Electronic Notes in Theoretical Computer Science 44 No. 1 (2001)
URL: `http://www.elsevier.nl/locate/entcs/volume44.html` 20 pages

Process Calculi *à la* Bird-Meertens

Luís S. Barbosa [1]

Departamento de Informática
Universidade do Minho
Braga, Portugal

Abstract

This paper is an attempt to apply the reasoning principles and calculational style underlying the so-called *Bird-Meertens formalism* to the design of process calculi, parametrized by a behaviour model. In particular, basically equational and point-free proofs of process properties are given, relying on the universal characterisation of anamorphisms and therefore avoiding the explicit construction of bisimulations. The developed calculi can be directly implemented on a functional language supporting coinductive types, which provides a convenient way to prototype processes and assess alternative design decisions.

1 Introduction

It is well known that *initial* algebras and *final* coalgebras provide abstract descriptions of a variety of phenomena in programming, in particular of *data* and *behavioural* structures, respectively. Both initiality and finality, as universal properties, entail definitional and proof principles, *i.e.*, a basis for the development of program calculi directly based on (actually driven by) type specifications. Moreover, such properties can be turned into programming *combinators* and used, not only to calculate programs, but also to program with. In functional programming the role of such universals — combined with the 'calculational' style entailed by category theory — has been fundamental to a whole discipline of algorithm derivation and transformation. This can be traced back to the so-called *Bird-Meertens formalism* [5,6] and the foundational work of T. Hagino [8]. Since then, the area has known a remarkable progress, as witnessed by the vast bibliography published both on theory and applications — see [12,13,4], among many others references.

This paper reports on an attempt to apply the same reasoning principles and calculational style to the design of process calculi, relying on the representation of processes as inhabitants of coinductive types, *i.e.*, final coalgebras for

[1] Email: `lsb@di.uminho.pt`

suitable Set endofunctors. Final semantics for processes is an active research area, namely after Aczel's landmark paper [1]. Our emphasis is, however, actually placed on the design side: we intend to show how process calculi can be developed and their laws proved along the lines one gets used to in (data-oriented) program calculi. Although only the 'fine grain' observational equivalence entailed by (strict) bisimulation is considered in the sequel, we believe that the proposed approach has a number of advantages:

- First of all it provides a *uniform* treatment of processes and other computational structures, *e.g.*, data structures, both represented as categorical types for functors capturing signatures of, respectively, observers and constructors. Placing data and behaviour at a similar level conveys the idea that process models can be chosen and specified according to a given application area, in the same way that a suitable data structure is defined to meet a particular engineering problem. Moreover, processes and data become expressible in programming languages supporting categorical types, such as Charity [7], providing a convenient way to prototype processes and compare alternative design decisions. This builds on the author previous work on *component* algebras [3].
- Proofs are carried out in a purely calculational (basically *equational* and *pointfree*) style, therefore circumventing the explicit construction of bisimulations used in most of the literature on process calculi. In particular, a 'conditional fusion' result is proved to handle conditional laws.
- Finally the approach is independent of any particular process calculus and makes explicit the different ingredients present in the design of any such calculi. In particular structural aspects of the underlying *behaviour model* (*e.g.*, the dichotomies such as active *vs* reactive, deterministic *vs* non deterministic) become clearly separated from the *interaction* structure which defines the synchronisation discipline.

2 Preliminaries

Anamorphisms.

Technically, the approach sketched here amounts to the systematic use of the universal property of *anamorphisms*. Recall that, for a functor T, a T-*anamorphism* is the *unique* T-comorphism to the final coalgebra $\omega_{\mathsf{T}} : \nu_{\mathsf{T}} \longrightarrow \mathsf{T}\,\nu_{\mathsf{T}}$ from any other coalgebra $\langle U, p\rangle$. Also called the *coinductive extension* of p [17], it is written, in the tradition of [13], as $[\![(p)]\!]_{\mathsf{T}}$ or, simply, $[\![(p)]\!]$, and satisfies the following universal property:

$$k = [\![(p)]\!]_{\mathsf{T}} \quad \Leftrightarrow \quad \omega_{\mathsf{T}} \cdot k = \mathsf{T}\,k \cdot p \tag{1}$$

from which other laws are easily derived, *e.g.*:

$$\omega_{\mathsf{T}} \cdot [\![p]\!] = \mathsf{T}\, [\![p]\!] \cdot p \tag{2}$$

$$[\![\omega_{\mathsf{T}}]\!] = \mathsf{id}_{\nu_{\mathsf{T}}} \tag{3}$$

$$[\![p]\!] \cdot h = [\![q]\!] \quad \text{if} \quad p \cdot h = \mathsf{T}\, h \cdot q \tag{4}$$

In the context of the *Bird-Meertens formalism* equations (2) to (4) are seen as instances of a *cancellation*, *reflection* and *fusion* result, respectively.

Invariants.

In order to derive conditional laws (section 4) we shall resort to the notion of a *p-invariant*, *i.e.*, a predicate ϕ over the carrier of a T-coalgebra p closed under the p dynamics. Following [11], ϕ is an invariant if, as a set, it satisfies $\phi \subseteq \bigcirc_p \phi$, where $\bigcirc_p \phi$ denotes the set of all states whose immediate successors, under p, if any, satisfy ϕ. $\bigcirc_p$ is, in fact, a *modal* combinator which corresponds to the familiar (weak) *next* operator in modal logics [2]. [11] introduces a definition of $\bigcirc_p$ in terms of a lifting operation $(\ )^{\mathsf{T}} : \mathcal{P}X \longrightarrow \mathcal{P}\mathsf{T}\, X$ defined inductively on the structure of (extended polynomial) functors (see [9] for the complete picture). Formally, $\bigcirc_p \phi = \{u \in U |\ p\, u \in (\phi)^{\mathsf{T}}\}$. The infinite extension of $\bigcirc_p$ characterise the (future) 'box' operator relative to a coalgebra p. This is denoted in [11] by $\Box_p$, where $\Box_p$ is defined as the *greatest* fix point of $\lambda_x .\ \phi \cap \bigcirc_p x$. Informally, $\Box_p \phi$ reads 'ϕ holds now and in all successor states'.

3 Process Structure and Combinators.

Processes.

In designing a process calculus, its operational semantics is usually given in terms of a transition relation $\xrightarrow{a}$ over processes, indexed by a set *Act* of actions [3], witnessing the collection of actions in which a process gets committed and the resulting 'continuations', *i.e.*, the behaviours subsequently exhibited. A first basic design decision concerns the definition of what should be understood by such a collection. As a rule it is defined as a *set*, in order to express non determinism. Other, more restrictive, possibilities consider a sequence or even just a single continuation, modelling, respectively, 'ordered' non determinism or determinism. In general, this underlying behaviour model can be represented by a functor B.

[2] The transition system defined by p is the structure upon which the operator is interpreted. In fact, it has been recently recognised by a number of authors (notably in [15] and [11]) that a modal language associated to a T-coalgebra is determined by its *shape*, as recorded in T.

[3] This set will later be equipped with further structure to support particular interaction disciplines. For the moment just assume that actions are generated from a set L of *labels*, *i.e.*, a set of formal names. The embedding $L \hookrightarrow Act$ will usually be left implicit.

An orthogonal decision concerns the intended interpretation of the transition relation, which is usually left implicit or underspecified in process calculi. We may, however, distinguish between

- An 'active' interpretation, in which a transition $p \xrightarrow{a} q$ is informally characterised as 'p evolves to q by performing an action a', both q and a being solely determined by p.
- A 'reactive' interpretation, informally reading 'p reacts to an external stimulus a by evolving to q'.

Processes will then be taken as inhabitants of the carrier of the final coalgebra $\omega : \nu \longrightarrow \mathsf{T}\,\nu$, with T defined as $\mathsf{B}\ (\mathit{Act} \times \mathsf{Id})$, in the first case, and $(\mathsf{B}\ \mathsf{Id})^{\mathit{Act}}$, in the second. To illustrate the proposed approach to the development of process calculi, we shall focus on a particular case where B is the finite powerset functor and the 'active' interpretation is adopted. The transition relation, for this case, is given by $p \xrightarrow{a} q \ \ \mathsf{iff} \ \ \langle a, q \rangle \in \omega\, p$. Although this corresponds to the main trend in the literature, some alternatives will be considered in section 5.

The restriction to the finite powerset avoids cardinality problems and assures the existence of a final coalgebra for T. This means, of course, we shall deal only with *image-finite* processes, a not too severe restriction in practice which may be partially circumvented by a suitable definition of the structure of *Act* [4].

Dynamic Combinators.

The cornerstone in the design of a process calculi is the judicious selection of a (hopefully small) set of process combinators. In [14], R. Milner classifies them into two distinct groups. The first group consists of all combinators which persist through action, *i.e.*, which are present before and after a transition occurs. They are called *static* and used to set up process' architectures, specifying how their components are linked and which parts of their interface are public or private. *Dynamic* combinators, on the other hand, are 'consumed' on action occurrence, disappearing from the expression representing the process continuation. In this paragraph the usual Ccs dynamic combinators — *i.e.*, *inaction*, *prefix* and non-deterministic *choice* — are defined as operators on the final universe of processes considered above. Notice that, being non recursive, they have a direct (coinductive) definition which depends solely on the chosen process structure. Therefore, the inactive process is represented as a constant $\mathsf{nil} : \mathbf{1} \longrightarrow \nu$ upon which no relevant observation can be made. Prefix gives rise to an *Act*-indexed family of operators $a. : \nu \longrightarrow \nu$, with $a \in \mathit{Act}$. Finally, the possible actions of the non deterministic choice of two processes p and q corresponds to the collection of all actions allowed for

[4] For instance, by taking *Act* as *channel* names through which data flows, which corresponds closely to 'Ccs with value passing' [14]. Therefore, only the set of channels, and not the messages (seen as pairs channel/data), must remain finite.

p and q. Therefore, the operator $+ : \nu \times \nu \longrightarrow \nu$ can only be defined over a process structure in which observations form a collection. Formally,

$$\begin{array}{rrcl} \mathsf{inaction} & \omega \cdot \mathsf{nil} & = & \underline{\emptyset} \\ \mathsf{prefix} & \omega \cdot a. & = & \mathsf{sing} \cdot \mathsf{label}_a \\ \mathsf{choice} & \omega \cdot + & = & \cup \cdot (\omega \times \omega) \end{array}$$

where $\mathsf{sing} = \lambda x\,.\ \{x\}$ and $\mathsf{label}_a = \lambda x\,.\ \langle a, x\rangle$.

Clearly, structure $\langle \nu; +, \mathsf{nil}\rangle$ forms an Abelian idempotent monoid, a fact that can be proved by simple equational reasoning, resorting to the corresponding properties of set union. Moreover, finality turns ω into an isomorphism and therefore, to prove $e = e'$ it is enough to show that $\omega \cdot e = \omega \cdot e'$. To illustrate the proposed proof style, consider the proof of a particularly simple result: $+$ associativity, *i.e.*, $+ \cdot (+ \times \mathsf{id}) = + \cdot (\mathsf{id} \times +) \cdot \mathsf{a}$ [5].

Proof.

$$\begin{array}{rcll} \omega \cdot + \cdot (+ \times \mathsf{id}) & = & \cup \cdot (\omega \times \omega) \cdot (+ \times \mathsf{id}) & \{\ \text{definition}\ \} \\ & = & \cup \cdot (\omega \cdot + \times \omega) & \{\ \text{functoriality}\ \} \\ & = & \cup \cdot ((\cup \cdot (\omega \times \omega)) \times \omega & \{\ \text{definition}\ \} \\ & = & \cup \cdot (\cup \times \mathsf{id}) \cdot ((\omega \times \omega) \cdot \omega) & \{\ \text{functoriality}\ \} \\ & = & \cup \cdot (\cup \times \mathsf{id}) \cdot ((\omega \times \omega) \cdot \omega) \cdot \mathsf{a}^\circ \cdot \mathsf{a} & \{\ \mathsf{a}\ \text{is isomorphism}\ \} \\ & = & \cup \cdot (\cup \times \mathsf{id}) \cdot \mathsf{a}^\circ \cdot (\omega \times (\omega \times \omega)) \cdot \mathsf{a} & \{\ \mathsf{a}\ \text{naturality}\ \} \\ & = & \cup \cdot (\mathsf{id} \times \cup) \cdot (\omega \times (\omega \times \omega)) \cdot \mathsf{a} & \{\ \cup\ \text{associativity}\ \} \\ & = & \cup \cdot (\omega \times (\cup \cdot (\omega \times \omega)) \cdot \mathsf{a} & \{\ \text{functoriality}\ \} \\ & = & \cup \cdot (\omega \times \omega \cdot +) \cdot \mathsf{a} & \{\ \text{definition}\ \} \\ & = & \cup \cdot (\omega \times \omega) \cdot (\mathsf{id} \times +) \cdot \mathsf{a} & \{\ \text{functoriality}\ \} \\ & = & \omega \cdot + \cdot (\mathsf{id} \times +) \cdot \mathsf{a} & \{\ \text{definition}\ \} \end{array}$$

$\square$

Interleaving and Restriction.

Persistence through action occurrence justifies the recursive definition of *static* combinators. This means that they arise as anamorphisms generated by suitable 'gene' coalgebras. Both *interleaving* and *restriction* are examples of *static* combinators, which, moreover, depend only on the process structure. We shall consider them in first place.

[5] In the sequel, process properties are stated pointfree, for which we shall resort to standard natural isomorphisms in Set. In particular, associativity, commutativity and product left and right units, will be denoted by $\mathsf{a} : (A \times B) \times C \longrightarrow A \times (B \times C)$, $\mathsf{s} : A \times B \longrightarrow B \times A$, $\mathsf{r} : \mathbf{1} \times A \longrightarrow A$ and $\mathsf{l} : A \times \mathbf{1} \longrightarrow A$, respectively. The converse of an isomorphism i is written as i°.

Although *interleaving*, a binary operator $||| : \nu \times \nu \longrightarrow \nu$, is not considered as a combinator in most process calculi, it represents the simplest form of 'parallel' aggregation in the sense that it is independent of any particular interaction discipline. The following definition captures the intuition that the observations over the interleaving of two processes correspond to all possible interleavings of the observations of its arguments. Thus, $||| \ = \ [\![\alpha_{|||}]\!]$, where [6]

$$\begin{aligned}\alpha_{|||} \ = \ & \nu \times \nu \xrightarrow{\triangle} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)} (\mathcal{P}(Act \times \nu) \times \nu) \times (\nu \times \mathcal{P}(Act \times \nu)) \\ & \xrightarrow{\tau_r \times \tau_l} \mathcal{P}(Act \times (\nu \times \nu)) \times \mathcal{P}(Act \times (\nu \times \nu)) \xrightarrow{\cup} \mathcal{P}(Act \times (\nu \times \nu))\end{aligned}$$

The *restriction* combinator $\backslash_K$, for each subset $K \subseteq L$, forbids the occurrence of actions in K. Formally, $\backslash_K = [\![\alpha_{\backslash_K}]\!]$ where

$$\alpha_{\backslash_K} \ = \ \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathsf{filter}_K} \mathcal{P}(Act \times \nu)$$

where $\mathsf{filter}_K = \lambda s\,.\ \{t \in s |\ \pi_1\, t \notin K\}$.

The interleaving combinator also forms (with nil), an Abelian monoid. Restriction, on the other hand, is idempotent and commutes with both choice and interleaving. As one could expect, the cornerstone in the proofs of equations involving static combinators, is the application of the fusion law (4). This is illustrated below in the proof of $|||$ commutativity.

Proof. By definition $||| \cdot \mathsf{s} = |||$ is equivalent to $[\![\alpha_{|||}]\!] \cdot \mathsf{s} = [\![\alpha_{|||}]\!]$, which, by fusion, is implied by $\alpha_{|||} \cdot \mathsf{s} = \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||}$. Then,

$$\begin{aligned}
& \alpha_{|||} \cdot \mathsf{s} \\
= \ & \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle \cdot \mathsf{s} && \{ \text{ definition } \} \\
= \ & \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot (\mathsf{s} \times \mathsf{s}) \cdot \triangle && \{ \ \triangle \text{ nat } \} \\
= \ & \cup \cdot (\tau_r \times \tau_l) \cdot (\mathsf{s} \times \mathsf{s}) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle && \{ \ \mathsf{s} \times \mathsf{s} \text{ nat } \} \\
= \ & \cup \cdot (\tau_r \cdot \mathsf{s} \times \tau_l \cdot \mathsf{s}) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle && \{ \text{ functor } \} \\
= \ & \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_l \times \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle && \{ \ \tau_r \ \textit{vs} \ \tau_l \ \} \\
= \ & \cup \cdot (\mathcal{P}(\mathsf{id} \times \mathsf{s}) \times \mathcal{P}(\mathsf{id} \times \mathsf{s})) \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle && \{ \text{ functor } \} \\
= \ & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle && \{ \ \cup \text{ nat } \} \\
= \ & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot \mathsf{s} \cdot (\tau_l \times \tau_r) \cdot ((\mathsf{id} \times \omega) \times (\omega \times \mathsf{id})) \cdot \triangle && \{ \ \cup \text{ comm } \} \\
= \ & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \mathsf{s} \cdot \triangle && \{ \ \mathsf{s} \text{ nat } \} \\
= \ & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \cup \cdot (\tau_r \times \tau_l) \cdot ((\omega \times \mathsf{id}) \times (\mathsf{id} \times \omega)) \cdot \triangle && \{ \ \mathsf{s} \cdot \triangle = \triangle \ \} \\
= \ & \mathcal{P}(\mathsf{id} \times \mathsf{s}) \cdot \alpha_{|||} && \{ \text{ definition } \}
\end{aligned}$$

□

[6] Morphisms $\tau_r : \mathcal{P}(Act \times X) \times C \longrightarrow \mathcal{P}(Act \times (X \times C))$ and $\tau_l : C \times \mathcal{P}(Act \times X) \longrightarrow \mathcal{P}(Act \times (C \times X))$ stand for, respectively, the right and left *strength* associated to functor $\mathcal{P}(Act \times \mathsf{Id})$.

4 Interaction and Parallel Composition

Interaction Structures.

Process combinators introduced so far depend solely on the process structure, as recorded in the shape of the functor. To specify *interaction*, however, there is a need to introduce some structure in the set *Act* of actions. Therefore, we define the *interaction structure* underlying a process calculus as an Abelian positive monoid $\langle Act; \theta, 1\rangle$ with a zero element 0. It is assumed that neither 0 nor 1 belong to the set L of labels. The intuition is that θ determines the interaction discipline whereas 0 represents the absence of interaction: for all $a \in Act$, $a\theta 0 = 0$. On the other hand, a positive monoid entails $a\theta a' = 1$ iff $a = a' = 1$. Notice that the role of both 0 and 1 is essentially technical in the description of the interaction discipline. In some situations 1 may be seen as an *idle* action, but its role, in the general case, is to equip the behaviour functor with a monadic structure, which would not be the case if *Act* were defined simply as an Abelian semigroup [7].

A basic example of an interaction structure captures action co-occurrence. Therefore, θ is defined as $a\theta b = \langle a, b\rangle$, for all $a, b \in Act$ different from 0 and 1. Ccs [14] synchronisation discipline provides another example. In this case the set L of labels carries an involutive operation represented by an horizontal bar as in $\overline{a}$, for $a \in L$. Any two actions a and $\overline{a}$ are called complementary. A special action $\tau \notin L$ is introduced to represent the result of a synchronisation between a pair of complementary actions. Therefore, the result of θ is τ whenever applied to a pair of complementary actions and 0 in all other cases, except, obviously, if one of the arguments is 1 [8].

Once an interaction structure is fixed, any homomorphism $f : Act \longrightarrow Act$ lifts to a renaming combinator $[f]$ between processes defined as $[f] = [\![(\alpha_{[f]})]\!]$, where

$$\alpha_{[f]} = \nu \xrightarrow{\omega} \mathcal{P}(Act \times \nu) \xrightarrow{\mathcal{P}(f \times \mathsf{id})} \mathcal{P}(Act \times \nu)$$

Conditional Laws.

The basic properties of renaming, namely that it preserves identity and composition of homomorphisms, extends along *prefix* and commutes with

[7] The structure is similar to what is called a *synchronisation algebra* in [18] apart from some minor details. In particular, Winskel synchronisation algebras carry a specific constant $\star$ to denote asynchronous occurrence and θ does not necessarily possess a unit. The monoid structure, however, allows for a more uniform characterisation of behaviour models. On the other hand, the definition of parallel composition below, in terms of synchronous product and interleaving, avoids the need for introducing $\star$.

[8] For the Ccs case we follow the standard notational convention under which complements are considered implicitly. In particular, a restriction combinator $\backslash_K$, for $K \subseteq L$ is interpreted as $\backslash_{K \cup \overline{K}}$. Similarly, the parameter f of a renaming (see below), specifies only the 'action' part although it also implies that if $f\,a = b$ then $f\,\overline{a} = \overline{b}$. Also, as τ is introduced as a constant in *Act*, f being a homomorphism forces $f\,\tau = \tau$.

both *choice* and *interleaving*, are proved in the style illustrated above, always avoiding the explicit construction of bisimulations. Often, however, process equalities hold just if some 'side conditions' are fulfilled. Let us study how such laws are derived in our framework starting with a very simple example. Let $f = \{b/a\}$ be substitution of a by b, *i.e.*, a homomorphism over *Act* which is the identity in all actions but a. In several cases, but not in all, we may conclude that renaming with f has no effect. Can this be expressed on a general law? A simple calculation yields

$$\begin{aligned}
& [f] = \mathsf{id} \\
\equiv \quad & \{ \text{ definition } \} \\
& [\![(\alpha_{[f]})]\!] = [\![(\omega)]\!] \\
\Leftarrow \quad & \{ \text{ fusion } \} \\
& \alpha_{[f]} \cdot \mathsf{id} = \mathcal{P}(\mathsf{id} \times \mathsf{id}) \cdot \omega \\
\equiv \quad & \{ \text{ identity } \} \\
& \alpha_{[f]} = \omega
\end{aligned}$$

Clearly, the last equality holds only if a does not show up as an action in the immediate continuations of the process being renamed. This condition is formally expressed by the following predicate:

$$\phi \quad = \quad =_{\emptyset} \cdot \cap \cdot (\mathcal{P}\pi_1 \cdot \omega \times \mathsf{sing} \cdot \underline{a}) \cdot \mathsf{l}^{\circ} \tag{5}$$

Note, however, that ϕ is stated as a local condition on the immediate continuations of any process candidate to satisfy the given equality. Therefore, it cannot be directly taken as a sufficient condition for $[f] = \mathsf{id}$. In fact, to proceed, predicate ϕ has to be made into an *invariant* in the sense of [11]. This is justified by the following result.

Theorem 4.1 *Let α and β be* T*-coalgebras and ϕ a predicate on the carrier of β. Then the following 'conditional' fusion law holds*

$$(\phi \Rightarrow (\alpha \cdot h = \mathsf{T}\, h \cdot \beta)) \;\; \Rightarrow \;\; (\Box_{\beta}\, \phi \Rightarrow ([\![(\alpha)]\!]_{\mathsf{T}} \cdot h = [\![(\beta)]\!]_{\mathsf{T}}))$$

Proof. Let X be the carrier of β and i_ϕ the embedding of the subset of X classified by ϕ, *i.e.*, $\phi \cdot i_\phi = \underline{true} \cdot !$. Recall also that any β-invariant ϕ induces a subcoalgebra β'. Consequently, i_ϕ becomes a comorphism from β' to β. Then

$$
\begin{array}{ll}
 & \phi \;\Rightarrow\; (\alpha \cdot h = \mathsf{T}\, h \cdot \beta) \\
\equiv & \qquad \{\; i_\phi \text{ definition } \} \\
 & \alpha \cdot h \cdot i_\phi \;=\; \mathsf{T}\, h \cdot \beta \cdot i_\phi \\
\Rightarrow & \qquad \{\; \Box_\beta\, \phi \subseteq \phi \;\} \\
 & \alpha \cdot h \cdot i_{\Box_\beta \phi} \;=\; \mathsf{T}\, h \cdot \beta \cdot i_{\Box_\beta \phi} \\
\equiv & \qquad \{\; i_{\Box_\beta \phi} \text{ is a comorphism from } \beta' \text{ to } \beta \} \\
 & \alpha \cdot h \cdot i_{\Box_\beta \phi} \;=\; \mathsf{T}\, h \cdot \mathsf{T}\, i_{\Box_\beta \phi} \cdot \beta' \\
\equiv & \qquad \{ \text{ functoriality} \} \\
 & \alpha \cdot h \cdot i_{\Box_\beta \phi} \;=\; \mathsf{T}\, (h \cdot i_{\Box_\beta \phi}) \cdot \beta' \\
\equiv & \qquad \{ \text{ fusion law (4)} \} \\
 & [\![(\alpha)]\!]_{\mathsf{T}} \cdot h \cdot i_{\Box_\beta \phi} \;=\; [\![(\beta')]\!]_{\mathsf{T}} \\
\equiv & \qquad \{\; i_{\Box_\beta \phi} \text{ being a comorphism implies } [\![(\beta')]\!]_{\mathsf{T}} = [\![(\beta)]\!]_{\mathsf{T}} \cdot i_{\Box_\beta \phi} \} \\
 & [\![(\alpha)]\!]_{\mathsf{T}} \cdot h \cdot i_{\Box_\beta \phi} \;=\; [\![(\beta)]\!]_{\mathsf{T}} \cdot i_{\Box_\beta \phi} \\
\equiv & \qquad \{\; i_\phi \text{ definition } \} \\
 & \Box_\beta\, \phi \;\Rightarrow\; ([\![(\alpha)]\!]_{\mathsf{T}} \cdot h = [\![(\beta)]\!]_{\mathsf{T}})
\end{array}
$$

Notice the proof would work if $\Box_\beta\, \phi$ is replaced by any other β-invariant contained in ϕ. As $\Box_\beta\, \phi$ is the greatest such invariant, it provides the most 'generous' condition. $\Box$

Deriving the Condition.

Applying the previous theorem to the case under consideration, leads to

$$[\{b/a\}] \;=\; \mathsf{id} \;\Leftarrow\; \Box_\omega\, \phi \tag{6}$$

with ϕ given by (5). Now recall that $\Box_\omega\, \phi$ is defined as the greatest fixpoint of $\Phi = \lambda x \,.\, \phi \cap \bigcirc_\omega x$. Looking at predicates as sets, Φ is a function over a complete lattice — $\langle \mathcal{P}\nu, \subseteq \rangle$ — whose monotony is easily proved by induction on the functor structure. Therefore, a concrete representation for $\Box_\omega\, \phi$ can be computed, by the Knaster-Tarski theorem [16], as the union of all post-fixpoints of Φ, *i.e.*,

$$\Box_\omega\, \phi \;=\; \bigcup \{ s \in \mathcal{P}\nu \,|\; s \subseteq \phi \cap \bigcirc_\omega s \}$$

Being a post-fixpoint means, for each s above that, for any process p,

$$
\begin{aligned}
& p \in s \\
& \quad \Rightarrow \; p \in \phi \;\wedge\; p \in \bigcirc_\omega s \\
& \quad \equiv \; p \in \phi \;\wedge\; p \in \{x \in \nu |\; \omega\, x \in (s)^{\mathcal{P}(Act \times \mathsf{Id})}\} \\
& \quad \equiv \; p \in \phi \;\wedge\; p \in \{x \in \nu |\; \omega\, x \in \{c \in \mathcal{P}(Act \times \nu) |\; \forall_t \,.\, t \in c \Rightarrow t \in (s)^{Act \times \mathsf{Id}}\}\} \\
& \quad \equiv \; p \in \phi \;\wedge\; p \in \{x \in \nu |\; \omega\, x \in \{c \in \mathcal{P}(Act \times \nu) |\; \forall_t \,.\, t \in c \Rightarrow \pi_2\, t \in s\}\} \\
& \quad \equiv \; p \in \phi \;\wedge\; p \in \{x \in \nu |\; (\mathcal{P}\pi_2 \cdot \omega)\, x \in s\}
\end{aligned}
$$

Seen as a set, predicate (5) is given by $\phi = \{x \in \nu |\; (\mathcal{P}\pi_1 \cdot \omega)\, x \cap \{a\} = \emptyset\}$. Therefore,

$$\Box_\omega\, \phi \;=\; \bigcup \{s \in \mathcal{P}\nu |\; x \in s \;\Rightarrow\; ((\mathcal{P}\pi_1 \cdot \omega)\, x \cap \{a\} = \emptyset) \;\wedge\; ((\mathcal{P}\pi_2 \cdot \omega)\, x \in s))\}$$

or, in words, the set of all processes whose derivations never exhibit an action a.

In Ccs, the set of all labels, seen as actions, in which a process p can commit itself, *i.e.*, that appear in at least one derivation of p, is called the *sort* of p and denoted by $\mathcal{L}(p)$. [14] provides a syntactic criterion to compute a majoring approximation of $\mathcal{L}(p)$ by induction on the process expression. A semantic definition can, however, be given as

$$\mathcal{L}(p) = (\mathcal{P}\pi_1 \cdot \bigcup \cdot \mathcal{P}p)\; \Box_p\, true \tag{7}$$

where, again, the embedding of L in Act is left implicit. Law (6) may then be rewritten as [9]

$$[\{b/a\}] \;=\; \mathsf{id} \quad \Leftarrow \notin^a \cdot \mathcal{L} \tag{8}$$

Parallel.

The next static operator considered here is *synchronous product*, modelling the simultaneous execution of its two arguments. In each step the resulting action is determined by the interaction structure for the calculus. Formally, $\otimes \;=\; [\![\alpha_\otimes]\!]$ where

$$\alpha_\otimes \;=\; \nu \times \nu \xrightarrow{(\omega \times \omega)} \mathcal{P}(Act \times \nu) \times \mathcal{P}(Act \times \nu) \xrightarrow{\mathsf{sel} \cdot \boldsymbol{\delta_r}} \mathcal{P}(Act \times (\nu \times \nu))$$

where $\mathsf{sel} = \mathsf{filter}_{\{0\}}$ filters out all synchronisation failures. Notice how interaction is catered by $\boldsymbol{\delta_r}$ — the distributive law for the strong monad $\mathcal{P}(Act \times \mathsf{Id})$ [10]. $\boldsymbol{\delta_r}$ is the Kleisli composition of the left and the right strengths. This, on its

[9] Going pointwise and noticing that, by convention, the parameter of the Ccs renaming operator represents 'coactions' implicitly, we end up with the familiar Ccs law $p\,[\{b/a\}] = p \;\Leftarrow\; a, \overline{a} \notin \mathcal{L}(p)$. Function $\notin^a$ is defined as $\lambda x \,.\, a \notin x$.

[10] Notice that the monoidal structure in Act extends functor $\mathcal{P}(Act \times \mathsf{Id})$ to a strong monad.

turn, involves the application of the monad multiplication to 'flatten' the result and this, for a monoid monad, requires the suitable application of the underlying monoidal operation, which, in our case, fixes the interaction discipline. In fact,

$$\begin{aligned}\boldsymbol{\delta_r}^{\mathcal{P}(Act\times\mathsf{Id})} &= \mu^{\mathcal{P}(Act\times\mathsf{Id})} \cdot \mathcal{P}(\mathsf{id}\times\tau_r{}^{\mathcal{P}(Act\times\mathsf{Id})}) \cdot \tau_l{}^{\mathcal{P}(Act\times\mathsf{Id})}\\ &= \mathcal{P}((\theta\times\mathsf{id})\cdot\mathsf{a}^{\circ})\cdot\bigcup\cdot\mathcal{P}\tau_l{}^{\mathcal{P}} \cdot\mathcal{P}(\mathsf{id}\times\tau_r{}^{\mathcal{P}(Act\times\mathsf{Id})})\cdot\tau_l{}^{\mathcal{P}(Act\times\mathsf{Id})}\end{aligned}$$

i.e., going pointwise,

$$\boldsymbol{\delta_r}^{\mathcal{P}(Act\times\mathsf{Id})}\,\langle c_1, c_2\rangle \;=\; \{\langle a'\theta a, \langle p, p'\rangle\rangle |\; \langle a, p\rangle \in c_1 \,\wedge\, \langle a', p'\rangle \in c_2\}$$

Finally, *parallel* composition arises as a combination of *interleaving* and *synchronous product*, in the sense that the evolution of $p \mid q$, for processes p and q, consists of all possible derivations of p and q plus the ones associated to the synchronisations allowed by the particular interaction structure for the calculus. This cannot be achieved by a simple composition of the corresponding combinators $|||$ and $\otimes$: it has to be performed at the 'genes' level for $|||$ and $\otimes$. Formally, $\mid\, = [\![(\alpha_{|})]\!]$, where

$$\alpha_{|} \;=\; \nu\times\nu \xrightarrow{\triangle} (\nu\times\nu)\times(\nu\times\nu) \xrightarrow{\cup\cdot(\alpha_{|||}\times\alpha_{\otimes})} \mathcal{P}(Act\times(\nu\times\nu))$$

Synchronous product is commutative, associative and has nil as a zero element. Furthermore, it distributes over *choice* and, conditionally, over *restriction* and *renaming*. On the other hand, as expected, the $\mid$ combinator shares some properties that are common to both $|||$ and $\otimes$. In particular, it gives rise, with nil, to an Abelian monoid and distributes along renaming and restriction in certain cases. However, it lacks a zero element and does not distribute through choice. Notice that the verification of such properties 're-uses' the proofs of the corresponding results for $|||$ and $\otimes$.

Two Proofs.

This paragraph is concerned with the proof of restriction distributivity over $\otimes$ and $|$, illustrating two important points in our approach: the derivation of side conditions along a proof and proof re-use. The proofs rely on the following immediate consequence of law (1): to prove the equality $\phi = \psi$ it is enough to show that both $\omega\cdot\phi = \mathsf{T}\,\phi\cdot\alpha$ and $\omega\cdot\psi = \mathsf{T}\,\psi\cdot\alpha$ hold.

Consider, then, the derivation of the following property:

$$\backslash_K\cdot\otimes \;=\; \otimes\cdot(\backslash_K\times\backslash_K) \qquad \Leftarrow\; \mathsf{uniform\_restriction} \tag{9}$$

Proof. We proceed by unfolding the composite of ω with both sides of the equation. Therefore,

$$\begin{array}{ll}
& \omega \cdot \otimes \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{\text{ comorphism, definition}\} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\omega \times \omega) \cdot (\backslash_K \times \backslash_K) \\
= & \quad \{\text{ functoriality}\} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\omega \cdot \backslash_K \times \omega \cdot \backslash_K) \\
= & \quad \{\text{ comorphism, functoriality}\} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\mathcal{P}(\mathsf{id} \times \backslash_K) \times \mathcal{P}(\mathsf{id} \times \backslash_K)) \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= & \quad \{\ \boldsymbol{\delta_r} \text{ naturality}\} \\
& \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \mathcal{P}(\mathsf{id} \times (\backslash_K \times \backslash_K)) \cdot \boldsymbol{\delta_r} \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K}) \\
= & \quad \{\ \mathsf{sel} \text{ definition, functoriality}\} \\
& \mathcal{P}(\mathsf{id} \times \otimes \cdot (\backslash_K \times \backslash_K)) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})
\end{array}$$

Next a similar calculation is done for $\omega \cdot \backslash_K \cdot \otimes$, trying to arrive at an expression $\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \alpha$, with $\alpha = \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})$ as above.

$$\begin{array}{ll}
& \omega \cdot \backslash_K \cdot \otimes \\
= & \quad \{\text{ comorphism, definition}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \omega \cdot \otimes \\
= & \quad \{\text{ comorphism}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \alpha_{\otimes} \\
= & \quad \{\text{ definition}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\omega \times \omega) \\
= & \quad \{\ \mathsf{filter}_K \text{ naturality}\} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{filter}_K \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\omega \times \omega) \\
\overset{?}{=} & \quad \{\ \star\ \} \\
& \mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \otimes) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\mathsf{filter}_K \times \mathsf{filter}_K) \cdot (\omega \times \omega) \\
= & \quad \{\text{ functoriality }\} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\mathsf{filter}_K \cdot \omega \times \mathsf{filter}_K \cdot \omega) \\
= & \quad \{\text{ definition}\} \\
& \mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \otimes)) \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})
\end{array}$$

We have succeeded only partially: the step marked with a $\star$ does not hold universally. We are then left with the task of establishing under what conditions, if any, the following equality holds:

$$\mathsf{filter}_K \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r} \ = \ \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\mathsf{filter}_K \times \mathsf{filter}_K)$$

Unfolding the definitions of the functions involved and going pointwise for a while, we get

$$\begin{aligned}
&(\,\mathsf{filter}_K \cdot \mathsf{sel} \cdot \boldsymbol{\delta_r})\,\langle c_1, c_2\rangle \;= \\
&\;= \;(\mathsf{filter}_K \cdot \mathsf{sel})\,\{\langle a'\theta a, \langle p, p'\rangle\rangle |\; \langle a, p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2\} \\
&\;= \;\mathsf{filter}_K\,\{\langle a'\theta a, \langle p, p'\rangle\rangle |\; \langle a, p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a'\theta a \neq 0\} \\
&\;= \;\{\langle a'\theta a, \langle p, p'\rangle\rangle |\; \langle a, p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a'\theta a \neq 0 \;\wedge\; a'\theta a \notin K\}
\end{aligned}$$

On the other hand,

$$\begin{aligned}
&(\,\mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\mathsf{filter}_K \times \mathsf{filter}_K))\,\langle c_1, c_2\rangle \;= \\
&\;= \;(\mathsf{sel} \cdot \boldsymbol{\delta_r})\;\langle \{\langle a, p\rangle \in c_1 |\; a \notin K\}, \{\langle a', p'\rangle \in c_2 |\; a' \notin K\}\rangle \\
&\;= \;\mathsf{sel}\;\{\langle a'\theta a, \langle p, p'\rangle\rangle |\; \langle a, p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a, a' \notin K\} \\
&\;= \;\{\langle a'\theta a, \langle p, p'\rangle\rangle |\; \langle a, p\rangle \in c_1 \;\wedge\; \langle a', p'\rangle \in c_2 \;\wedge\; a, a' \notin K \;\wedge\; a'\theta a \neq 0\}
\end{aligned}$$

Clearly the two sets can be identified iff, for all possible a and a', such that $a'\theta a \neq 0$, $a'\theta a \notin K \equiv a, a' \notin K$. Therefore, step $\star$ is only possible if the expression scope is restricted to pairs of processes satisfying the following predicate:

$$\phi\,\langle p, q\rangle \;=\; \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\,p,\, a' \in (\mathcal{P}\pi_1 \cdot \omega)\,q}\;.\; a\theta a' \neq 0 \;\Rightarrow\; (a\theta a' \notin K \equiv a, a' \notin K) \quad (10)$$

which is lifted to the invariant $\mathsf{uniform\_restriction} \;=\; \Box_\alpha\,\phi$. □

As expected, a similar result holds for parallel composition.

Proof. The attempt to prove $\backslash_K \cdot \;|\!=\!|\; \cdot (\backslash_K \times \backslash_K)$, proceeds by reducing the composite of ω with both sides of the equation to identify a common coalgebra α'. Thus,

$$\begin{aligned}
&\omega \cdot \backslash_K \cdot \;| \\
=\;\; &\{\text{ double application of comorphism and definition}\} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathsf{filter}_K \cdot \mathcal{P}(\mathsf{id} \times \;|) \cdot \cup \cdot (\alpha_{|||} \times \alpha_{\otimes}) \cdot \triangle \\
=\;\; &\{\;\mathsf{filter}_K \text{ naturality}\} \\
&\mathcal{P}(\mathsf{id} \times \backslash_K) \cdot \mathcal{P}(\mathsf{id} \times \;|) \cdot \mathsf{filter}_K \cdot \cup \cdot (\alpha_{|||} \times \alpha_{\otimes}) \cdot \triangle \\
=\;\; &\{\;\cup \text{ naturality, functoriality}\} \\
&\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \;|)) \cdot \cup \cdot (\mathsf{filter}_K \cdot \alpha_{|||} \times \mathsf{filter}_K \cdot \alpha_{\otimes}) \cdot \triangle \\
=\;\; &\{\text{ reusing the proof of (9) and a similar result for } |||\} \\
&\mathcal{P}(\mathsf{id} \times (\backslash_K \cdot \;|) \cdot \cup \cdot ((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle) \\
&\qquad \times \;\mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})) \cdot \triangle
\end{aligned}$$

Similarly, it is shown that $\omega \cdot \;| \cdot (\backslash_K \times \backslash_K) \;=\; \mathcal{P}(\mathsf{id} \times (| \cdot (\backslash_K \times \backslash_K))) \cdot \alpha'$, where

$$\alpha' \;=\; \cup \cdot ((\cup \cdot (\tau_r \times \tau_l) \cdot ((\alpha_{\backslash_K} \times \mathsf{id}) \times (\mathsf{id} \times \alpha_{\backslash_K})) \cdot \triangle) \times \mathsf{sel} \cdot \boldsymbol{\delta_r} \cdot (\alpha_{\backslash_K} \times \alpha_{\backslash_K})) \cdot \triangle$$

is the common coalgebra. We are, thus, almost ready to conclude. Notice, however, that, on reusing the proof of law (9) in the last step of this derivation, we must also take into account the predicate which constrains the substitution made. Clearly, predicate (10) acts again as a local condition to validate the derivation here. We may then conclude the validity of the law, subjected to the restriction given by $\mathsf{uniform\_restriction}' = \Box_{\alpha'}\,\phi$. $\Box$

We may ask what form such invariants take given a particular interaction structure. For example, in the Ccs case, the result of θ does not belong to L — θ has only three possible results under Ccs interaction discipline: τ, 1 and 0. Therefore, as $K \subseteq L$, condition $a'\theta a \notin K$ holds for any K and ϕ becomes

$$
\begin{array}{ll}
 & \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\,p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\,q} \;.\; a\theta a' \neq 0 \;\Rightarrow\; (a\theta a' \notin K \;\equiv\; a, a' \notin K) \\
\equiv & \{\ \text{Ccs interaction structure}\ \} \\
 & \forall_{a \in \mathcal{P}(\pi_1 \cdot \omega)\,p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\,q} \;.\; (a\theta a' = \tau \,\vee\, a\theta a' = 1) \;\Rightarrow\; a, a' \notin K \\
\equiv & \{\ a\theta a' = 1 \ \mathsf{iff}\ a = a' = 1\ \} \\
 & \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\,p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\,q} \;.\; a\theta a' = \tau \;\Rightarrow\; a, a' \notin K \\
\equiv & \{\ a\theta a' = \tau \ \mathsf{iff}\ a' = \overline{a}\} \\
 & \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\,p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\,q} \;.\; a' = \overline{a} \;\Rightarrow\; a, a' \notin K \\
\equiv & \{\ \text{rearranging}\} \\
 & (\mathcal{P}\pi_1 \cdot \omega)\,p \cap \overline{(\mathcal{P}\pi_1 \cdot \omega)\,q} \cap (K \cup \overline{K}) = \emptyset
\end{array}
$$

where the overbar notation stands here for the lifting of the involutive Ccs complement operation to sets of actions. Now note that, although both invariants are derived from the same local condition (10), they stand for the closure of ϕ under different coalgebras and are, consequently, distinct. In fact, the number of derivations that have to be considered under ϕ is much greater in the second case. In particular, $\mathsf{uniform\_restriction}$ does not consider configurations representing interleavings. For example, if one of the processes exhausts after n steps, the condition on the actions is not required to hold after that. On the other hand, in $\mathsf{uniform\_restriction}'$, ϕ is closed wrt the transitions on α', which include both interleavings and synchronisations. Therefore, and recalling the notion of *sort*, we conclude that $\mathsf{uniform\_restriction}' \;\equiv\; \mathcal{L}(p) \cap \overline{\mathcal{L}(p)} \cap (K \cup \overline{K}) = \emptyset$ arriving to the following familiar Ccs presentation of this result

$$(p \mid q)\backslash_K = p\backslash_K \mid q\backslash_K \qquad \Leftarrow\; \mathcal{L}(p) \cap \overline{\mathcal{L}(p)} \cap (K \cup \overline{K}) = \emptyset$$

This discussion illustrates our claim that such an approach to process calculi allows us to 'discover' the appropriate restrictions a law is constrained by, instead of 'postulating' them and verifying their suitability. It also makes explicit that such conditions are essentially dependent only on the calculus interaction structure. Consider, for example, what would happen to the law at hands if a Csp-like interaction discipline is chosen instead. In Csp [10] only equally named actions synchronise, leading to the following definition of θ:

$$a\theta a \;=\; a\,, \quad a\theta 1 \;=\; 1\theta a \;=\; a \;\text{ and }\; a\theta b \;=\; 0 \;\text{ in all other cases}$$

Therefore, the condition $a\theta a' \neq 0 \;\Rightarrow\; (a\theta a' \notin K \;\;\equiv\;\; a, a' \notin K)$ becomes trivially true and the law holds without any side condition.

Another Example.

A similar situation occurs when studying the distribution of *renaming* over *parallel*. In an attempt to proof $[f]\boldsymbol{\cdot} \;|=|\; \boldsymbol{\cdot}([f] \times [f])$, application of theorem 4.1 makes the following local condition to pop out (see [2] for the complete derivation):

$$\phi\,\langle p, q\rangle \;=\; \forall_{a \in (\mathcal{P}\pi_1 \cdot \omega)\,p, a' \in (\mathcal{P}\pi_1 \cdot \omega)\,q}\;.\; a\theta a' \neq 0 \;\equiv\; fa\,\theta fa' \neq 0$$

Under the Ccs interaction discipline, $a\theta a' \neq 0$ implies that $a\theta a' = 1$ or $a\theta a' = \tau$. Hence, when does the equivalence $a\theta a' \neq 0 \;\equiv\; fa\,\theta fa' \neq 0$ hold? Clearly the implication from left to right holds trivially because f is a homomorphism on the interaction structure:

$$\begin{array}{ll} & a\theta a' \;=\; \star \\ \Rightarrow & \quad \{\text{ Leibniz }\} \\ & f\,(a\theta a') \;=\; f\,\star \\ \equiv & \quad \{\; f \text{ is an } \textit{Act}\text{-homomorphism }\} \\ & fa\,\theta fa' \;=\; \star \end{array}$$

when $\star$ stands for either 1 or τ. The implication in the opposite direction, however, reads

$$fa\,\theta fa' = \star \;\Rightarrow\; a\theta a' \;=\; \star$$

which, f being an *Act*-homomorphism, is equivalent to

$$f\,(a\theta a') \;=\; f\star \;\Rightarrow\; a\theta a' \;=\; \star$$

again for $\star$ standing for either 1 or τ. This holds only if f is *mono*. Thus, for the Ccs case, the generated invariant requires the injectivity of f or, at least, of its restriction to the relevant process sorts. The resulting law is usually written in Ccs as

$$(p \mid q)[f] = p[f] \mid q[f] \qquad \Leftarrow \; f \text{ restricted to } \mathcal{L}(p \mid q) \cup \overline{\mathcal{L}(p \mid q)} \text{ is mono}$$

5 Variants and Conclusions

Behaviour Monads.

In the introduction to this paper we have remarked that the approach to process calculi design sketched here could cope with a variety of particular cases because the emphasis was placed on the common underlying structures rather than on the distinctive particularities. A first source of genericity has already been introduced by separating the *behaviour* from the *interaction* structures. Note that all the process combinators introduced are either *independent* of any particular interaction discipline or *parametrized* by it. We shall now briefly examine what happens if the behaviour structure itself is changed. Recall that processes were defined as inhabitants of the carrier of the final coalgebra for $\mathsf{T} = \mathsf{B}\ (Act \times \mathsf{Id})$ where B was taken as the finite powerset functor. Next we have shown that, assuming a commutative monoidal structure over *Act*, the behaviour model captured by $\mathcal{P}(Act \times \mathsf{Id})$ is a *strong Abelian monad*. The definitions of some combinators build upon this — in particular *synchronous product* basically relies on the monad distribution law $\boldsymbol{\delta_r}$. Commutativity of $\otimes$, and consequently of $|$, depends on the monoid underlying the interaction structure being itself Abelian. Therefore, a first line of enquire followed below consists of replacing B by different monads and extracting a correspondent family of calculi still parametrized by the interaction structure.

The simplest case takes B as the identity Id. The result is, of course, a universe of *deterministic* (and perpetual) processes. A further elaboration of this replaces Id by $\mathsf{Id} + \mathbf{1}$, entailing a calculus for deterministic but *partial* processes in the sense that the derivation of a 'dead' state is always possible. Such a calculus is far less expressive than the one previously discussed. In fact derivations do not form any kind of *collection* and, therefore, non determinism is ruled out. Similarly, combinators which explore non determinism, lack a counterpart here. Such is the case of *choice*, *interleaving* and, as a generalisation of the later, *parallel*. On the other hand, the composition of $\mathsf{Id} + \mathbf{1}$ with the monoidal monad generated by *Act* is still a strong Abelian monad, and therefore *synchronous product* is still definable. *Inaction* and *prefix*, for partial processes, are defined as $\omega \cdot \mathsf{nil} = \iota_2$ and $\omega \cdot a. \ = \ \iota_1 \cdot \mathsf{label}_a$, respectively.

On the other hand, *product*, *restriction* and *renaming* can be defined in a rather generic way as anamorphisms whose 'genes' are parametrized by the monad B:

$$\begin{aligned}
\alpha_{[f]} &= \nu \xrightarrow{\omega} \mathsf{B}\ (Act \times \nu) \xrightarrow{\mathsf{B}\ (f \times \mathsf{id})} \mathsf{B}\ (Act \times \nu) \\
\alpha_{\backslash K} &= \nu \xrightarrow{\omega} \mathsf{B}\ (Act \times \nu) \xrightarrow{\mathsf{filter}_K{}^{\mathsf{B}}} \mathsf{B}\ (Act \times \nu) \\
\alpha_{\otimes} &= \nu \times \nu \xrightarrow{(\omega \times \omega)} \mathsf{B}\ (Act \times \nu) \times \mathsf{B}\ (Act \times \nu) \xrightarrow{\boldsymbol{\delta_r}^{\mathsf{B}\ (Act \times \mathsf{Id})}} \mathsf{B}\ (Act \times (\nu \times \nu)) \\
&\quad \xrightarrow{\mathsf{sel}^{\mathsf{B}}} \mathsf{B}\ (Act \times (\nu \times \nu))
\end{aligned}$$

where $\boldsymbol{\delta_r}^{\mathsf{B}\ (Act \times \mathsf{Id})}$ is the distribution law associated to the composed monad $\mathsf{B}\ (Act \times \mathsf{Id})$, therefore encapsulating the θ operation on actions. On the other hand, $\mathsf{sel}^{\mathsf{B}}$ and $\mathsf{filter}_K{}^{\mathsf{B}}$ explore the B structure in order to rule out synchronisation failures, in the first case, and to perform the action restriction in the second. For $\mathsf{B} = Act + \mathbf{1}$, $\mathsf{filter}_K{}^{\mathsf{Id}+\mathbf{1}} = ((\notin_K \cdot \pi_1) \ \rightarrow \ \iota_1, \, \iota_2 \cdot !) + \mathsf{id}$ is expressed by a conditional, whereas, for $\mathsf{B} = \mathcal{P}$, such conditional was iterated over a set. Notice that $\notin_K$ is defined as $\lambda a\,.\; a \notin K$ and, again, $\mathsf{sel}^{\mathsf{Id}+\mathbf{1}} = {\mathsf{filter}_{\{0\}}}^{\mathsf{Id}+\mathbf{1}}$. Finally, notice that when B is non commutative, which is the case of, *e.g.*, $\mathsf{B}\; X = X^{\star}$, not only is commutativity lost for several combinators, but also two non bisimilar versions of $\otimes$ emerge, based, respectively, in $\boldsymbol{\delta_r}$ and $\boldsymbol{\delta_l}$, as equation $\boldsymbol{\delta_r} = \boldsymbol{\delta_l}$ holds only for commutative monads.

Reactive Processes.

Up to this point we have assumed an 'active' interpretation of processes. Similarly, an universe for reactive processes may be specified as a final coalgebra $\overline{\omega}$ for $\mathsf{T}\,X = (\mathsf{B}\; X)^{Act}$, where B captures, as usual, the behaviour structure. Notice that in this paragraph the overbar notation is used to refer to the transpose of a morphism under the *curry/uncurry* isomorphism. Let us revisit briefly the process combinators in this new setting, taking again B as the finite powerset monad.

The definitions of *inaction*, *prefix* and *choice* follow closely the ones already considered for 'active' processes, reflecting, however, the fact that *Act* appears now as an exponent. Notice, for example, how prefixing a process p by an action a results in a new process which is blind for every stimulus different from a. Formally,

$$\begin{aligned}
\overline{\omega} \cdot \mathsf{nil} \;&=\; \overline{\underline{\emptyset}\cdot !} \\
\overline{\omega} \cdot a. \;&=\; \overline{((=_a \cdot \pi_2) \;\rightarrow\; \mathsf{sing} \cdot \pi_1,\; \underline{\emptyset}\cdot !)} \\
\overline{\omega} \cdot + \;&=\; \overline{\cup \cdot (\omega \times \omega) \cdot \mathsf{pdl}}
\end{aligned}$$

where $\mathsf{pdl} : (X \times Y) \times Z \longrightarrow (X \times Y) \times (Y \times Z)$ is defined as $\langle \pi_1 \times \mathsf{id}, \pi_2 \times \mathsf{id} \rangle$.

For any $K \in L$ and renaming homomorphism f, *restriction* and *renaming* are given by $\backslash_K = [\![\overline{\alpha_K}]\!]$ and $[f] = [\![\overline{\alpha_f}]\!]$, where $\alpha_K = ((\notin_K \cdot \pi_2) \;\rightarrow\; \omega,\; \underline{\emptyset}\cdot !)$ and $\alpha_f = \omega \cdot (\mathsf{id} \times f)$. Notice that both f and the restriction set K act by constraining the set of meaningful stimuli, thus before the effective derivation is computed. Under the 'active' interpretation their effect was defined over the (previously computed) derivations.

The *synchronous product* is defined, in this setting, as $\otimes \;=\; [\![\alpha_\otimes]\!]$, where,

$$\begin{aligned}
\alpha_\otimes \;=\; \nu \times \nu \xrightarrow{(\overline{\omega}\times\overline{\omega})} \mathcal{P}\nu^{Act} \times \mathcal{P}\nu^{Act} \xrightarrow{\mathsf{prod}} \mathcal{P}(\mathcal{P}\nu \times \mathcal{P}\nu)^{Act} \\
\xrightarrow{(\mathcal{P}\boldsymbol{\delta_r}^{\mathcal{P}})^{Act}} \mathcal{P}\mathcal{P}(\nu \times \nu)^{Act} \xrightarrow{\cup^{Act}} \mathcal{P}(\nu \times \nu)^{Act}
\end{aligned}$$

where

$$\mathsf{prod}\,\langle d_1, d_2 \rangle \;=\; \lambda a\,.\;\; (a = 0 \;\rightarrow\; \emptyset,\; \{\langle d_1\, a_1, d_2\, a_2 \rangle \,|\, \forall_{a_1,a_2 \in Act}\,.\; a = a_1 \theta a_2\})$$

Recall that in the 'active' case, interaction was neatly captured by the distribution law for the $\mathcal{P}(Act \times \mathsf{Id})$ monad, which is no longer the case here. This entails the need to introduce function prod, which, additionally, filters out synchronisation failures. On the other hand, the definition of the *interleaving* combinator corresponds to the one for the 'active' case, but for the replacement of set union by merge where $\mathsf{merge}\,\langle d_1, d_2\rangle = \lambda a.\ \ d_1\, a \cup d_2\, a$. Thus, $||| = [\![(\alpha_{|||})]\!]$ with

$$\begin{aligned}\alpha_{|||} &= \nu \times \nu \xrightarrow{\Delta} (\nu \times \nu) \times (\nu \times \nu) \xrightarrow{(\overline{\omega}\times\mathsf{id})\times(\mathsf{id}\times\overline{\omega})} (\mathcal{P}\nu^{Act} \times \nu) \times (\nu \times \mathcal{P}\nu^{Act}) \\ &\xrightarrow{\tau_r \times \tau_l} \mathcal{P}(\nu \times \nu)^{Act} \times \mathcal{P}(\nu \times \nu)^{Act} \xrightarrow{\mathsf{merge}} \mathcal{P}(\nu \times \nu)^{Act}\end{aligned}$$

where, of course, the right and left strengths are relative to the $(\mathcal{P}\mathsf{Id})^{Act}$ monad. The same observation applies to *parallel* composition, which arises, once again, as a combination of product and interleaving at the 'genes' level. The basic properties of this model of reactive processes are essentially the properties of the corresponding calculus of 'active' processes. The proof style is also similar, although calculation resorts now heavily to the properties of exponentiation (see [2]).

Process Languages and Prototyping.

One advantage of thinking about processes as inhabitants of (coinductive) types is the possibility of developing prototypes for process calculi in functional languages supporting such types. Once a prototype implementation of a particular calculus is developed, processes can be defined in the language and their execution traced. Furthermore, to deal with *recursive* processes, a *language* of process expressions has to be defined allowing for *guarded* occurrences of process *variables* as valid terms. As expected, a term language for processes, over a set of labels, is defined as an inductive type. The set of terms is then taken as the carrier of a $\mathsf{B}\ (Act \times \mathsf{Id})$-coalgebra, whose coinductive extension (*i.e.*, the associated *anamorphism*) provides an interpreter for the calculus. Moreover, process environments have to be introduced to collect all the process defining equations relevant to conduct experiments on a particular network of processes. The environment acts as context information for the interpreter which is, consequently, redefined as a *strong* anamorphism. Such a strategy is used by the author, in [2], to develop Charity implementations of interpreters for elementary process calculi parametric on the interaction structure.

References

[1] P. Aczel. Final universes of processes. In Brooks et al, editor, *Proc. Math. Foundations of Programming Semantics.* Springer Lect. Notes Comp. Sci. (802), 1993.

[2] L. S. Barbosa. *Components as Coalgebras.* PhD thesis, Universidade do Minho (submitted), 2000.

[3] L. S. Barbosa. Components as processes: An exercise in coalgebraic modeling. In S. F. Smith and C. L. Talcott, editors, *FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems*, pages 397–417, Stanford, USA, September 2000. Kluwer Academic Publishers.

[4] R. Bird and O. Moor. *The Algebra of Programming.* Series in Computer Science. Prentice-Hall International, 1997.

[5] R. S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*, volume 36 of *NATO ASI Series F*, pages 3–42. Springer-Verlag, 1987.

[6] R. S. Bird and L. Meertens. Two exercises found in a book on algorithmics. In L. Meertens, editor, *Program Specification and Transformation*, pages 451–458. North-Holland, 1987.

[7] R. Cockett and T. Fukushima. About Charity. Yellow Series Report No. 92/480/18, Dep. Computer Science, University of Calgary, June 1992.

[8] T. Hagino. *Category Theoretic Approach to Data Types.* Ph.D. thesis, tech. rep. ECS-LFCS-87-38, Laboratory for Foundations of Computer Science, University of Edinburgh, UK, 1987.

[9] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Information & Computation*, (145):105–121, 1998.

[10] C. A. R Hoare. *Communicating Sequential Processes.* Series in Computer Science. Prentice-Hall International, 1985.

[11] B. Jacobs. The temporal logic of coalgebras via Galois algebras. Techn. rep. CSI-R9906, Comp. Sci. Inst., University of Nijmegen, 1999.

[12] G. R. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–279, 1990.

[13] E. Meijer, M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In J. Hughes, editor, *Proceedings of the 1991 ACM Conference on Functional Programming Languages and Computer Architecture*, pages 124–144. Springer Lect. Notes Comp. Sci. (523), 1991.

[14] A. J. R. G. Milner. *Communication and Concurrency.* Series in Computer Science. Prentice-Hall International, 1989.

[15] L. Moss. Coalgebraic logic. *Ann. Pure & Appl. Logic*, 1999.

[16] A. Tarski. A lattice–theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[17] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Mathematical Structures in Computer Science*, 8(5):481–540, 1998.

[18] G. Winskel and M. Nielsen. Models for Concurrency. In S. Abramsky, D. M. Gabbay, and T. S. E. Gabbay, editors, *Handbook of Logic in Computer Science (vol. 4)*, pages 1–148. 1995.