Bialgebraic Modelling of Timed Processes

Marco Kick*

LFCS, University of Edinburgh, Edinburgh EH9 3JZ, Scotland
Email: mk@dcs.ed.ac.uk

Abstract. We give an abstract axiomatic account of timed processes using monoids and their (partial and total) actions. Subsequently, we present categorical formulations thereof, including a novel characterisation of partial monoid actions as coalgebras for an *evolution comonad*. Adapting the approach of Turi and Plotkin [24], we then exhibit an abstract theory of well-behaved operational rules suitable for timed processes and, for discrete time, also derive a concrete syntactic format encompassing all rules we found in the literature.

Introduction

Over the past decade, much research effort has been directed towards establishing a theory of real-time systems. Amongst other approaches, extensions of standard process algebras [6, 10, 16] with timing features have been considered. The meaning of such *timed processes* is usually given by *structural operational semantics (SOS)* [20]: one inductively defines a *labelled transition system (LTS)* [20] on the set of processes, describing the behaviour of programs as (sequences of) transitions inferred from syntax-directed rules.

In this paper, we attempt to unify the plethora of studied languages, e.g., [4, 17, 19, 22, 25], by axiomatising shared fundamental mathematical properties; we model timed processes by *timed transition systems (TTSs)* – specific LTSs incorporating intuitive properties of time passing. In particular, we adopt the widely accepted design principle of separating computation from time passing, using *action* and *time* transitions to denote, respectively, instantaneous computations and pure time passing. We only consider the case of action and time transitions defined independently of each other, with Temporal CCS (TeCCS) [17] as the paradigmatic example.

In the study of processes, strong emphasis has been put on *behavioural equivalences* (see [8] for an overview) and on proving *congruence results* for them. This has led to *rule formats*: syntactic constraints on SOS rules whose satisfaction automatically establishes a congruence result. Arguably, the most well-known format is GSOS [7] which ensures that strong bisimulation [16] is a congruence. To our knowledge, there is no corresponding format for timed processes. Paving the way for such a format, we present a mathematical theory of well-behaved SOS rules for timed processes; furthermore, for the case of *discrete* time, we derive a syntactic format which, we believe, captures all examples in the literature.

* This work is supported by EPSRC grant GR/M56333.

To achieve these last two goals, our approach builds on the framework of [24]. There, *abstract operational rules* are a suitable kind of natural transformation, parametric in functorial notions Σ and B of *signature* and (one-step) *behaviour*, specifying how to derive the meaning of compound expressions from the meaning of their arguments (as traditional SOS rules do). Mathematically, the core of the theory is that such abstract rules inductively define a *distributive law* of a monad over a comonad [21] (respectively (co)freely generated from Σ and B). Using the *bialgebras* [24] of such distributive law, and under a mild condition on B (preservation of weak pullbacks), a congruence result for coalgebraic B-bisimulation [1] follows.

GSOS arises by considering abstract rules for the behaviour functor

$$B_A X = \mathcal{P}_{\mathrm{fi}}(X)^{\mathcal{A}} \tag{1}$$

on **Set** (see [24] for definitions): B_A-coalgebras are (image finite) LTSs, the corresponding coalgebraic bisimulation is strong bisimulation, and B_A preserves weak pullbacks. Hence, [24] yields, apart from a new proof, a conceptual reason *why* bisimulation is a congruence for GSOS languages. We want to apply these techniques to our model of timed processes, i.e., TTSs, by fitting it into the general framework; as we will see, to successfully do so we need to slightly generalise [24].

The structure of the paper is as follows. In §1, stressing the crucial concepts, we consider *time domains* (special monoids) and TTSs to model time and timed processes. We then show that TTSs are partial monoid actions of a time domain; total monoid actions play a prominent role as *delay operators*; both kinds of actions are then suitably combined in the new notion of *biaction*. In preparation for applying [24], we give categorical characterisations of these notions in §2, in particular of partial monoid actions (i.e., TTSs) as coalgebras of a novel *evolution comonad*, and of biactions as bialgebras for a distributive law (of the monad for total actions over the evolution comonad).

Finally, in §3, we adapt the framework of [24] to accommodate our categorical description of timed processes: rather than using behaviour functors (as in [24]), we need to begin with a behaviour *comonad* to account for TTSs (as coalgebras for the evolution comonad). Thus, a correspondingly different type of natural transformation modelling abstract rules must be used: still inducing a distributive law, yet removing the restriction to cofreely generated comonads. In some sense, behaviour comonads model 'big-step' semantics as they usually describe 'complete' computations, rather than just one step. For the case of discrete time, where the evolution comonad is actually cofreely generated from a functor, thus enabling us to directly apply [24], we then derive the aforementioned syntactic format.

Benefits from the approach of the present paper are: a more unified view of timed processes by isolating fundamental mathematical properties underlying many calculi; highlighting important mathematical structures in languages for timed processes; and providing another case study documenting the universality and flexibility of the categorical approach to operational semantics of [24].

1 A Mathematical Model of Timed Processes

In this section we present our abstract model of timed processes based on an axiomatic treatment of time transitions: time is modelled by special monoids, timed processes by LTSs with their transition relations restricted to incorporate special properties of time passing[1]. The following definitions of time domain and TTS are synthesised from the (slightly differing) ones in [12, 13, 18, 25].

Definition 1. *A* time domain *is a commutative monoid* $(\mathcal{T}, +, 0)$ *(abbreviated as* $\mathcal{T}$*) whose induced preorder* $\leq$*, defined by* $s \leq t \Leftrightarrow (\exists u).\ s + u = t$*, is a linear order, and which satisfies the* cancellation rule

$$t + u = t + v \Rightarrow u = v \tag{2}$$

Note that 0 is the least element with respect to $\leq$, $+$ is monotone with respect to $\leq$ (implied by (2)), and whenever $s \leq t$, there is a unique u such that $s + u = t$, written as $t - s$. This *partial subtraction* (only defined for $s \leq t$) can be extended to the total *truncated subtraction* $\dot{-}$, setting $t \dot{-} s = 0$ if $s > t$, making time domains *closed* [15]. Examples of time domains include the naturals $\mathbb{N}$ for *discrete time*, the non-negative reals $\mathbb{R}_{\geq 0}$ for *continuous time*, and the trivial domain $\{0\}$; note that $\mathcal{T}$ is either trivial or infinite (cf. [13]).

Definition 2. *A* TTS *is an LTS* $(P, \mathcal{T}, \rightsquigarrow)$ *where* P *is a set of* processes*,* $\mathcal{T}$ *is a time domain, and the* transition relation $\rightsquigarrow \subseteq P \times \mathcal{T} \times P$ *satisfies the following axioms, writing* $p \overset{t}{\rightsquigarrow} p'$ *for* $(p, t, p') \in \rightsquigarrow$*:*

(Determinacy) $\quad p \overset{t}{\rightsquigarrow} p' \wedge p \overset{t}{\rightsquigarrow} p'' \Rightarrow p' = p''$

(Zero-Delay) $\quad p \overset{0}{\rightsquigarrow} p$

(Continuity) $\quad p \overset{s+t}{\rightsquigarrow} p' \Leftrightarrow (\exists p'').\ p \overset{s}{\rightsquigarrow} p'' \overset{t}{\rightsquigarrow} p'$

Axiom (Determinacy) states that no choices are resolved by the passage of time. Axioms (Determinacy) and (Continuity) (*additivity* in [18]) are widely accepted, e.g., in [17, 18, 25] (sometimes only one implication of (Continuity) is used, e.g. in [12, 13]). Axiom (Zero-Delay) is usually not assumed since most languages allow only non-zero time transitions; yet it is intuitively clear and can be added without inconsistencies.

Definition 3. *Given TTSs* $(P_i, \mathcal{T}, \rightsquigarrow_i)$*,* $i \in \{1, 2\}$*, a relation* $R \subseteq P_1 \times P_2$ *is a* (strong) time bisimulation (over $\mathcal{T}$) *if* $(p_1, p_2) \in R$ *implies for all* $t \in \mathcal{T}$ *that*

$$p_1 \overset{t}{\rightsquigarrow}_1 p_1' \Rightarrow (\exists p_2').\ p_2 \overset{t}{\rightsquigarrow}_2 p_2' \wedge (p_1', p_2') \in R$$
$$p_2 \overset{t}{\rightsquigarrow}_2 p_2' \Rightarrow (\exists p_1').\ p_1 \overset{t}{\rightsquigarrow}_1 p_1' \wedge (p_1', p_2') \in R$$

We write $p_1 \sim p_2$ *if there exists a strong time bisimulation containing* (p_1, p_2)*.*

[1] Since action transitions are defined independently by GSOS rules, we omit their treatment, as it is just an instance of [24], using the behaviour functor B_A.

Note that calculi like TeCCS [17] combine time bisimulation with standard action bisimulation. Axiom (Determinacy) forces the transition relation $\rightsquigarrow$ of a TTS $(P, \mathcal{T}, \rightsquigarrow)$ to be a binary partial function $* : P \times \mathcal{T} \rightharpoonup P$ (written in infix notation), (Zero-Delay) and (Continuity) ensure the following equations up to Kleene equality ($\simeq$):

$$p * 0 \simeq p \qquad p * (s + t) \simeq (p * s) * t \tag{3}$$

In other words, $*$ has to be a *partial monoid action* of $\mathcal{T}$ on P. Note that the TTS is completely determined by $*$:

$$p \overset{t}{\rightsquigarrow} p' \Leftrightarrow p' \simeq p * t \tag{4}$$

Definition 4. *Let $\mathcal{T}$ be a time domain and $*_i$ partial $\mathcal{T}$-actions on P_i, $i \in \{1, 2\}$. A* homomorphism from $*_1$ to $*_2$ *is a (total) function $f : P_1 \to P_2$ such that for $p \in P_1$ and $t \in \mathcal{T}$, $f(p *_1 t) \simeq (fp) *_2 t$.*

Using (4), $R \subseteq P_1 \times P_2$ is a time bisimulation if $(p_1, p_2) \in R$ implies for all $t \in \mathcal{T}$, with $(\dots)\downarrow$ denoting that the expression $(\dots)$ is defined (dually, undefinedness will be expressed as $(\dots)\uparrow$):

$$(p_1 *_1 t)\downarrow \Leftrightarrow (p_2 *_2 t)\downarrow \qquad (p_1 *_1 t)\downarrow \Rightarrow (p_1 *_1 t, p_2 *_2 t) \in R$$

Thus, R is *closed* under partial actions. Note how the rewritten definition incorporates (Determinacy): $p_1 \overset{t}{\rightsquigarrow}_1 p_1'$ can only hold for $p_1' = p_1 *_1 t$, hence existential quantification is no longer required. Furthermore, a time bisimulation R can be endowed with a canonical partial action, viz., $(p_1, p_2) * t \simeq (p_1 *_1 t, p_2 *_2 t)$, making the *projections* $\pi_i : R \to P_i$ homomorphisms from $*$ to $*_i$.

Consider TeCCS now; after adding transitions according to (Zero-Delay)[2], its operational rules define a TTS on the set $\text{TeCCS}_{/\sim}$ of processes modulo time bisimulation. An important operator is *time prefixing*: intuitively, $(t).p$ represents the process p *delayed* by $t > 0$ units of time; formally, its behaviour is given by the following SOS rules[3]:

$$\frac{}{(s+t).p \overset{s}{\rightsquigarrow} (t).p}\ (t > 0) \qquad \frac{}{(t).p \overset{t}{\rightsquigarrow} p} \qquad \frac{p \overset{s}{\rightsquigarrow} p'}{(t).p \overset{s+t}{\rightsquigarrow} p'} \tag{5}$$

Extending for $t = 0$, (Zero-Delay) forces $(0).p = p$ since $(0).p \overset{0}{\rightsquigarrow} p$ is derivable from the rules, hence $(0).p \sim p$; one also easily shows $(s).(t).p \sim (s + t).p$. These equations, plus the delay intuition, inspire the following definition:

Definition 5. *Let $(P, \mathcal{T}, \rightsquigarrow)$ be a TTS; a* delay operator *on it is a binary total function $\bullet : \mathcal{T} \times P \to P$ (written in infix notation) satisfying the equations*

$$0 \bullet p = p \qquad s \bullet (t \bullet p) = (s + t) \bullet p \tag{6}$$

[2] TeCCS does not define $\overset{0}{\rightsquigarrow}$-transitions.

[3] The side condition $t > 0$ was implicit in [17] since $(0).p$ is not a TeCCS process.

Hence, a delay operator is a *total monoid action* of $\mathcal{T}$ on P; for example, the extended time prefixing is a delay operator on $\mathrm{TeCCS}_{/\sim}$. Analysing the interaction of time transitions with time prefixing or, more abstractly, of a partial action with a total one, bearing in mind that $\leq$ is a total order so that at least one of $s \dot{-} t$ and $t \dot{-} s$ is equal to 0, we introduce the following notion.

Definition 6. *A* $\mathcal{T}$-biaction *is a set* P *with a partial action* $*$ *and a delay operator* $\bullet$ *satisfying*

$$(t \bullet p) * s \simeq (t \dot{-} s) \bullet (p * (s \dot{-} t)) \tag{7}$$

A $\mathcal{T}$-biaction gives a minimal account of timed processes which can be delayed and perform time transitions, (7) linking these actions in the 'right' way. From our previous remarks, it follows that $\mathrm{TeCCS}_{/\sim}$ is an example of a biaction.

2 Timed Processes Categorically

In order to fit the previous considerations into the framework of [24], we now give categorical formulations of (total and partial) actions and biactions; in the remainder of this section, let $\mathcal{T}$ be a time domain. For partial actions, driven by their equivalence with TTSs, and the necessity to account for time bisimulation, we will present a coalgebraic description; for dual reasons, we use the following folklore description of total $\mathcal{T}$-actions as algebras for a monad:

Proposition 1. *The map* $X \mapsto \mathcal{T} \times X$ *induces a monad on* **Set** *whose algebras are the* $\mathcal{T}$*-actions.*

Instead of viewing partial actions as partial functions $X \times \mathcal{T} \rightharpoonup X$, we will regard them as *total* functions $X \to E_{\mathcal{T}}X$ where $E_{\mathcal{T}}X$ contains $\mathcal{T}$*-evolutions* – certain partial functions $\mathcal{T} \rightharpoonup X$ with properties mimicking (3):

Definition 7. *A* $\mathcal{T}$-evolution (on X) *is a partial function* $e : \mathcal{T} \rightharpoonup X$ *such that*

$$e(0)\downarrow \qquad\qquad e(s+t)\downarrow \Rightarrow e(s)\downarrow \tag{8}$$

We denote the set of all $\mathcal{T}$*-evolutions on* X *by* $E_{\mathcal{T}}X$ *(or simply* EX*).*

Intuitively, an evolution e in EX describes a timed process whose time transitions are defined by $e \overset{t}{\rightsquigarrow} x \overset{\mathsf{df}}{\Leftrightarrow} e(t) \simeq x$; using this notation, and abbreviating $(\exists x).\ e \overset{t}{\rightsquigarrow} x$ by $e \overset{t}{\rightsquigarrow}$, the two axioms (8) become $e \overset{0}{\rightsquigarrow}$ and $e \overset{s+t}{\rightsquigarrow} \Rightarrow\ e \overset{s}{\rightsquigarrow}$, i.e., very basic versions of (Zero-Delay) and (one direction of) (Continuity).

Given a function $f : X \to Y$, defining $Ef : EX \to EY$ by $e \mapsto f \circ e$ makes E an endofunctor on **Set**, as is routinely verified. Note that, since f is a total map, the domains of e and $f \circ e$ are equal; E is not only a functor:

Proposition 2. *E is a comonad, with counit ε and comultiplication δ given by*

$$\varepsilon_X : EX \to X,\ e \mapsto e(0)$$

$$\delta_X : EX \to E^2X,\ e \mapsto \begin{cases} \left(\lambda t. \begin{cases} e(s+t) & \text{if } e(s+t)\downarrow \\ \text{undef} & \text{if } e(s+t)\uparrow \end{cases}\right) & \text{if } e(s)\downarrow \\ \text{undef} & \text{if } e(s)\uparrow \end{cases}$$

Definition 8. *We call* (E, ε, δ) *the* evolution comonad *on* **Set**.

Note that δ transforms an evolution e on X into an evolution $\delta(e)$ on EX by acting as a *parameterised shift* or *lookahead*: $\delta(e)(t)$ is equal to $e + t$, i.e., the evolution e after $t \in \mathcal{T}$ units of time have passed, the case distinction merely taking care of potential undefinedness. Furthermore, the comonad law $\varepsilon_E \circ \delta = id_E$ states $\delta(e)(0) = e + 0 = e$, i.e., shifting by 0 is the same as not shifting at all.

Standard (image finite) LTSs are coalgebras for the behaviour functor B_A; in contrast, TTSs are partial $\mathcal{T}$-actions and need the notion of coalgebras for a comonad to account for the axioms in (3):

Proposition 3. *E-coalgebras are partial $\mathcal{T}$-actions, which in turn are TTSs; the passage from an E-coalgebra $k : P \to EP$ to a TTS $(P, \mathcal{T}, \rightsquigarrow)$, and vice versa, is given by*

$$p \overset{t}{\rightsquigarrow} p' \;\Leftrightarrow\; k(p)(t) \simeq p' \tag{9}$$

A comonad D is *cofreely generated* by an endofunctor B if each DX is the carrier of the final $(X \times B)$-coalgebra (see e.g. [23]). One important consequence is then that D–$\mathsf{Coalg} \cong B$–Coalg; this can be interpreted as big-step transitions (from D-coalgebras) being completely determined by one-step transitions (from B-coalgebras). For the evolution comonad and discrete time, we note the following well-known fact:

Proposition 4. *$E_{\mathbb{N}}$ is cofreely generated by $B_{\mathbb{N}} \overset{\mathsf{df}}{=} 1 + Id : \mathbf{Set} \to \mathbf{Set}$.*

Hence $E_{\mathbb{N}}$–$\mathsf{Coalg} \cong B_{\mathbb{N}}$–$\mathsf{Coalg}$, i.e., any partial $\mathbb{N}$-action on some processes P corresponds to a unique function $P \to 1 + P$: for $t \in \mathbb{N}$ we have $t = 1 + \ldots + 1$ (t times); hence, knowing the 'next step' (via a $B_{\mathbb{N}}$-coalgebra) is by (Continuity) sufficient to define all time transitions (via an $E_{\mathbb{N}}$-coalgebra). We can use $B_{\mathbb{N}}$ to model the *qualitative* notions of time in [9, 19]: a special deterministic action (χ or σ) denoting the passage of an (unspecified) amount of time (which might be thought of as the duration of a clock cycle).

For a behaviour functor B, there is a notion of *coalgebraic B-bisimulation* [1]: a *B-bisimulation* between B-coalgebras $k_1 : X_1 \to BX_1$ and $k_2 : X_2 \to BX_2$ is a B-coalgebra $k : X \to BX$ and B-coalgebra homomorphisms $f_i : X \to X_i$. For example, for B_A, one obtains strong bisimulation in this way. This definition readily generalises to comonads, yielding for the evolution comonad, via the equivalence of TTSs and partial monoid actions:

Proposition 5. *E-bisimulation is strong time bisimulation.*

For $\mathcal{T} = \mathbb{N}$, we obtain strong time bisimulation over $\mathbb{N}$; yet, such a bisimulation is completely determined by matching the next step, which is exactly given by $B_{\mathbb{N}}$-bisimulation, and which is also the appropriate notion of (time) bisimulation for qualitative time. The following proposition will be needed later.

Proposition 6. *E preserves pullbacks.*

Finally, we can now give a categorical description of the $\mathcal{T}$-biactions introduced above. A *distributive law of a monad* (T, η, μ) *over a comonad* (D, ε, δ) [21] is a natural transformation $\ell : TD \Rightarrow DT$ subject to the equations

$$\ell \circ \eta_D = D\eta \qquad\qquad \ell \circ \mu_D = D\mu \circ \ell_T \circ T\ell$$

and their duals

$$\varepsilon_T \circ \ell = T\varepsilon \qquad\qquad \delta_T \circ \ell = D\ell \circ \ell_D \circ T\delta$$

The ℓ-bialgebras [24] are then structures $TX \xrightarrow{h} X \xrightarrow{k} DX$, consisting of a T-algebra h and a D-coalgebra k such that $k \circ h = Dh \circ \ell \circ Tk$.

Proposition 7. *The map*

$$\ell_X : \mathcal{T} \times EX \to E(\mathcal{T} \times X), \quad (t, e) \mapsto \lambda s.(t \dot{-} s, e(s \dot{-} t)) \tag{10}$$

induces a distributive law whose bialgebras are biactions.

Hence, biactions are obtained as bialgebras, distributing total over partial actions. As total $\mathcal{T}$-actions can alternatively be described as coalgebras for the exponential comonad $(\_)^{\mathcal{T}}$ (dually to Prop. 1), an interesting open question is whether there is an entirely coalgebraic description of biactions, using a distributive law of comonads in order to combine $(\_)^{\mathcal{T}}$ with E.

3 Categorical Operational Semantics for Timed Processes

We now turn our attention to abstract operational rules for timed processes or, more abstractly, behaviour comonads. In [24], *abstract GSOS rules* were given by a natural transformation of type

$$\Sigma(Id \times B) \Rightarrow BT \tag{11}$$

for functorial notions Σ and B of signature and (local) behaviour, with Σ freely generating the syntax monad T. Such rules then inductively induce a distributive law $TD \Rightarrow DT$ of T over the comonad D cofreely generated by B, hence distributing free syntax over cofree behaviour.

For B_A, we can translate (11) into concrete rules as follows: for each set X of variables and each n-ary operator σ in the signature Σ, there is a map

$$[\![\sigma]\!] : (X \times \mathcal{P}_{\text{fi}}(X)^{\mathcal{A}})^n \to \mathcal{P}_{\text{fi}}(TX)^{\mathcal{A}}$$

Its arguments are n pairs of variables $x_i \in X$ and 'behaviours' $\beta_i \in \mathcal{P}_{\text{fi}}(X)^{\mathcal{A}}$, interpreted as process names and transitions (described by mapping labels to targets of transitions with that label); its result is a behaviour encoding the transitions of the compound process $\sigma(x_1, \ldots, x_n)$. Careful analysis, in particular of naturality, shows that this is actually equivalent to defining GSOS rules.

The Id-component in the premises of (11) is needed to associate names to behaviours; for E, with its counit $\varepsilon : E \Rightarrow Id$ available for that purpose, it suffices to use ΣE instead of $\Sigma(Id \times E)$. Inspection of rules in the literature allows even further simplification, as shown by the following proposition.

Proposition 8. *The rules of TeCCS induce a natural transformation of type*

$$\Sigma E \Rightarrow E(Id + \Sigma) \tag{12}$$

Proof. We will only illustrate how to define some rules, leaving the proof of naturality to the reader. Consider weak choice $\oplus$ *with its (slightly adapted) rules*

$$\frac{p_1 \overset{t}{\rightsquigarrow} p_1',\ p_2 \overset{t}{\rightsquigarrow} p_2'}{p_1 \oplus p_2 \overset{t}{\rightsquigarrow} p_1' \oplus p_2'} \qquad \frac{p_1 \overset{t}{\rightsquigarrow} p_1',\ p_2 \overset{t}{\not\rightsquigarrow}}{p_1 \oplus p_2 \overset{t}{\rightsquigarrow} p_1'} \qquad \frac{p_1 \overset{t}{\not\rightsquigarrow},\ p_2 \overset{t}{\rightsquigarrow} p_2'}{p_1 \oplus p_2 \overset{t}{\rightsquigarrow} p_2'}$$

Recalling (9)*, in particular* $p \overset{t}{\not\rightsquigarrow}$ *being equivalent to* $k(p)(t)\uparrow$*, we obtain the map* $[\![\oplus]\!] : (EX)^2 \to E(X + \Sigma X)$*,*

$$[\![e_1 \oplus e_2]\!] = \lambda t. \begin{cases} e_1(t) \oplus e_2(t) & \text{if } e_1(t)\downarrow \wedge\ e_2(t)\downarrow \\ e_1(t) & \text{if } e_1(t)\downarrow \wedge\ e_2(t)\uparrow \\ e_2(t) & \text{if } e_1(t)\uparrow \wedge\ e_2(t)\downarrow \\ \text{undef} & \text{if } e_1(t)\uparrow \wedge\ e_2(t)\uparrow \end{cases}$$

Note how the cases match the rules. Writing $\mathcal{T}^+ \overset{\mathsf{df}}{=} \mathcal{T} \setminus \{0\}$*, time prefixing, with rules given in* (5)*, yields the map* $[\![(\_).\_]\!] : \mathcal{T}^+ \times EX \to E(X + \Sigma X)$*, recalling* $\varepsilon(e) = e(0)$*:*

$$[\![(t).e]\!] = \lambda s. \begin{cases} (t - s).\varepsilon(e) & \text{if } s < t \\ \varepsilon(e) & \text{if } s = t \\ e(s - t) & \text{if } s > t\ \wedge\ e(s-t)\downarrow \\ \text{undef} & \text{if } s > t\ \wedge\ e(s-t)\uparrow \end{cases}$$

More precisely, $[\![(\_).\_]\!] : \mathcal{T}^+ \times EX \to E(X + (\mathcal{T}^+ \times X)) \cong E(\mathcal{T} \times X)$*, since* $X + (\mathcal{T}^+ \times X) \cong (X \times \{0\}) + (X \times \mathcal{T}^+) \cong X \times (\{0\} + \mathcal{T}^+) \cong X \times \mathcal{T}$*, and also* $[\![(\_).\_]\!] = \ell|_{(\mathcal{T}^+ \times EX)}$*, i.e.,* (10) *restricted to arguments of type* $\mathcal{T}^+ \times EX$*. Note that this definition also makes sense for* $t = 0$*, yielding* $[\![(0).e]\!] = e$ *since* $s < 0$ *never holds, hence resulting in* $[\![(\_).\_]\!] = \ell$*.*

We will now show how to obtain a distributive law $TE \Rightarrow ET$ from (12). Since E is no longer cofreely generated from a behaviour functor, we need to place some conditions on the rules relating them to the operations ε and δ of E (which were void in the cofree case). Despite stating the following theorem for E, it holds for arbitrary comonads (D, ε, δ).

Theorem 1. *Let* Σ *be a functor freely generating a monad* T*; furthermore suppose* ρ *is a natural transformation of type* (12) *satisfying*

$$\begin{array}{ccc} \Sigma E & \overset{\rho}{\Longrightarrow} & E(Id + \Sigma) \\ {\scriptstyle \Sigma\epsilon}\downarrow & & \downarrow{\scriptstyle \epsilon_{Id+\Sigma}} \\ \Sigma & \underset{inr}{\Longrightarrow} & Id + \Sigma \end{array} \qquad \begin{array}{ccccc} \Sigma E & & \overset{\rho}{\Longrightarrow} & & E(Id + \Sigma) \\ {\scriptstyle \Sigma\delta}\downarrow & & & & \downarrow{\scriptstyle \delta_{Id+\Sigma}} \\ \Sigma E^2 & \underset{\rho_E}{\Longrightarrow} & E(E + \Sigma E) & \underset{E[Einl,\rho]}{\Longrightarrow} & E^2(Id + \Sigma) \end{array} \tag{13}$$

Then this induces a distributive law $TE \Rightarrow ET$*.*

Note that (12) is the simplest case in a hierarchy of natural transformations describing operational rules for behaviour comonads, distinguished by the complexity of terms allowed in the right-hand side of rule conclusions; the other extreme is $\Sigma E \Rightarrow ET$, permitting arbitrary terms, as opposed to (12) allowing at most one function symbol. Increasing expressivity is traded off against the complexity of constraints to be satisfied in order to respect the operations of the comonad, with (13) also being the simplest case. For TeCCS, we obtain:

Proposition 9. *The natural transformation of Prop. 8 satisfies* (13)*.*

Corollary 1. *Time bisimulation is a congruence for TeCCS.*

Proof. This follows from Thm. 1, Props. 5, 6, and 9, and [24, Cor. 7.5].

Note that this is a weaker congruence result than what was shown in [17]: there, the bisimulation obtained by *combining* action and time bisimulation was a congruence; here, we only obtain that each of the two on its own is a congruence (by [24], since the action rules are GSOS, and by the preceding corollary).

Discrete Time. Let us now consider the case $\mathcal{T} = \mathbb{N}$. As shown in Prop. (4), $E_{\mathbb{N}}$ is cofreely generated by the functor $B_{\mathbb{N}}$, enabling us to use (11) to define the rules and to introduce the following syntactic rule format. Let Σ be a signature and fix an enumeration (without repetitions) $\{x_k \mid k \geq 1\}$ of a countable set X of variables. Our rule format will use GSOS [7] rules of the form

$$\frac{\{x_i \rightsquigarrow x_{n+i}\}_{i \in I},\ \{x_j \not\rightsquigarrow\}_{j \in J}}{\sigma(x_1, \ldots, x_n) \rightsquigarrow \theta} \tag{14}$$

where $\sigma \in \Sigma$ is an n-ary function symbol, $I, J \subseteq \{1, \ldots, n\}$, and θ a term over Σ and X. The fact that we consider GSOS rules means that θ contains no fresh variables, i.e., all variables in θ must occur somewhere else in the rule[4]. Moreover, for each x_k, $1 \leq k \leq n$, there is at most one positive and one negative premise, and the targets of the positive premises (if there are any) are fixed to be the next $|I|$ variables after x_n in the order of the enumeration.

A rule (14) is *consistent* if $I \cap J = \emptyset$, *complete* if $I \cup J = \{1, \ldots, n\}$, and it has *type* (I, J); two such rules, with respective types (I_k, J_k), are *mutually exclusive* if $(I_1 \cap J_2) \cup (I_2 \cap J_1) \neq \emptyset$, i.e., there is at least one variable occurring both positively (i.e., in a positive premise) in one rule and negatively in the other; for consistent rules, note that this is equivalent to $I_1 \neq I_2$ (since $J_k = \{1, \ldots, n\} \setminus I_k$). Say then that a set of consistent and complete rules (14) is in *deterministic single-label GSOS (dslGSOS)* format over X if any two rules for the same operator $\sigma \in \Sigma$ are mutually exclusive.

Consistent rules have no conflicting premises, i.e., no variable occurs both positively and negatively in the same rule; yet, since we only use complete rules

[4] Also all variables in $\{x_k \mid 1 \leq k \leq n + |I|\}$ must be pairwise distinct, but that is already guaranteed by our use of the enumeration.

for the dslGSOS format, each x_k must occur inside the premises, therefore, it occurs in *exactly one* premise. Mutual exclusion then ensures for each operator that there is at most one rule applicable at a time (assuming each x_k has either no or exactly one next step, as defined by a $B_{\mathbb{N}}$-coalgebra). Note that there can only be finitely many rules for each operator $\sigma \in \Sigma$ without violating mutual exclusion; hence, if Σ is finite, each set of rules in dslGSOS format is automatically finite. Furthermore, mutual exclusion is a *global* condition on *sets* of rules, unlike the *local* variable conditions of the GSOS format, or consistency and completeness, which refer only to single rules; also the restriction to image-finite sets of GSOS rules in [24], ensuring the one-to-one correspondence between such sets of rules and natural transformations (11) for the functor B_A, is a weak but nevertheless global condition.

Theorem 2. *There is a one-to-one correspondence between rules in dslGSOS format and natural transformations of type*

$$\Sigma(Id \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T$$

Proof. The proof is done along the lines of the proof of [24, Thm 1.1]. We obtain an exact correspondence (not just up to equivalence of sets of rules as in [24]) since the dslGSOS format, deploying the variable enumeration, unambiguously prescribes which variables occur at which places in the rules.

An example of rules in the above format is given by the time rules of the language ATP [19]. We have also defined a new one-step version of the time rules of TeCCS fitting in dslGSOS. Using the above theorem and the equivalence between $E_{\mathbb{N}}$- and $B_{\mathbb{N}}$-coalgebras, we were then able to prove categorically that the one-step rules induce the same TTS as the original (big-step) rules.

Note that for both the above languages, the action rules are GSOS, hence we can describe them using the behaviour functor B_A and (11). For the full languages, combining the two sets of rules given by two natural transformations $\rho_A : \Sigma(Id \times B_A) \Rightarrow B_A T$ and $\rho_{\mathbb{N}} : \Sigma(Id \times B_{\mathbb{N}}) \Rightarrow B_{\mathbb{N}}T$, precomposing ρ_A and $\rho_{\mathbb{N}}$ with the respective projections yields natural transformations

$$(15) \qquad \Sigma(Id \times (B_A \times B_{\mathbb{N}})) \Rightarrow B_A T \qquad \Sigma(Id \times (B_A \times B_{\mathbb{N}})) \Rightarrow B_{\mathbb{N}}T$$

Since $(B_A \times B_{\mathbb{N}})T \cong B_A T \times B_{\mathbb{N}}T$, (15) is equivalent to (11) instantiated with $B = B_A \times B_{\mathbb{N}}$. Thus, by [24], the combined bisimulation is a congruence for both languages since it corresponds exactly to coalgebraic bisimulation for $B_A \times B_{\mathbb{N}}$.

Future Work

Most pressingly, we plan to give a syntactic characterisation of the abstract format (12) for timed processes for the general case: this is as yet beyond the scope of this paper but we envisage a format corresponding to (12) on which to impose (global) side conditions corresponding to (13).

For discrete time, (15) should also allow to treat such calculi where action and time transitions are no longer defined independently, in particular by adopting the *maximal progress assumption* [11], e.g., (discrete time) TiCCS [25] and TPL [9]. We want to extend this to the case of an arbitrary time domain, also to obtain a congruence result for the combined bisimulation for (big-step) TeCCS. Hopefully, we can achieve this by instantiating (12) with products of comonads (alike to using products of functors in the discrete case).

In a different direction, we will investigate the idea of *normal forms*: for instance, the normal forms of $(0).p$ and $(s).(t).p$ would be p and $(s+t).p$. In order to achieve this, we could use a retraction $T_{NF} \triangleleft T$ for two (syntax) monads, corresponding to rewriting terms into normal forms and the inclusion of normal forms into terms, coupled with a distributive law $T_{NF}E \Rightarrow ET_{NF}$ (induced by rules like (12)) to model the operational semantics of normal forms only; the retraction should then induce a distributive law for the full language.

The last point seems closely connected to defining rules corresponding to (12) in the category of $\mathcal{T}$-actions, using a more complex syntax with a 'built-in' total $\mathcal{T}$-action corresponding to time prefixing; as behaviour, we would use the *lifting* (see [24]) of E obtained by the distributive law (10). This might also clear up the somewhat mysterious rôle of said law, since currently it is only used in a restricted form (by the rules for time prefixing, see Prop. 8), although clearly an important mathematical structure.

Finally, we hope to be able to deal with operational semantics for timed automata [2], a very prominent approach to formalising real-time systems. Treating also this quite different (from the process-algebraic point of view) approach within the same or a similar framework as in the present paper would even further emphasise the flexibility of the categorical framework.

Acknowledgements. The author would like to thank Daniele Turi and Gordon Plotkin for numerous helpful discussions.

References

1. P. Aczel and P. F. Mendler. A final coalgebra theorem. In D. H. Pitt, D. E. Rydeheard, P. Dybjer, A. M. Pitts, and A. Poigné, editors, *Category Theory and Computer Science*, *LNCS* 389, pp. 357–365, 1989. Springer.
2. R. Alur and D. L. Dill A theory of timed automata. *Theoretical Computer Science* **25(2)**, pp. 183–235, 1994.
3. J.C.M. Baeten and J.W. Klop, editors. *Concurrency Theory (CONCUR '90)*, *LNCS* 458, 1990. Springer.
4. J.C.M. Baeten and C.A. Middelburg. Process algebra with timing: real time and discrete time. In Bergstra et al. [5], chapter 10.
5. J. A. Bergstra, A. Ponse, and S. A. Smolka, editors. *Handbook of Process Algebra*. North-Holland, 2001.
6. J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Computation* **60**, pp. 109–137, 1984.
7. B. Bloom, S. Istrail, and A. R. Meyer. Bisimulation can't be traced. *Journal of the ACM* **42(1)**, pp. 232–268, 1995.

8. R.J. van Glabbeek. The linear time – branching time spectrum I; the semantics of concrete, sequential processes. In Bergstra et al. [5], chapter 1, pages 3–99.
9. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation* **117**, pp. 221–239, 1995.
10. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
11. J.J.M. Hooman and W.P. de Roever. Design and verification in real-time distributed computing: an introduction to compositional methods. In E. Brinksma, G. Scollo, and Chris A. Vissers, editors. *International Conference on Protocol Specification, Testing and Verification*, 1989. North-Holland.
12. A. Jeffrey. A linear time process algebra. In Larsen and Skou [14], pp. 432–442.
13. A. S. A. Jeffrey, S. A. Schneider, and F. W. Vaandrager. A comparison of additivity axioms in timed transition systems. Technical Report CS-R9366, CWI, 1993.
14. K.G. Larsen and A. Skou, editors. *Computer Aided Verification (CAV '91)*, *LNCS* 575, 1991. Springer.
15. W. Lawvere. Metric spaces, generalized logic, and closed categories. In *Rendiconti del Seminario Matematico e Fisico di Milano, XLIII*. Tipografia Fusi, 1973.
16. R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.
17. F. Moller and C. Tofts. A temporal calculus of communicating systems. In Baeten and Klop [3], pp. 401–415.
18. X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In Larsen and Skou [14], pp. 376–398.
19. X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation* **114**, pp. 131–178, 1994.
20. G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.
21. J. Power and H. Watanabe. Distributivity for a monad and a comonad. In B. Jacobs and J. Rutten, editors, *Second Workshop on Coalgebraic Methods in Computer Science (CMCS'1999)*, volume 19 of *ENTCS*, 1999.
22. S. A. Schneider. An operational semantics for timed CSP. Technical Report PRG-TR-1-91, Oxford University, 1991.
23. D. Turi. *Functorial Operational Semantics and its Denotational Dual*. PhD thesis, Free University, Amsterdam, 1996.
24. D. Turi and G. Plotkin. Towards a mathematical operational semantics. In *Twelfth Annual Symposium on Logic in Computer Science (LICS '97)*, pp. 280–291, 1997. IEEE Computer Society Press.
25. Y. Wang. Real-time behaviour of asynchronous agents. In Baeten and Klop [3], pp. 502–520.