatible in order to be put in parallel. Instead, each of them can change the duration of the corresponding evolutions according to the behaviour of the other. The price to be paid is that the previous pullback (or any limit in general) is no longer a suitable formalisation. Actually, adding a monoidal structure [26] to functor $\mathcal{H}$, as we will see in the sequel, seems to be a better alternative.

Definition 9. We say that functor $\mathcal{H}$ is monoidal (with respect to $\times$) if it comes equipped with a morphism $m : 1 \to \mathcal{H}1$, and a natural transformation $\delta : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$ that make the following diagrams to commute for any topological spaces $X, Y \in |\mathbf{Top}|$.

$$\begin{array}{ccc}
(\mathcal{H}X \times \mathcal{H}Y) \times \mathcal{H}Z & \xrightarrow{\alpha} & \mathcal{H}X \times (\mathcal{H}Y \times \mathcal{H}Z) \\
{\scriptstyle \delta \times id}\downarrow & & \downarrow{\scriptstyle id \times \delta} \\
\mathcal{H}(X \times Y) \times \mathcal{H}Z & & \mathcal{H}X \times \mathcal{H}(Y \times Z) \\
{\scriptstyle \delta}\downarrow & & \downarrow{\scriptstyle \delta} \\
\mathcal{H}((X \times Y) \times Z) & \xrightarrow[\mathcal{H}\alpha]{} & \mathcal{H}(X \times (Y \times Z))
\end{array}$$

$$\begin{array}{ccc}
\mathcal{H}X \times 1 & \xrightarrow{id \times m} & \mathcal{H}X \times \mathcal{H}1 \\
{\scriptstyle \pi_1}\downarrow & & \downarrow{\scriptstyle \delta} \\
\mathcal{H}X & \xleftarrow[\mathcal{H}\pi_1]{} & \mathcal{H}(X \times 1)
\end{array}
\qquad
\begin{array}{ccc}
1 \times \mathcal{H}X & \xrightarrow{m \times id} & \mathcal{H}1 \times \mathcal{H}X \\
{\scriptstyle \pi_2}\downarrow & & \downarrow{\scriptstyle \delta} \\
\mathcal{H}X & \xleftarrow[\mathcal{H}\pi_2]{} & \mathcal{H}(1 \times X)
\end{array}$$

Hence, functor $\mathcal{H}$ can be made monoidal once a suitable morphism $m : 1 \to \mathcal{H}1$ and a natural transformation $\delta : \mathcal{H} \times \mathcal{H} \to \mathcal{H}$ are defined.

Definition 10. Let us define such mappings as

$$m \mathrel{\widehat{=}} copy$$
$$\delta_{X,Y}\ ((f,d),(g,e)) \mathrel{\widehat{=}} (\langle f,g\rangle, d \curlyvee e)$$

where continuous function $\curlyvee : \mathsf{D} \times \mathsf{D} \to \mathsf{D}$ is defined as $\curlyvee \mathrel{\widehat{=}} \pi_1 \lhd (\geq) \rhd \pi_2$.

As a side note, observe that a possible definition of δ resorts to the minimum function $\curlywedge$ (instead of $\curlyvee$) but then the diagrams above would not commute. Indeed, for such an alternative to work, m would need to be changed into a variant of function $copy$ whose evolutions are always infinite.

Lemma 6. *δ is a natural transformation.*

Proof. We know that function $\delta : \mathcal{H}X \times \mathcal{H}Y \to \mathcal{H}(X \times Y)$ is defined as,

$$\mathcal{H}X \times \mathcal{H}Y \longrightarrow X^{\mathsf{T}} \times Y^{\mathsf{T}} \times \mathsf{D} \times \mathsf{D} \xrightarrow{\cong} (X \times Y)^{\mathsf{T}} \times \mathsf{D} \times \mathsf{D} \xrightarrow{id \times \curlyvee} \mathcal{H}(X \times Y).$$

Since $\curlyvee : \mathsf{D} \times \mathsf{D} \to \mathsf{D}$ is continuous, $\delta : \mathcal{H}X \times \mathcal{H}Y \to \mathcal{H}(X \times Y)$ must be continuous as well. To show that the naturality property holds, we reason

$$\begin{array}{ll} & \mathcal{H}(a \times b) \cdot \delta\ ((f,d),(g,e)) \\ = & \quad \{\ \text{Definition of } \mathcal{H} \text{ and } \delta\ \} \\ & ((a \times b) \cdot \langle f,g\rangle,\ d \curlyvee e) \\ = & \quad \{\ \text{Universal property of product}\ \} \\ & (\langle a \cdot f, b \cdot g\rangle,\ d \curlyvee e) \\ = & \quad \{\ \text{Definition of } \delta\ \} \\ & \delta\ ((a \cdot f, d),(b \cdot g, e)) \\ = & \quad \{\ \text{Definition of } \mathcal{H}\ \} \\ & \delta \cdot (\mathcal{H}a \times \mathcal{H}b)\ ((f,d),(g,e)) \qquad \square \end{array}$$

We can now state the expected result.

Theorem 6. *When equipped with natural transformation δ and morphism m, $\mathcal{H}$ is a monoidal functor.*

Proof. In appendix. □

The monoidal structure $\langle \mathcal{H}, \delta, m\rangle$ defines a specific operator for synchronised parallelism, which behaves as follows: given two components with the same domain $c_1 : I \to \mathcal{H}A$, $c_2 : I \to \mathcal{H}B$, define $\delta \cdot \langle c_1, c_2\rangle : I \to \mathcal{H}A \times \mathcal{H}B \to \mathcal{H}(A \times B)$, to be denoted in sequel by $(\!|c_1, c_2|\!)$.

System $(\!|c_1, c_2|\!)$ runs c_1 and c_2 in parallel; however, if one finishes earlier than the other, it is forced to stall its evolution so that both components end at the same time. In other words, the duration of the shorter evolution is increased by keeping it constant until the longer evolution terminates.

Again, this form of parallelism is a lax version of strict parallelism, the cost being that many laws that hold before are now lost. Nevertheless, the monoidal structure of $\mathcal{H}$ still makes straightforward to show the following properties.

$$\widehat{f \times g} \bullet (\!|c_1, c_2|\!) = (\!|\widehat{f} \bullet c_1, \widehat{g} \bullet c_2|\!) \tag{23}$$
$$\widehat{\alpha} \bullet (\!|(\!|c_1, c_2|\!), c_3|\!) = (\!|c_1, (\!|c_2, c_3|\!)|\!) \tag{24}$$
$$\widehat{\pi_1} \bullet (\!|c, copy|\!) = c \tag{25}$$
$$\widehat{\pi_2} \bullet (\!|copy, c|\!) = c \tag{26}$$

Moreover, we are able to canonically define a new operator, following a path similar to the one used to define $\boxplus$ and $\boxtimes$.

Definition 11. Given systems $c_1 : I_1 \nrightarrow O_1$, $c_2 : I_2 \nrightarrow O_2$, component $c_1 \,\boxed{S}\, c_2 : I_1 \times I_2 \nrightarrow O_1 \times O_2$ is defined by

$$c_1 \,\boxed{S}\, c_2 \mathrel{\widehat{=}} (\!|c_1 \bullet \widehat{\pi_1}\,, c_2 \bullet \widehat{\pi_2}\,|\!)$$

The following laws arise from routine calculations

$$\widehat{sw} \bullet (c_2 \boxed{S} c_1) = (c_1 \boxed{S} c_2) \cdot sw \tag{27}$$

$$\widehat{\alpha} \bullet ((c_1 \boxed{S} c_2) \boxed{S} c_3) = (c_1 \boxed{S} (c_2 \boxed{S} c_3)) \cdot \alpha \tag{28}$$

$$copy_X \boxed{S} copy_Y = copy_{X \times Y} \tag{29}$$

$$\widehat{f} \boxed{S} \widehat{g} = \widehat{f \times g} \tag{30}$$

Note that strict and synchronised parallel composition behave identically but with one exception: in any given execution, the latter increases the execution time of a system that finishes earlier than the other. Hence, for compatible components both operators behave exactly in the same way, and, therefore, the former inherits all laws derived in this section for the latter.

Next, we introduce iteration for continuous systems. This facilitates component specification and, moreover, can be used to express (or detect) *Zeno* behaviour [2].

Definition 12. Given a component $c : X \nrightarrow X$, component $c^n : X \nrightarrow X$ is defined by the (Kleisli) composition of c with itself n times. Formally,

$$c^0 \mathrel{\widehat{=}} copy, \quad c^n \mathrel{\widehat{=}} c^{n-1} \bullet c$$

It is straightforward to check that the following equations hold.

$$copy^n = copy \tag{31}$$

$$c^1 = c \tag{32}$$

$$(c^n)^m = c^{n \times m} \tag{33}$$

$$c^n \bullet c^m = c^{n+m} \tag{34}$$

$$(c \boxplus d)^n = c^n \boxplus d^n \tag{35}$$

$$(c \boxtimes d)^n = c^n \boxtimes d^n \tag{36}$$

Infinite iteration leads to the familiar notion of *feedback*.

Definition 13. Let (X, d) be a complete metric space, and $c : X \nrightarrow X$ a pre-dynamical system; denote the series $(\pi_2 \cdot c^i(x))_{i \in \mathbb{N}}$ by $(s_i)_{i \in \mathbb{N}}$, and the sequence $(\pi_1 \cdot c^i(x))_{i \in \mathbb{N}}$ by $(f_i)_{i \in \mathbb{N}}$.

Then, assume that for any $x \in X$ whenever the series $(s_i)_{i \in \mathbb{N}}$ converges the sequence $(f_i)_{i \in \mathbb{N}}$ is Cauchy. More concretely, its elements get progressively closer to each other with respect to the metric,

$$d^*(g, h) \mathrel{\widehat{=}} \sup_{t \in \mathsf{T}} d(g(t), h(t)).$$

The interested reader will find in [27] more details about this metric.

Finally, define infinite iteration $(X \overset{c}{\nrightarrow} X \overset{c}{\nrightarrow} X \overset{c}{\nrightarrow} \dots)$ as $\nu c : X \nrightarrow X$ where

$$\pi_2 \cdot \nu c\ (x) \mathrel{\widehat{=}} \begin{cases} \infty & \text{if the series } (s_i)_{i \in \mathbb{N}} \text{ diverges} \\ \lim_{i \to \infty} s_i & \text{otherwise} \end{cases}$$

$$(\pi_1 \cdot \nu c\ (x))\ t \mathrel{\widehat{=}} \begin{cases} f_k\ t & \text{if } t < (\pi_2 \cdot \nu c\ (x)) \\ \left(\lim_{i \to \infty} f_i\right) t & \text{otherwise} \end{cases}$$

for k the smallest value such that $t \leq s_k$.

Intuitively, to compute the value at a certain instant (t) in the evolution $(\pi_1 \cdot \nu c\ (x))$, we need to compose c with itself the necessary number of times for the composite 'to reach that instant'; only then it is possible to extract the value. To be concrete, if each iteration of c has two seconds of duration, to calculate the value at five seconds in the evolution $(\pi_1 \cdot \nu c\ (x))$, we consider the composite c^3 and compute the expression $(\pi_1 \cdot c^3\ (x))\ 5$.

Observe that, since c is pre-dynamical, the calculated value is not changed by additional iterations, *i.e.*

$$(\pi_1 \cdot c^k\ (x))\ t = (\pi_1 \cdot (c \bullet c^k)\ (x))\ t.$$

Actually, in the definition above one may forget the assumption of c being pre-dynamical as long as it is ensured that the sequence $(f_i)_{i \in \mathbb{N}}$ is always Cauchy.

The following section gives concrete examples of parallel operators and (infinite) iteration at work. The role of feedback in handling Zeno behaviour is illustrated as well.

7. From continuous to hybrid systems

Having characterised a calculus of continuous components based on the structure of the Kleisli category of monad $\mathcal{H}$, the next step is to broaden the picture in order to handle systems that exhibit continuous and discrete behaviour intertwined. Such is the purpose of this section. A number of examples will illustrate the approach proposed here as well as some of the operators introduced in the previous sections.

Our aim is to equip continuous systems with an (internal) state space that behaves in a discrete manner. Therefore, arrows become typed as

$$S \times I \longrightarrow S \times \mathcal{H}O\,.$$

Intuitively, given a state ($s \in S$) and an input ($i \in I$), the component transits (internally) into another state and presents continuous evolutions that can be directly observed. This gets us closer to the notion of hybrid system, as a family of continuous systems indexed by a state space. On the other hand, this approach is aligned with the notion of components as coalgebras (as described in [3]). Actually, our aim is to characterise hybrid systems as coalgebras with a discrete (internal) behaviour, and (external) continuous evolutions.

The cornerstone of this move from continuous to hybrid components is the notion of tensorial strength for monad $\mathcal{H}$: a natural transformation $\tau : Id \times \mathcal{H} \to \mathcal{H}(Id \times Id)$ that commutes with the monad operations and with specific monoidal structure of the base category (see the formal definition in [4]). Indeed, tensorial strength allows us to transport such systems to $\mathbf{Top}_{\mathcal{H}}$, via composition:

$$\frac{c : S \times I \to S \times \mathcal{H}O}{\tau \cdot c : S \times I \to \mathcal{H}(S \times O)}$$

Definition 14. Given topological spaces $X, Y \in |\mathbf{Top}|$ a (right) tensorial strength of monad $\mathcal{H}$ is the function $\tau_{X,Y} : X \times \mathcal{H}Y \to \mathcal{H}(X \times Y)$ defined by

$$\tau_{X,Y}\ (x, (f, d)) \mathrel{\widehat{=}} (\langle \underline{x}, f \rangle, d).$$

Interestingly, function τ corresponds to the *uniform characterisation* of tensorial strength for monads over **Set** (*cf.* [28]). This entails that all diagrams that need to commute do commute, and therefore we just need to show that τ is continuous. For this, observe that τ can alternatively be defined as $\langle \lambda\tau_a, \tau_b \rangle : X \times \mathcal{H}Y \to \mathcal{H}(X \times Y)$ where,

$$\tau_a\ ((x, (f, d)), t) \mathrel{\widehat{=}} (x, f\ t)$$
$$\tau_b\ (x, (f, d)) \mathrel{\widehat{=}} d$$

Since τ_a, τ_b are continuous, so is τ.

Corollary 2. *Natural transformation $\tau : Id \times \mathcal{H} \to \mathcal{H}(Id \times Id)$ defines a tensorial strength for monad $\mathcal{H}$.*

Note that one can also define a natural transformation $\tau_l : \mathcal{H} \times Id \to \mathcal{H}(Id \times Id)$ (known as left tensorial strength for $\mathcal{H}$), via the equation $\tau_l \mathrel{\widehat{=}} (\mathcal{H}sw) \cdot \tau \cdot sw$. Moreover, a monad is commutative, if the equation below holds.

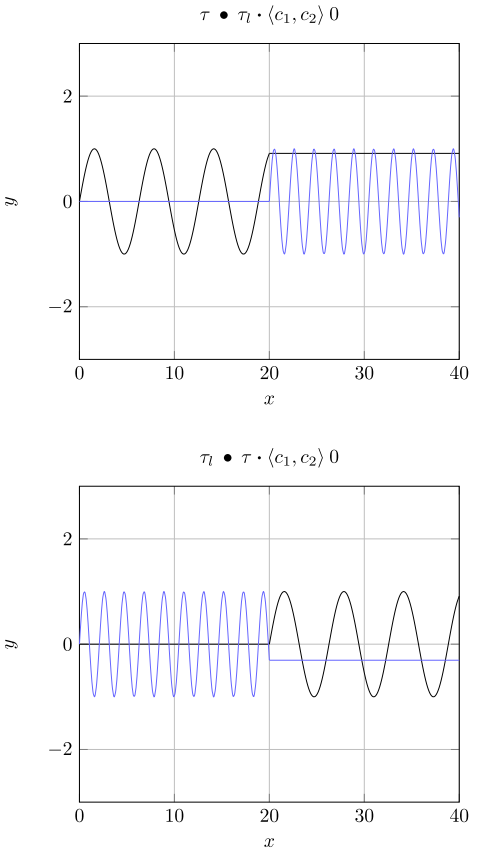
$$\tau \bullet \tau_l = \tau_l \bullet \tau$$

This is not, however, the case for monad $\mathcal{H}$, as the following counter-example reports.

Example 6. Recall the two signal generators, introduced in Example 5.

$$c_1\ x = (\ x + (\sin\ \_)\ ,\ 20\ ),\ \ c_2\ x = (\ x + \sin\ (3 \times \_)\ ,\ 20\ )$$

The application of left and right tensorial strength to the composed function $\langle c_1, c_2 \rangle : \mathbb{R} \to \mathcal{H}\mathbb{R} \times \mathcal{H}\mathbb{R}$ yields the behaviours depicted below.



Clearly, $\tau \bullet \tau_l \neq \tau_l \bullet \tau$; but note that the plots illustrate an interesting aspect: specification $\tau \bullet \tau_l \cdot \langle c_1, c_2 \rangle$ reads 'first let the component in the left to act, then the one in the right'; and conversely for $\tau_l \bullet \tau \cdot \langle c_1, c_2 \rangle$. Moreover, note that each component 'waits' for the other by stalling the corresponding evolution. This introduces yet another synchronisation mechanism.

Equipped with tensorial strength τ, we may now explore two classical examples of hybrid systems from a component-based perspective. We start with the *bouncing ball* system.

Example 7. Consider a bouncing ball dropped at some positive height and with no initial velocity. Due to the gravitational effect, it will fall into the ground but then bounce back up, losing, of course, part of its kinetic energy in the process.

From this description, one may regard the bouncing ball as a hybrid component whose (continuous) observable behaviour is the evolution of its spacial position, whereas the internal memory records velocity, updated at each bounce. To define such a component we resort to Newton's equations of motion.

$$pos_a\,(v, p, t) = p + vt - \tfrac{1}{2}at^2, \quad vel_a\,(v, t) = v - at$$

from which we can derive the function that, given a positive height and a current velocity, returns the time needed to reach the ground; formally,

$$zpos_a\,(v, p) = \frac{\sqrt{2ap + v^2} + v}{a}$$

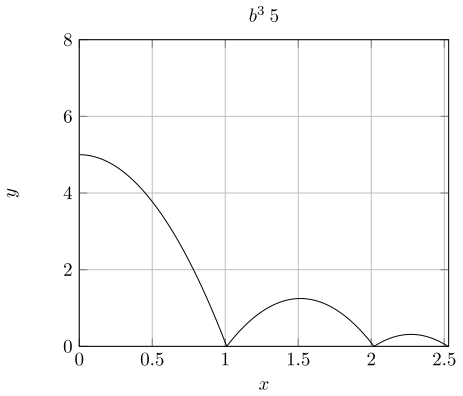
Let us then define the discrete behaviour of the bouncing ball $b_d : V \times P \to V$

$$b_d\,(v, p) \mathrel{\widehat{=}} vel_g(v, zpos_g(v, p)) \times -0.5$$

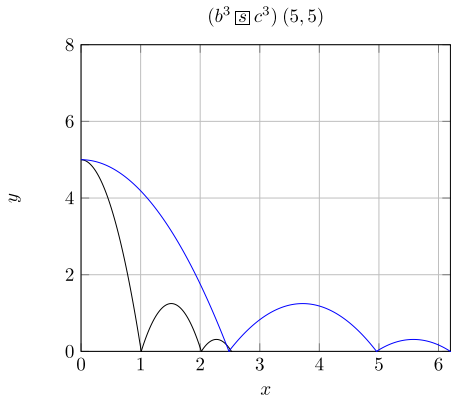
where 0.5 is the dampening coefficient. For the continuous part $b_c : V \times P \to \mathcal{H}P$

$$b_c \mathrel{\widehat{=}} \langle pos_g, zpos_g \rangle$$

where $g = 9.8$ (Earth's gravity). The resulting system is a ball bouncing on planet Earth, denoted by b and formally defined as $b \mathrel{\widehat{=}} \tau \cdot \langle b_d, b_c \rangle$. Assume that the initial state of b is 0. Then, through the iteration operator, and assuming five as the initial position one gets, for instance, the following behaviour.



Analogously, we can define a ball bouncing in the Moon (here denoted by letter c), and compare the behaviour of both bouncing balls by putting them in parallel, with the same initial state 0.



Note that $(b^3 \boxed{S} c^3) \neq (b \boxed{S} c)^3$. An interesting question to pose is about the durations that components νb and νc output. Indeed, the intuition is that durations are always infinite (since feedback involves infinite sums), however, due to the Zeno effect, the durations that concern this example are actually finite: they correspond to the time at which the ball stops moving. Such durations are given precisely by the computation of $\pi_2 \cdot (\nu b)$ and $\pi_2 \cdot (\nu c)$ with respect to a given input.

Example 8. *Alternating pumping systems* are often used to regulate the water level of reservoirs. Consider one that fills two tanks alternatively in cycles of ten seconds, which means that some sort of internal memory is required (to remember which was the last tank served).

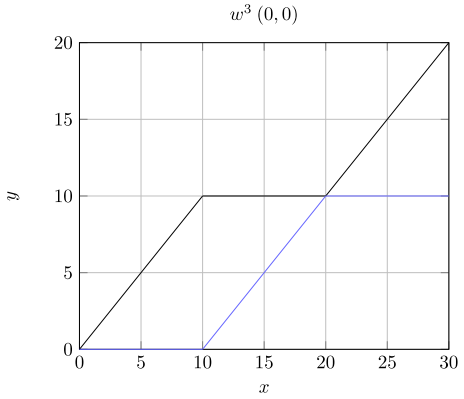
Thus, the discrete part $w_d : S \times L \to S$ is defined as

$$w_d \mathrel{\widehat{=}} flip \cdot \pi_1$$

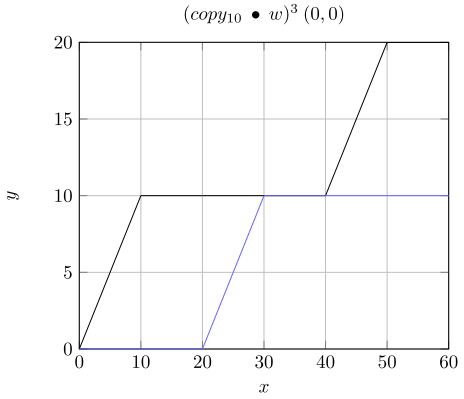
where $S = \{\top, \bot\}$ is the discrete state space and $flip$ the function that switches between the elements. Let us assume that the initial state is $\top$. Then, we define the continuous behaviour $w_c : S \times L \to \mathcal{H}L$

$$w_c(s, (l_1, l_2)) \mathrel{\widehat{=}} (f_s(l_1, l_2), 10)$$

where $f_\top(l_1, l_2) \mathrel{\widehat{=}} ((l_1 + \_),\ l_2)$ and $f_\bot(l_1, l_2) \mathrel{\widehat{=}} (l_1,\ (l_2 + \_))$. As expected, the pumping system is given by equation $w = \tau \cdot \langle w_d, w_c \rangle$, which, for input $(0, 0)$, yields the following plot.



On a different note, it is natural to consider that the pump takes some time to switch from one tank to the other: for illustration purposes let us assume that time to be ten seconds. To simulate such a delay we can define a variant of *copy*, denoted by $copy_{10}$, that always outputs evolutions with duration ten. Then, again for input $(0, 0)$, system $(copy_{10} \bullet w)^3$ outputs



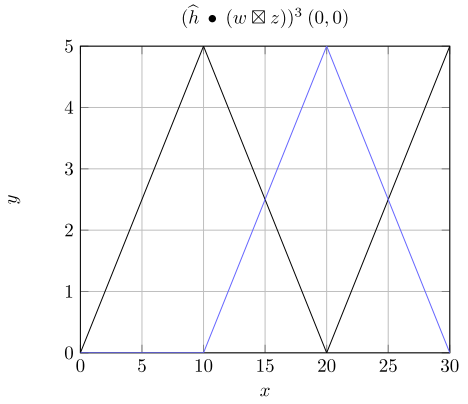
It is also important to analyse situations in which water flows out. Thus, consider a hybrid system $z : 1 \times 1 \to \mathcal{H}(1 \times L)$ (with trivial state space 1) whose continuous part

$$z_c(\star, \star) \mathrel{\widehat{=}} (\langle /_2, /_2 \rangle, 10)$$

dictates the rate of water flowing out in each tank, here represented by a clock that runs at half the normal speed. Then, we specify the result of w and z acting together in the same set of variables. For this, we define function $h : (S \times L) \times (1 \times L) \to (S \times L) \times (1 \times 1)$ where

$$h\ ((s, l_1, l_2), (\star, x, y)) \mathrel{\widehat{=}} ((s, l_1 \ominus x, l_2 \ominus y), (\star, \star))$$

Intuitively, function h subtracts water in accordance with the rate specified by component z. For input $(0, 0)$, system $\widehat{h} \bullet (w \boxtimes z)$ yields the plot below.



8. Conclusions and future work

It is well known that software systems are becoming prevalently intertwined with (continuous) physical processes. Such an architecture, however, renders their rigorous design (and analysis) a difficult challenge that calls for a wide, uniform framework combining the continuous and discrete sides of Mathematics.

As a first step towards a component-based framework for hybrid systems, in the spirit of [3], this paper showed how continuous evolutions can be encoded in the form of a strong (topological) monad. As discussed in Section 1, to capture specific behavioural models through monads has been a successful path in Computer Science: such was the case of non-deterministic behaviour, and the (discrete) probabilistic one; but occurrences in the continuous domain also exist. A prime example is the *Giry* monad [29], which captures stochastic processes and has been object of study in a number of papers (*e.g.*, [8,30–32]). Along similar lines, monad $\mathcal{H}$ provides a categorial universe for continuous, and hybrid systems, where the effects of continuity over different forms of composition can be isolated and suitably studied.

This universe, *i.e.* the Kleisli category $\mathbf{Top}_{\mathcal{H}}$, offers different forms of system composition, wiring mechanisms, and synchronisation techniques. For example, Kleisli composition lets the control of an evolution to be transferred from one system to the other, but also allows evolutions to be dynamically modified (as observed in the case of signal amplification). Such behavioural patterns, as discussed in Section 3, are often found in systems like thermostats, cruise control systems, and signal generators. But more generally, in control loop systems – traditionally comprised of a network of digital controllers that manage a physical process over time through a feedback loop architecture. In this case, the controllers, possessing different functionalities, periodically pass control of the physical process among themselves.

The underlying categorial framework hinted at several composition operators (through corresponding universal constructions), and facilitated the elicitation of several compositional laws. Throughout the paper, the results achieved were illustrated with classic examples of hybrid systems, namely a thermostat, a bouncing ball, and a water tank system.

8.1. Related work

Hybrid automata [33] are the *de facto* formalism for the specification of hybrid systems. Roughly speaking, they are a variant of classic automata that allows variables to continuously evolve while in a state. This defines the continuous behaviour of an hybrid system, which is then paired with discrete actions given by the usual state transitions. Parallel composition of hybrid automata proceeds similarly to the classic case, where common labels act as synchronising events. Interestingly, in [34] Bornot and Sifakis introduced additional synchronisation mechanisms that make one system wait for the evolution of the other to end, or, on the contrary, force it to finish earlier. This seems to be intimately related to whatever monoidal structure is given to functor $\mathcal{H}$.

During the last years there were also developments concerning the addition of new dimensions to hybrid automata: for example, [35] shows how to take *reaction times* into consideration in a compositional setting. In our case, we took advantage of dawdler components, like $copy_{10}$, to introduce such delays.

The 'rationale' underlying hybrid automata is powerful, and highly intuitive, but in some cases lacks expressive power: for example, those systems in which evolutions can be dynamically changed by some of the components are very hard to specify. Moreover, aside from parallel composition, the authors have no knowledge of deep developments that concern new compositional operators for hybrid automata.

The industrial tool Simulink,[1] on the other hand, offers a highly expressive component-based language, and is thus closely related to the framework proposed in this paper. Indeed, Simulink supports a rich palette of compositional operators, and computational units. It possesses behavioural patterns that involve dynamical alteration of evolutions, delays, and

[1] http://www.mathworks.com/products/simulink.

synchronisation. Moreover, the transfer of the control of some evolution is not hard to define. All this renders Simulink a very interesting tool. The cost is the lack of a clear semantics, which impairs formal analysis and the elicitation of compositional laws – actually, some recent efforts have been made towards the formal verification of Simulink models in alternative tools (*cf.* [36,37]). In addition, the components available are rather limited in what concerns the characterisation of their internal memory and respective transition dynamics.

It would be interesting to study the embedding of (a subset of) Simulink's language into $\mathbf{Top}_{\mathcal{H}}$. In principle, $\mathbf{Top}_{\mathcal{H}}$ could act as a tool complement, providing a basis for the formal analysis of (critical fragments) of hybrid systems. We stress, however, that we do not aim at emulating Simulink, but rather at a suitable coalgebraic framework for hybrid components, where we consider the discrete transitions to be internal behaviour, and the continuous evolutions the observable part. From this point of view, Simulink is very distant from such a line of work.

There is also a close relation between the work here reported and P. Höfner's algebra of hybrid systems [38]: the latter's main operator is used to concatenate evolutions. Moreover, the algebra possesses secondary operators, like parallelism and synchronisation, that are equally available in $\mathbf{Top}_{\mathcal{H}}$. Our approach, however, and differently from P. Höfner's calculus, is structured around a monad that encodes the notion of continuous evolution; this brings up a number of canonical constructions and smooths the integration with other behavioural effects, such as nondeterminism or probabilistic behaviour.

Finally, a few categorial models for hybrid systems have been proposed along the last two decades. For example, document [39] introduces an institution – in essence, a categorial rendering of a logic – for hybrid systems, and provides basic forms of composition such as free aggregation (*i.e.*, parallelism without interaction) and interconnection where some attributes and events are shared between two systems. Around the same time, Jacobs [40] suggested an object oriented coalgebraic framework where hybrid systems are regarded as coalgebras equipped with a monoid action: coalgebras define the discrete transitions, and monoid actions the continuous evolutions. Some years later Haghverdi et al. [41] explored the connection between a formalisation of hybrid systems (close to hybrid automata) and open maps. The objective was to provide appropriate notions of bisimulation both for dynamical, and hybrid systems. Composition mechanisms, however, were not studied in this context.

8.2. Future work

Our next step is the development of a calculus of hybrid components (as in [3]) based on monad $\mathcal{H}$ and its Kleisli category. The calculus from [3], in its coalgebraic spirit, is bisimulation-based, with bisimulation given as the usual span of simulations [28]. The framework that this paper sets, however, offers a promising basis to explore alternative notions of (bi)simulation for continuous and hybrid systems. This has points of contact with the work of Haghverdi et al. in [41]; but note that we use coalgebraic machinery, and follow a component-based perspective, which makes possible to study the relation between (bi)simulation and (the different) compositional operators.

A second line of research concerns the development of a taxonomy of continuous, and hybrid systems living in $\mathbf{Top}_{\mathcal{H}}$. Indeed, as Stauner showed at the beginning of the century in his PhD thesis [42], topologies are useful to elicit a number of important properties. For example, the notion of *robustness* (prevalent in control theory) becomes simple to formulate: intuitively, a system is robust if small changes in the input lead to very similar evolutions. In $\mathbf{Top}_{\mathcal{H}}$, since each system has a topological semantic base, one can express how robust it is by varying the topology in its source object. At one limit, if the topology is discrete, the system is seen as *chaotic*. At the other end, *i.e.*, if the topology is indiscrete, the system must always output the same evolution.

Actually, the compositional nature that underlies $\mathbf{Top}_{\mathcal{H}}$ allows us to reason about the robustness of the system at hands through the analysis of (the robustness of) its simpler constituents. One disadvantage of this approach is that composition in $\mathbf{Top}_{\mathcal{H}}$ is *strict*, in the sense that components with different topologies in the connecting points cannot be composed. For example, it is hard to put a chaotic component after a robust one. Part of our current research tries to relax this condition while maintaining stability, whenever possible.

Acknowledgements

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation – COMPETE 2020 Programme and by National Funds through the Portuguese funding agency, FCT – Fundação para a Ciência e a Tecnologia within project POCI-01-0145-FEDER-016692.

The first author is also sponsored by FCT grant SFRH/BD/52234/2013, and the second one by FCT grant SFRH/BSAB/113890/2015. Moreover, D. Hofmann and M. Martins are supported by the EU FP7 Marie Curie PIRSES-GA-2012-318986 project GeTFun: Generalizing Truth-Functionality and FCT project UID/MAT/04106/2013 through CIDMA.

We are grateful for the interesting discussions that the first author had with Ichiro Hasuo, Toshiki Kataoka, and Soichiro Fujii about the characterisation of monad $\mathcal{H}$. Finally, we gratefully acknowledge the anonymous reviewers for their interesting comments along the revision process.

Appendix

Proof of Lemma 2. The proof is divided in two parts: the first establishes continuity of the mappings, the second concerns naturality. Consider the mapping $\mu_X : \mathcal{H}\mathcal{H}X \to \mathcal{H}X$; we are going to show its continuity. First we observe that μ_X can be alternatively defined as $\langle \lambda a, b \rangle$ where

$$a \mathrel{\widehat{=}} \mathcal{H}\mathcal{H}X \times \mathsf{T} \xrightarrow{i \times id} (X^{\mathsf{T}\times\mathsf{T}} \times \mathsf{D}) \times \mathsf{T} \xrightarrow{conc} X$$

$$i \mathrel{\widehat{=}} \mathcal{H}\mathcal{H}X \xrightarrow{\pi_1^{\mathsf{T}} \times id} (X^{\mathsf{T}})^{\mathsf{T}} \times \mathsf{D} \xrightarrow{\cong} X^{\mathsf{T}\times\mathsf{T}} \times \mathsf{D}$$

for $conc\,((f,d),t) \mathrel{\widehat{=}} f\,(t \curlywedge d, t \ominus d)$. The definitions clearly show that a is continuous. For function b we have

$$b \mathrel{\widehat{=}} \mathcal{H}\mathcal{H}X \xrightarrow{\pi_2^{\mathsf{T}} \times id} \mathsf{D}^{\mathsf{T}} \times \mathsf{D} \xrightarrow{c} \mathsf{D}$$

where $c\,(f,d) \mathrel{\widehat{=}} (f\,d) + d \lhd (d \neq \infty) \rhd \infty$. Since the canonical restriction $(+) \cdot \langle ev, \pi_2 \rangle : \mathsf{D}^{\mathsf{T}} \times \mathsf{T} \to \mathsf{D}$ of c is continuous we just need to show that the latter is continuous at infinity. Actually, this comes for free once proved that given any neighbourhood $N \supseteq (x, \infty]$ in D of ∞ we can find a neighbourhood V in $\mathsf{D}^{\mathsf{T}} \times \mathsf{D}$ of (f, ∞) such that $c\,(V) \subseteq N$.

Consider neighbourhood $\mathsf{D}^{\mathsf{T}} \times (x, \infty]$. It is clear that $c\,(\mathsf{D}^{\mathsf{T}} \times (x, \infty]) \subseteq (x, \infty] \subseteq N$.

Next we show that μ is natural, *i.e.*, that for any continuous function $h : X \to Y$ the diagram below commutes.

$$\begin{array}{ccc} \mathcal{H}\mathcal{H}X & \xrightarrow{\mathcal{H}\mathcal{H}h} & \mathcal{H}\mathcal{H}Y \\ {\scriptstyle \mu_X}\downarrow & & \downarrow{\scriptstyle \mu_Y} \\ \mathcal{H}X & \xrightarrow[\mathcal{H}h]{} & \mathcal{H}Y \end{array}$$

First we assume that $(f,d) \in \mathcal{H}\mathcal{H}X$ has finite duration,

$$\begin{array}{ll} & \mu \cdot \mathcal{H}\mathcal{H}h\,(f,d) \\ = & \quad \{ \text{ Definition of } \mathcal{H}, \mu \} \\ & (\theta \cdot \mathcal{H}h \cdot f, d) +\!\!+ (\mathcal{H}h \cdot f\,d) \\ = & \quad \{ \text{ } \theta \text{ is natural } \} \\ & (h \cdot \theta \cdot f, d) +\!\!+ (\mathcal{H}h \cdot f\,d) \\ = & \quad \{ \text{ Definition of } \mathcal{H}, \text{ composition } \} \\ & \mathcal{H}h\,(\theta \cdot f, d) +\!\!+ \mathcal{H}h\,(f\,d) \\ = & \quad \{ \text{ } (+\!\!+) \text{ is natural } \} \\ & \mathcal{H}h\,((\theta \cdot f, d) +\!\!+ (f\,d)) \\ = & \quad \{ \text{ Definition of } \mathcal{H}, \mu \} \\ & \mathcal{H}h \cdot \mu\,(f,d) \end{array}$$

The proof for the case in which $(f,d) \in \mathcal{H}\mathcal{H}X$ has infinite duration is analogous to the above. □

Proof of Theorem 2. We have to show that the following diagrams commute.

$$\begin{array}{ccccc} \mathcal{H} & \xrightarrow{\eta_{\mathcal{H}}} & \mathcal{H}^2 & \xleftarrow{\mathcal{H}\eta} & \mathcal{H} \\ & {\scriptstyle 1_{\mathcal{H}}}\searrow & {\scriptstyle \mu}\downarrow & \swarrow{\scriptstyle 1_{\mathcal{H}}} & \\ & & \mathcal{H} & & \end{array} \qquad \begin{array}{ccc} \mathcal{H}^3 & \xrightarrow{\mu_{\mathcal{H}}} & \mathcal{H}^2 \\ {\scriptstyle \mathcal{H}\mu}\downarrow & & \downarrow{\scriptstyle \mu} \\ \mathcal{H}^2 & \xrightarrow[\mu]{} & \mathcal{H} \end{array}$$

Note that the proof below becomes much more simpler if the evolutions involved have infinite duration.

Let us start with the left triangle.

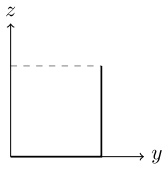
$$
\begin{aligned}
& \mu \cdot \eta\ (f, d) \\
= \quad & \{\ \text{Definition of } \eta\ \} \\
& \mu\ (\underline{(f, d)}, 0) \\
= \quad & \{\ \text{Definition of } \mu\ \} \\
& (\theta \cdot \underline{(f, d)}, 0) + ((\underline{f, d})\ 0) \\
= \quad & \{\ \text{Definition of constant}\ \} \\
& (\theta \cdot \underline{(f, d)}, 0) + (f, d) \\
= \quad & \{\ \text{Definition of } +\!\!+\text{, definition of constant}\ \} \\
& (f, d)
\end{aligned}
$$

For the right triangle we have,

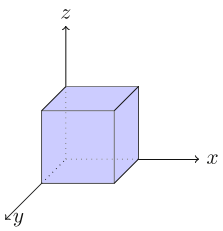
$$
\begin{aligned}
& \mu \cdot \mathcal{H}\eta\ (f, d) \\
= \quad & \{\ \text{Definition of } \mathcal{H}\ \} \\
& \mu\ (\eta \cdot f, d) \\
= \quad & \{\ \text{Definition of } \mu\ \} \\
& (\theta \cdot \eta \cdot f, d) + (\eta \cdot f\ \ d) \\
= \quad & \{\ \text{Definition of } \eta\ \} \\
& (\theta \cdot \eta \cdot f, d) + (\underline{f\ d}, 0) \\
= \quad & \{\ \text{Definition of } +\!\!+\ \} \\
& (\theta \cdot \eta \cdot f, d) \\
= \quad & \{\ \text{Eilenberg–Moore}\ \} \\
& (f, d)
\end{aligned}
$$

It remains to show that the square commutes. Before giving the formal proof, we present the corresponding intuition from a geometric perspective.

Let us then start by observing that an element in $\mathcal{H}\mathcal{H}X$ may be intuitively seen as a square, where each column is a function in $\mathcal{H}X$. Then, note that multiplication ($\mu : \mathcal{H}\mathcal{H}X \to \mathcal{H}X$) keeps just the first row and last column of the square, as illustrated below.

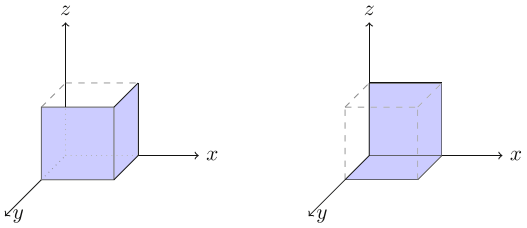


As expected, the intuitive picture of an element in $\mathcal{H}^3X$ is a cube,

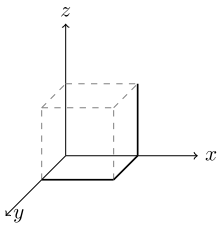


such that a projection on the x-axis yields an element of $\mathcal{H}\mathcal{H}X$ (geometrically, a square as described above).

Let us now observe that, resorting to multiplication, we can reduce the cube into a square. Actually, we can do this in two different ways: via $\mu : \mathcal{H}^3 X \to \mathcal{H}\mathcal{H}X$, or $\mathcal{H}\mu : \mathcal{H}^3 X \to \mathcal{H}\mathcal{H}X$. In the former case, only the front and right surfaces are kept (picture below in the left). In contrast, function $\mathcal{H}\mu$ applies μ to each projection on the x-axis, and thus only the bottom and back surfaces are kept (picture below in the right).



Finally, applying $\mu : \mathcal{H}\mathcal{H}X \to \mathcal{H}X$ to the resulting squares yields the same result,



More formally, we reason

$$
\begin{array}{ll}
 & \mu \cdot \mathcal{H}\mu\ (f, d) \\
= & \quad \{\ \text{Definition of } \mathcal{H}\ \} \\
 & \mu\ (\mu \cdot f, d) \\
= & \quad \{\ \text{Definition of } \mu\ \} \\
 & (\theta \cdot \mu \cdot f, d) \mathbin{+\!\!+} (\mu \cdot f\ d) \\
= & \quad \{\ \text{Eilenberg–Moore}\ \} \\
 & (\theta \cdot \mathcal{H}\theta \cdot f, d) \mathbin{+\!\!+} (\mu \cdot f\ d) \\
= & \quad \{\ \text{Let } f\ d = (f', d'), \text{ definition of } \mathbin{+\!\!+}\ \} \\
 & \big((\theta \cdot \mathcal{H}\theta \cdot f, d\ ) \mathbin{+\!\!+} (\theta \cdot f', d') \big) \mathbin{+\!\!+} (f'\ d') \\
= & \quad \{\ \theta \text{ is natural}\ \} \\
 & \big((\theta \cdot \theta \cdot f, d\ ) \mathbin{+\!\!+} (\theta \cdot f', d') \big) \mathbin{+\!\!+} (f'\ d') \\
= & \quad \{\ \text{Notation } (f\ d), \text{ definition of } \mathbin{+\!\!+}, \text{ definition of } \mathcal{H}\ \} \\
 & \big(\mathcal{H}\theta\ ((\theta \cdot f, d\ ) \mathbin{+\!\!+} (f\ d)) \big) \mathbin{+\!\!+} (f'\ d') \\
= & \quad \{\ \text{Definition of } \mu\ \} \\
 & \big(\mathcal{H}\theta \cdot \mu\ (f, d) \big) \mathbin{+\!\!+} (f'\ d') \\
= & \quad \{\ \text{Definition of } \mu\ \} \\
 & (\ \mathcal{H}\theta \cdot \mu\ (f, d)\ ) \mathbin{+\!\!+} (\ \pi_1 \cdot \mu\ (f, d)\ \ \pi_2 \cdot \mu\ (f, d)\ ) \\
= & \quad \{\ \text{Definition of } \mathcal{H}\ \} \\
 & (\theta \cdot \pi_1 \cdot \mu\ (f, d), \pi_2 \cdot \mu\ (f, d)) \mathbin{+\!\!+} (\pi_1 \cdot \mu\ (f, d)\ \ \pi_2 \cdot \mu\ (f, d)) \\
= & \quad \{\ \text{Definition of } \mu\ \}
\end{array}
$$

$$\begin{aligned} & \mu\ (\mu\ (f, d)) \\ = \quad & \{\ \text{Composition}\ \} \\ & \mu \cdot \mu\ (f, d) \qquad \square \end{aligned}$$

Proof of Theorem 5. Consider the following pullback in **Top**

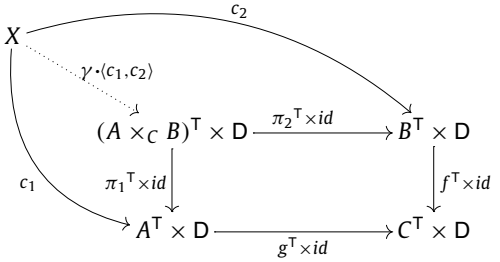
$$\begin{array}{ccc} A \times_C B & \xrightarrow{\pi_2} & B \\ {\scriptstyle \pi_1}\downarrow & & \downarrow{\scriptstyle g} \\ A & \xrightarrow[f]{} & C \end{array}$$

where f and g are arbitrary continuous functions. We need to show that

$$\begin{array}{ccc} \mathcal{H}(A \times_C B) & \xrightarrow{\mathcal{H}\pi_2} & \mathcal{H}B \\ {\scriptstyle \mathcal{H}\pi_1}\downarrow & & \downarrow{\scriptstyle \mathcal{H}g} \\ \mathcal{H}A & \xrightarrow[\mathcal{H}f]{} & \mathcal{H}C \end{array}$$

also forms a pullback in **Top**.

For this, observe that functor $\mathcal{H}$ comes from the composition of functors $(\_)^{\mathsf{T}}$, and $(\_ \times \mathsf{D})$, both of which preserve pullbacks. Indeed, they give rise to the commuting diagram



where $\gamma\ ((e_1, d), (e_2, d)) = (\langle e_1, e_2\rangle, d)$. Let us denote $\gamma \cdot \langle c_1, c_2\rangle$ by $\langle\langle c_1, c_2\rangle\rangle$.

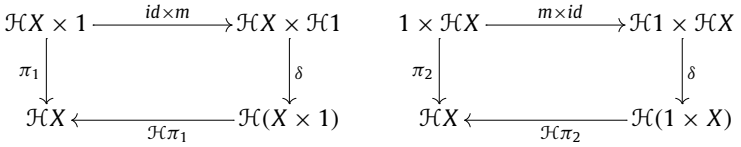
Since functor $\mathcal{H}$ forces specific conditions on evolutions (recall that $(e, d) \in \mathcal{H}X$ implies $e \cdot \curlywedge_d = e$) some work remains to be done. In fact, we need to show that $\text{img}\ \langle\langle c_1, c_2\rangle\rangle \subseteq \mathcal{H}(A \times_C B)$ whenever $\text{img}\ c_1 \subseteq \mathcal{H}A$, $\text{img}\ c_2 \subseteq \mathcal{H}B$, and c_1, c_2 make the outer square to commute. In other words, we need to show that, under these conditions, $\langle\langle c_1, c_2\rangle\rangle$ factors through $\iota : \mathcal{H}(A \times_C B) \hookrightarrow (A \times_C B)^{\mathsf{T}} \times \mathsf{D}$; diagrammatically,

$$\begin{array}{ccc} X & \xrightarrow{\langle\langle c_1, c_2\rangle\rangle} & (A \times_C B)^{\mathsf{T}} \times \mathsf{D} \\ & \searrow & \uparrow{\scriptstyle \iota} \\ & & \mathcal{H}(A \times_C B) \end{array}$$

Consider an element $x \in X$, and denote $\langle\langle c_1, c_2\rangle\rangle\ x$ by $(\langle e_1, e_2\rangle, d)$. Since by assumption $e_1 \cdot \curlywedge_d = e_1$, $e_2 \cdot \curlywedge_d = e_2$, it is clear that $\langle e_1, e_2\rangle \cdot \curlywedge_d = \langle e_1, e_2\rangle$ and therefore $(\langle e_1, e_2\rangle, d) \in \mathcal{H}(A \times_C B)$. $\square$

Proof of Theorem 6. We need to show that the following diagrams commute.

$$\begin{array}{ccc} (\mathcal{H}X \times \mathcal{H}Y) \times \mathcal{H}Z & \xrightarrow{\alpha} & \mathcal{H}X \times (\mathcal{H}Y \times \mathcal{H}Z) \\ {\scriptstyle \delta \times id}\downarrow & & \downarrow{\scriptstyle id \times \delta} \\ \mathcal{H}(X \times Y) \times \mathcal{H}Z & & \mathcal{H}X \times \mathcal{H}(Y \times Z) \\ {\scriptstyle \delta}\downarrow & & \downarrow{\scriptstyle \delta} \\ \mathcal{H}((X \times Y) \times Z) & \xrightarrow[\mathcal{H}\alpha]{} & \mathcal{H}(X \times (Y \times Z)) \end{array}$$



We start with the upper square.

$$
\begin{aligned}
& \mathcal{H}\alpha \cdot \delta \cdot (\delta \times id)\ \big(((e_1, d_1), (e_2, d_2)), (e_3, d_3)\big) \\
= \quad & \{\ \text{Definition of } \delta \text{ and } \mathcal{H}\ \} \\
& (\alpha \cdot \langle \langle e_1, e_2 \rangle, e_3 \rangle, ((d_1 \curlyvee d_2) \curlyvee d_3))\ ) \\
= \quad & \{\ \text{Definition of product, } \curlyvee \text{ is associative}\ \} \\
& (\langle e_1, \langle e_2, e_3 \rangle \rangle, (d_1 \curlyvee (d_2 \curlyvee d_3)))\ ) \\
= \quad & \{\ \text{Definition of } \delta\ \} \\
& \delta\ (\ (e_1, d_1), (\langle e_2, e_3 \rangle, d_2 \curlyvee d_3)\ ) \\
= \quad & \{\ \text{Definition of } id \times \delta\ \} \\
& \delta \cdot (id \times \delta)\ \big((e_1, d_1), ((e_2, d_2), (e_3, d_3))\big) \\
= \quad & \{\ \text{Definition of } \alpha\ \} \\
& \delta \times (id \times \delta) \cdot \alpha\ \big(((e_1, d_1), (e_2, d_2)), (e_3, d_3)\big)
\end{aligned}
$$

Then, for the diagram above in the left we reason, and proceed similarly with the one in the right.

$$
\begin{aligned}
& \mathcal{H}\pi_1 \cdot \delta \cdot (id \times m)\ ((f, d), \star) \\
= \quad & \{\ \text{Definition of } m, \delta, \text{ and } \mathcal{H}\ \} \\
& (\pi_1 \cdot \langle f, \underline{\star} \rangle, d \curlyvee 0) \\
= \quad & \{\ \text{Cancellation } \times, 0 \text{ is the identity element (for } \curlyvee)\ \} \\
& (f, d) \\
= \quad & \{\ \text{Definition of } \pi_1\ \} \\
& \pi_1\ ((f, d), \star) \qquad \square
\end{aligned}
$$

References

[1] P. Tabuada, Verification and Control of Hybrid Systems – A Symbolic Approach, Springer, 2009.
[2] R. Alur, Principles of Cyber-Physical Systems, MIT Press, 2015.
[3] L.S. Barbosa, Towards a calculus of state-based software components, J. Univers. Comput. Sci. 9 (2003) 891–909.
[4] A. Kock, Strong functors and monoidal monads, Arch. Math. 23 (1) (1972) 113–120.
[5] E. Moggi, Notions of computation and monads, Inf. Comput. 93 (1) (1991) 55–92.
[6] P. Wadler, Monads for functional programming, in: J. Jeuring, E. Meijer (Eds.), Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, Båstad, Sweden, May 24–30, 1995, in: Lecture Notes in Computer Science, vol. 925, Springer, 1995, pp. 24–52, Tutorial Text.
[7] S. Marlow, R. Newton, S.L.P. Jones, A monad for deterministic parallelism, in: K. Claessen (Ed.), Proceedings of the 4th ACM SIGPLAN Symposium on Haskell, Haskell 2011, Tokyo, Japan, 22 September 2011, ACM, 2011, pp. 71–82.
[8] E.-E. Doberkat, Stochastic Coalgebraic, Logic, Monographs in Theoretical Computer Science. An EATCS Series, Springer, 2009.
[9] I. Hasuo, B. Jacobs, A. Sokolova, Generic trace theory, Electron. Notes Theor. Comput. Sci. 164 (1) (2006) 47–65.
[10] E. Moggi, Computational lambda-calculus and monads, in: Proceedings of the Fourth Annual Symposium on Logic in Computer Science, LICS '89, Pacific Grove, California, USA, June 5–8, 1989, IEEE Computer Society, 1989, pp. 14–23.
[11] L.S. Barbosa, J.N. Oliveira, Transposing partial components: an exercise on coalgebraic refinement, Theor. Comput. Sci. 365 (1–2) (2006) 2–22.
[12] J.N. Oliveira, Preparing relational algebra for "just good enough" hardware, in: P. Höfner, P. Jipsen, W. Kahl, M.E. Müller (Eds.), Proceedings of Relational and Algebraic Methods in Computer Science – 14th International Conference, RAMiCS 2014, Marienstatt, Germany, April 28–May 1, 2014, in: Lecture Notes in Computer Science, vol. 8428, Springer, 2014, pp. 119–138.
[13] E. Meijer, J. Jeuring, Merging monads and folds for functional programming, in: J. Jeuring, E. Meijer (Eds.), International Summer School on Advanced Functional Programming, in: Lect. Notes Comp. Sci., vol. 925, Springer, 1995, pp. 228–266.
[14] A. Pardo, Monadic corecursion – definition, fusion laws, and applications, Electron. Notes Theor. Comput. Sci. 11 (1998) 105–139.
[15] J.F. Ferreira, A. Mendes, R. Backhouse, L.S. Barbosa, Which mathematics for the information society?, in: J. Gibbons, J.N. Oliveira (Eds.), Inter. Conf. on Teaching Formal Methods, TFM'09, in: Lect. Notes Comp. Sci., vol. 5846, Springer, 2009, pp. 39–56.

[16] J.N. Oliveira, Extended static checking by calculation using the pointfree transform, in: A. Bove, L.S. Barbosa, A. Pardo, J.S. Pinto (Eds.), Language Engineering and Rigorous Software Development, International LerNet ALFA Summer School 2008, Piriapolis, Uruguay, February 24–March 1, 2008, in: Lecture Notes in Computer Science, vol. 5520, Springer, 2009, pp. 195–251, Revised Tutorial Lectures.
[17] S. Mu, J.N. Oliveira, Programming from Galois connections, J. Log. Algebraic Program. 81 (6) (2012) 680–704.
[18] J. Oliveira, Towards a linear algebra of programming, Form. Asp. Comput. 24 (4–6) (2012) 433–458.
[19] H.D. Macedo, J.N. Oliveira, Typing linear algebra: a biproduct-oriented approach, Sci. Comput. Program. 78 (11) (2013) 2160–2191.
[20] A. Katok, B. Hasselblatt, Introduction to the Modern Theory of Dynamical Systems, Encyclopedia of Mathematics and Its Applications, Cambridge University Press, Cambridge, 1996. URL http://opac.inria.fr/record=b1077897.
[21] M. Brin, G. Stuck, Introduction to Dynamical Systems, Cambridge University Press, 2002.
[22] M. Escardó, R. Heckmann, Topologies on spaces of continuous functions, in: Topology Proceedings, vol. 26, 2001, pp. 545–564.
[23] R. Brown, Moore hyperrectangles on a space form a strict cubical omega-category, arXiv:0909.2212, 2009.
[24] J. Goubault-Larrecq, Non-Hausdorff Topology and Domain Theory – Selected Topics in Point-Set Topology, New Mathematical Monographs, vol. 22, Cambridge University Press, 2013.
[25] J. Adámek, H. Herrlich, G.E. Strecker, Abstract and Concrete Categories – The Joy of Cats, Dover Publications, 2009.
[26] G.J. Seal, Tensors, monads and actions, Theory Appl. Categ. 28 (15) (2013) 403–433.
[27] J. Kelley, General Topology, Van Nostrand, 1955, reprinted by Graduate Texts in Mathematics, vol. 27, Springer-Verlag, 1975.
[28] B. Jacobs, Introduction to coalgebra. Towards mathematics of states and observations, URL http://www.cs.ru.nl/B.Jacobs/CLG/JacobsCoalgebraIntro.pdf, 2012.
[29] P. Panangaden, Probabilistic relations, in: School of Computer Science, McGill University, Montreal, 1998, pp. 59–74.
[30] E.-E. Doberkat, Kleisli morphisms and randomized congruences for the Giry monad, J. Pure Appl. Algebra 211 (3) (2007) 638–664.
[31] V. Danos, J. Desharnais, F. Laviolette, P. Panangaden, Bisimulation and cocongruence for probabilistic systems, Inf. Comput. 204 (4) (2006) 503–523.
[32] B. Jacobs, Measurable spaces and their effect logic, in: 28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25–28, 2013, 2013, pp. 83–92.
[33] T.A. Henzinger, The theory of hybrid automata, in: Proceedings, 11th Annual IEEE Symposium on Logic in Computer Science, New Brunswick, New Jersey, USA, July 27–30, 1996, IEEE Computer Society, 1996, pp. 278–292.
[34] S. Bornot, J. Sifakis, On the composition of hybrid systems, in: T.A. Henzinger, S. Sastry (Eds.), Hybrid Systems: Computation and Control, Proceedings of First International Workshop, HSCC'98, Berkeley, California, USA, April 13–15, 1998, in: Lecture Notes in Computer Science, vol. 1386, Springer, 1998, pp. 49–63.
[35] W. Damm, H. Dierks, J. Oehlerking, A. Pnueli, Towards component based design of hybrid systems: safety and stability, in: Z. Manna, D.A. Peled (Eds.), Time for Verification, Essays in Memory of Amir Pnueli, in: Lecture Notes in Computer Science, vol. 6200, Springer, 2010, pp. 96–143.
[36] L. Zou, N. Zhan, S. Wang, M. Fränzle, Formal verification of simulink/stateflow diagrams, in: B. Finkbeiner, G. Pu, L. Zhang (Eds.), Proceedings of Automated Technology for Verification and Analysis – 13th International Symposium, ATVA 2015, Shanghai, China, October 12–15, 2015, in: Lecture Notes in Computer Science, vol. 9364, Springer, 2015, pp. 464–481.
[37] R. Reicherdt, S. Glesner, Formal verification of discrete-time matlab/simulink models using boogie, in: D. Giannakopoulou, G. Salaün (Eds.), Proceedings of Software Engineering and Formal Methods – 12th International Conference, SEFM 2014, Grenoble, France, September 1–5, 2014, in: Lecture Notes in Computer Science, vol. 8702, Springer, 2014, pp. 190–204.
[38] P. Höfner, Algebraic calculi for hybrid systems, Ph.D. thesis, University of Augsburg, 2009.
[39] H. Lourenço, A. Sernadas, An institution of hybrid systems, in: D. Bert, C. Choppy, P.D. Mosses (Eds.), Recent Trends in Algebraic Development Techniques, in: Lecture Notes in Computer Science, vol. 1827, Springer, Berlin, Heidelberg, 2000, pp. 219–236.
[40] B. Jacobs, Object-oriented hybrid systems of coalgebras plus monoid actions, Theor. Comput. Sci. 239 (1) (2000) 41–95.
[41] E. Haghverdi, P. Tabuada, G.J. Pappas, Bisimulation relations for dynamical, control, and hybrid systems, Theor. Comput. Sci. 342 (2–3) (2005) 229–261.
[42] T. Stauner, Systematic development of hybrid systems, Ph.D. thesis, TU München, 2001.