COMPONENTS AS PROCESSES: AN EXERCISE IN COALGEBRAIC MODELING

L. S. Barbosa
Computer Science Department
University of Minho - Portugal
lsb@di.uminho.pt

Abstract Software components, arising, typically, in systems' analysis and design, are characterized by a public interface and a private encapsulated state. They persist (and evolve) in time, according to some behavioural patterns. This paper is an exercise in modeling such components as *coalgebras* for some kinds of endofunctors on Set, capturing both (interface) types and behavioural aspects. The construction of component categories, cofibred over the interface space, emerges by generalizing the usual notion of a coalgebra morphism. A collection of composition operators as well as a generic notion of bisimilarity, are discussed.

Keywords: Software component; composition; coalgebra.

1. INTRODUCTION

1. MOTIVATION. Words like *component* or *process* are so semantically overloaded that their use is often a risk. What is understood by a *software component* in this paper is a specification of a state-based module, eventually acting as a *building block* of larger, often distributed, systems. Typically, a component encapsulates a number of services through a public interface which provides a limited access to the state space. Furthermore, it is intended to persist and evolve in time. Components arise, typically, as specification units in systems' analysis and design, namely in the so-called *model oriented* specification methods, such as VDM [11] or Z [19]. However, it is often difficult to identify and reason about their composition mechanisms and underlying behavioural patterns. This is due firstly to the presence of internal state spaces that cannot be discarded and, secondly, to the possibility of interaction proceeding during the overall computation.

This paper suggests *coalgebra* theory [16, 10, 20] as a suitable framework to approach such issues. This relatively new field has been recognized as a natural setting to deal with state-based dynamic systems where observations

and their patterns seem far more relevant than data construction. In particular, coalgebraic methods have been used to approach the semantics of concurrent systems [17] and the specification of object oriented programs [15, 8].

Although adopting a model-oriented approach to specification, instead of the axiomatic setting common to the above mentioned references, our contribution has a similar motivation. In particular, we discuss how software components can be modeled as coalgebras for a class of functors parametrized by a monad which captures particular behaviour paradigms (such as partiality or non determinism). The shape of the functor, determined by the application in mind, entails a suitable notion of *bisimilarity*. Finally, some composition operators are defined in a way which is independent of the chosen behavioural paradigm. Such operators have some resemblance with variants of parallel composition and renaming found in *process algebras* (see *e.g.*, [7, 12]). However, the presence of (eventually structured) input and output and their transformations, which plays a minor role on process algebra, is fully handled here. Presented as an exercise in coalgebraic modeling, this paper is divided in three main sections discussing, respectively, how software components can be *specified*, *compared* and *composed*. The notation used and the notion of a monad are briefly reviewed in the next two paragraphs.

2. NOTATION. Some elementary category theory will be used throughout the paper to establish our results. The notation is hopefully clear and standard. Set, the category of sets and set-theoretic functions, will be regarded as the underlying working universe. We denote the composition of arrows f and g by $f \cdot g$. The identity on an object A is denoted by id_A (or simply A if no confusion with the object A arises). All binary operators associate to the left.

The product of two objects A and B is denoted by $A \times B$ and its sum by $A + B$. The left (resp. right) projection of a binary product is denoted by π_1 (resp. π_2) and the arrow into a binary product induced by the universal property of products (usually called the *split* of f and g) by $\langle f, g \rangle$. Dually the left (resp. right) injection of a binary sum is ι_1 (resp. ι_2) and the arrow out of a binary sum induced by the sum's universal property is denoted by $[f, g]$ (pronounced *either* f or g). The unique arrow to (resp. from) a final (resp. initial) object $\mathbf{1}$ (resp. $\emptyset$) is denoted by $!_A$ (resp. $?_A$). Finally $\Delta_A : A \longrightarrow A \times A$ is the diagonal arrow, natural in A, given by $\langle \mathrm{id}_A, \mathrm{id}_A \rangle$. The dual codiagonal $\nabla_A : A + A \longrightarrow A$ is defined as $[\mathrm{id}_A, \mathrm{id}_A]$. Subscripts will be omitted wherever implicit in the context.

Associativity, commutativity, right and left distributivity are represented by the following isomorphisms, natural in A, B and C. $\mathbf{a} : (A \times B) \times C \longrightarrow A \times (B \times C)$, $\mathbf{s} : A \times B \longrightarrow B \times A$, $\mathbf{dr} : C \times (A + B) \longrightarrow (C \times A) + (C \times B)$ and $\mathbf{dl} : (A + B) \times C \longrightarrow (A \times C) + (B \times C)$. Their inverses are, respectively, $\mathbf{a}^\circ$, $\mathbf{s}^\circ$,

$\mathbf{dr}^\circ$ and $\mathbf{dl}^\circ$. A useful derived isomorphism is $\mathbf{x} : (A \times B) \times C \longrightarrow (A \times C) \times B$, defined as $\mathbf{a}^\circ \cdot (\mathrm{id}_A \times \mathbf{s}) \cdot \mathbf{a}$.

The exponential B^A represents the arrows from A to B, with $\mathrm{ap}_{A,B} : B^A \times A \longrightarrow B$ as the evaluation morphism. The *transpose* of an arrow $f : C \times A \longrightarrow B$ is denoted by $\overline{f} : C \longrightarrow B^A$. A *partial* map from A to B is written as $A \rightharpoonup B$.

Functor composition will be denoted by juxtaposition and the identity functor on Set by Id. For the constant functor on an object A, which maps every object to A and every morphism to the identity on A, we will write simply A. Given two functors F and G, $\sigma : \mathsf{F} \Longrightarrow \mathsf{G}$ denotes a natural transformation σ.

3. MONADS. Monads play an essential role in the paper as a way to encode in abstract terms different kinds of behavioural effects. Given a monad B, its unit and multiplication are represented by the natural transformations $\eta : \mathsf{Id} \Longrightarrow \mathsf{B}$ and $\mu : \mathsf{B}\,\mathsf{B} \Longrightarrow \mathsf{B}$, respectively. Notice that η and μ act as an identity and a multiplication, respectively, and therefore a monad is the categorical counterpart of a monoid in Set. Thinking of B as the encapsulation of a behavioural effect, η_U represents the the minimal such structure whereby a value $u \in U$ is embedded in $\mathsf{B}\,U$. On the other hand, μ is a flatenning operation, providing a way to view a B-effect of a B-effect as still a B-effect.

The composition of arrows $f : I \longrightarrow \mathsf{B}\,J$ and $g : J \longrightarrow \mathsf{B}\,O$ is given by

$$g \bullet f \;=\; I \xrightarrow{f} \mathsf{B}\,J \xrightarrow{\mathsf{B}\,g} \mathsf{B}\,\mathsf{B}\,O \xrightarrow{\mu} \mathsf{B}\,O$$

This form of composition is associative, with η_A, for each A, acting as an identity, originating, for each B, its Kleisli category.

A monad is *strong* if it comes equipped with a natural transformation $\tau_{\mathbf{r}}{}^{\mathsf{B}} : \mathsf{B} \times - \Longrightarrow \mathsf{B}\,(\mathsf{Id} \times -)$, called a *right strength* verifying some coherence conditions [5]. Dually, a *left strength* is given by $\tau_{\mathbf{l}}{}^{\mathsf{B}} : - \times \mathsf{B} \Longrightarrow \mathsf{B}\,(- \times \mathsf{Id})$ obeying similar conditions. It is well known [13] that in a distributive category strength can be defined for every regular functor. A pointwise definition reads $\tau_{\mathbf{r}}{}^{\mathsf{B}}_{I,X}\langle c, x\rangle \;=\; \mu_{I\times X} \cdot \mathsf{B}\,(\lambda m \,.\, \eta_{I\times X}\langle m, x\rangle)\, c$. It has the effect of distributing the free variable values in the context along the monad.

The Kleisli composition of $\tau_{\mathbf{r}}$ and $\tau_{\mathbf{l}}$ gives rise to a correspondence $\delta_{\mathbf{r}\,I,J} : \mathsf{B}\,I \times \mathsf{B}\,J \longrightarrow \mathsf{B}\,(I \times J)$ natural in I and J, given by $\tau_{\mathbf{r}\,I,J} \bullet \tau_{\mathbf{l}\,\mathsf{B}\,I,J}$. Or, dually, $\delta_{\mathbf{l}\,I,J} = \tau_{\mathbf{l}\,I,J} \bullet \tau_{\mathbf{r}\,I,\mathsf{B}\,J}$. Such transformations specify how the monad distributes over product. Whenever they coincide, the monad is said to be *commutative*.

2. MODELING

4. COALGEBRAS. Think of an endofunctor in Set as the specification of an uniform transformation of sets and functions preserving composition and identities. It is well known that the signature of a set of constructors of a data structure D can be represented as a functor. Furthermore, the data structure itself arises as a map $t : \mathsf{T}\,D \longrightarrow D$ which specifies how values of D are built using the available constructors. There are, however, several phenomena in programming practice that are hardly definable (or even simply not definable) in terms of a complete set of constructors. They do possess an observable behaviour, but their internal configurations remain hidden and should therefore be identified if not distinguishable by observation. This is the case not only of the state based components we want to consider in this paper, but also of objects in object-oriented programming frameworks or of infinite data structures. All of them are difficult to express in a purely algebraic way, but find their way in the dual paradigm of coalgebra theory.

The slogan is "reverse the arrows": a T-*coalgebra* is simply a map $p : U \longrightarrow \mathsf{T}\,U$. One way to look at this notion is as a transition structure, of *shape* T, defined on a set U, called the *carrier* or the *state space* of p. The shape of T expresses the way the state is (partially) accessed, through *observers*, and, on the other hand, how it evolves, through *actions*. T specifies a signature of actions and observers over the carrier of p, but omits its constructors. As a consequence equality has to be replaced by *bisimilarity* (*i.e.*, indistinguishability with respect to the observation structure provided by T) and coinduction replaces induction as a proof principle.

5. COMPONENTS AS COALGEBRAS. Our first concern is precisely the shape of T. In fact, different definitions of T give rise to different models for components. One that seems to cover a broad range of cases is

$$\mathsf{T}^{\mathsf{B}} \;=\; O'^{I'} \times \mathsf{B}\,(- \times O)^{I}$$

where the sets I, I' and O, O' are, respectively, the input and output observation universes which ensure the flow of data. Each T-coalgebra p over carrier U is written as split $\langle \overline{o_p}, \overline{a_p} \rangle$, where $o_p : U \times I' \longrightarrow O'$ is the *observer*, attribute or output function, and $a_p : U \times I \longrightarrow \mathsf{B}\,(U \times O)$ stands for the coalgebra *action*, method or update function.

On the other hand, T is parametrized by a (strong) monad, B, intended to capture a particular behaviour model associated to the temporal evolution of components. Such behaviour may be purely deterministic (in which case B is instantiated with the identity monad Id) or rather more complex. By an appropriate choice for B, different behavioural features might be considered. For example,

- *Partiality*, *i.e.*, the possibility of deadlock, captured by the usual maybe monad, $\mathsf{B} = \mathsf{Id} + \mathbf{1}$.
- *Nondeterminism*, introduced by the (finite) powerset monad, $\mathsf{B} = \mathcal{P}(\mathsf{Id})$.
- Monoidal *stamping*, with $\mathsf{B} = \mathsf{Id} \times M$. Notice that, for B to form a monad the parameter M should support a monoidal structure to be used in the definition of η and μ.
- *"Metric" nondeterminism* capturing situations in which, among the possible future evolutions of the component, some are more probable (or more secure, or more ...) than others. In fact, isomorphism $\mathcal{P}(X) \cong X \rightharpoonup \mathbf{1}$ suggests the extension of the powerset to a mapping expressing a richer notion of nondeterminism in which each possible state is assigned a confidence level, or probability, leading to $X \rightharpoonup M$ (for M a monoid). Its refinement to $\mathcal{P}(X \times M)$ constitutes a monad.

By instantiating T-interface parameters with particular sets, one gets more specialized notions of a component. Instantiations with $\mathbf{1}$ are interesting as they collapse part of the observation structure. For example, making $I' = O' = \mathbf{1}$, results in $\mathsf{T}^{\mathsf{B}} = \mathsf{B}\,(- \times O)^I$, a shape for *functional components*. A coalgebra of this type is a function $p : U \longrightarrow \mathsf{B}\,(U \times O)^I$. Given a state $u \in U$ and an input stimulus $i \in I$, p computes a B-structure of possible responses. Each response is a pair formed by a new state value (representing the state evolution) and an output value returned to the process environment.

Another relevant case – in fact the one focussed in the rest of the paper — results from making $I' = O = \mathbf{1}$, giving rise to $\mathsf{T} = O \times (\mathsf{B}\,-)^I$, a shape for what could be called *object components*. What is distinctive of this case is output independence wrt input. A coalgebra of this type is a split of two functions

$$\langle o_p, \overline{a_p} \rangle : U \longrightarrow O \times (\mathsf{B}\ U)^I$$

where $o_p : U \longrightarrow O$ is the state observer (usually called the *attribute* in the object-oriented programming paradigm) and $a_p : U \times I \longrightarrow \mathsf{B}\,U$ is the state update function (usually referred to as the *method* or the process *action*). Note that, in practice, O is generally a cartesian product $\prod_{x \in X} O_x$ of different, but simultaneously available, observers, whereas I takes the form of a sum $\sum_{y \in Y} I_y$ of (state update) non interfering operations. I_y is taken as the type of the argument of operation y.

What we have called here *functional* and *object* components, correspond to a (monadic generalization) of what is known as, respectively, Mealy and Moore machines in automata literature. Other specializations of T are still interesting. For example, $\mathsf{B}\,(- \times O)$ correspond to purely active components

whose evolution is not triggered by any external stimulus, and $\mathbf{2} \times (\mathsf{B} -)^I$ to (monadic) automata in which the only information available from a state is its classification as a final or intermediate one.

A last ingredient has to be added to our model for components. Since they are not required to possess a complete set of constructors, a *seed* value for the state space is required. It represents the initial configuration. A component with input I and output O is, then, modeled as a pair $\langle u_0 \in U, p : U \longrightarrow \mathsf{T}^{\mathsf{B}}_{I,O} U \rangle$. In the sequel, $\mathsf{T}^{\mathsf{B}}_{I,O}$ denotes functor $O \times (\mathsf{B} -)^I$. Actually, the seed value could be replaced by a set of possible such values (or a correspondent predicate) with no extra burden.

6. ELEMENTARY EXAMPLES. A *Stack* is a simple example of a component with two observers (*top* and *isempty*?) and two actions (*push* and *pop*). As pop is partial, we use the maybe monad in the type of the corresponding coalgebra to force deadlock whenever an illegal pop is performed. Let E be a set and consider the state space is modeled by sequences of E. Then, define

$$Stack = \langle <> \in E^*,\ \langle o_{St}, a_{St} \rangle : E^* \longrightarrow ((E + \mathbf{1}) \times \mathbf{2}) \times (E^* + \mathbf{1})^{E+\mathbf{1}} \rangle$$

where the operations have the expected definitions:

$$\begin{array}{lll} & o_{St} & = \langle top, isempty? \rangle \\ \text{where} & top & = \lambda\, s\ .\ \text{if } s = <> \text{ then } \iota_2\ * \text{else } \iota_1(\text{hd } s) \\ & isempty? & = \lambda\, s\ .\ s = <> \\ & a_{St} & = \overline{[push, pop]} \cdot \mathbf{dl} \\ \text{where} & push & = \lambda\,(s, e)\ .\ \iota_1\,(< e > \frown s) \\ & pop & = \lambda\,(s, *)\ .\ \text{if } s = <> \text{ then } \iota_2\ * \text{else } \iota_1\,(\text{tl } s) \end{array}$$

A dummy parameter of pop (of type $\mathbf{1}$) is made explicit in the component interface and represents the *trigger* for this action.

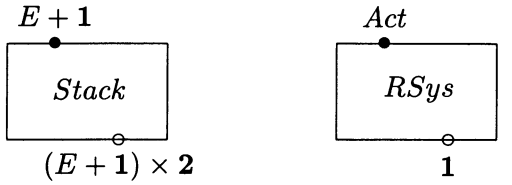
Another example of a simple component is the specification of the reactive system underlying a CCS [12] expression, for example, $R = \alpha.\beta.R + \beta.0 + \beta.R$. In this case the powerset monad is the appropriate choice and the attribute part may be considered trivial and, therefore, modeled by $\mathbf{1}$. Let Exp denote the set of CCS expressions and $Act = \{\alpha, \beta, ...\}$ the set of actions. Our specification is

$$RSys\ =\ \langle R \in Exp,\ \langle o_{RSys}, \overline{a_{RSys}} \rangle : Exp \longrightarrow \mathbf{1} \times \mathcal{P}(Exp)^{Act} \rangle$$

where $o_{RSys} = !_{Exp}$ and a_{RSys} is given by the following clauses,

$$\begin{aligned} a_{RSys}(R, \alpha) &= \{\beta.R\} \\ a_{RSys}(R, \beta) &= \{0, R\} \\ a_{RSys}(\beta.R, \beta) &= \{R\} \\ a_{RSys}(e, a) &= \emptyset \quad \text{for all other } e \in Exp \text{ and } a \in Act \end{aligned}$$

Both components can be represented in a diagrammatic form, making their input and output interfaces explicit. These are drawn, respectively, at the top and the bottom of a box:



7. MORPHISMS. A component morphism is a coalgebra morphism up to a natural transformation $\tau_{f,g}$ encoding the interface conversion, determined by functions f on the output and g on the input. Technically this amounts to a function h between the state spaces making the following diagram to commute

$$\begin{array}{ccccc} U & & \xrightarrow{\quad h \quad} & & V \\ {\scriptstyle p}\downarrow & & & & \downarrow{\scriptstyle q} \\ \mathsf{T}^{\mathsf{B}}_{I,O}U & \xrightarrow{\tau_{f,g}} & \mathsf{T}^{\mathsf{B}}_{I',O'}U & \xrightarrow{\mathsf{T}^{\mathsf{B}}_{I',O'}h} & \mathsf{T}^{\mathsf{B}}_{I',O'}V \end{array}$$

Furthermore, h has to preserve seeds, *i.e.*, $h\, u_0 = v_0$, taken u_0 and v_0 as the seeds for p and q, respectively.

We need, however, to be more explicit on the definition of $\tau_{f,g}$. For the deterministic case (*i.e.*, $\mathsf{B} = \mathsf{Id}$), it is simply $\tau = f \times \_^g$, for $f : O \longrightarrow O'$ and $g : I' \longrightarrow I$. However, the presence of a non trivial (monadic) behavioural structure calls for a broader definition. The basic observation is that each function $g : I' \longrightarrow \mathsf{B}\, I$ induces a function U_g from $(\mathsf{B}\, U)^I$ to $(\mathsf{B}\, U)^{I'}$ given by $\_ \bullet g$, where $\bullet$ is the Kleisli composition for B. U_g verifies a suitable rephrasing of the usual properties of exponentials. First of all, it is natural in U:

Proof. Let h and g as above. Then $(\mathsf{B}\, h)^{I'} \cdot U_g = (\mathsf{B}\, h \cdot \_) \cdot (\_ \bullet g) = \mathsf{B}\, h \cdot \_ \bullet g = (\_ \bullet g) \cdot (\mathsf{B}\, h \cdot \_) = V_g \cdot (\mathsf{B}\, h)^I$.

□

Furthermore, $U_\_$ is a contravariant functor from the Kleisli category for B to Set, assigning $(\mathsf{B}\, U)^I$ to an object I and U_g to a Kleisli arrow $g : I' \longrightarrow \mathsf{B}\, I$.

Clearly, $U_{\eta_I} = - \bullet g = - = \mathrm{id}_{(\mathsf{B}\,U)^I}$. On the other hand, $U_{g'} \cdot U_g = U_{g \bullet g'}$ is a direct corollary of the following more general law.

8. LEMMA. For $f : U \longrightarrow U'$ and $g : I' \longrightarrow (\mathsf{B}\,U)^I$, define $f_g : (\mathsf{B}\,U)^I \longrightarrow (\mathsf{B}\,U')^{I'}$ by

$$f_g \triangleq (\mathsf{B}\,f)^{I'} \cdot (\mathsf{B}\,U)_g = \mathsf{B}\,f \cdot \_ \bullet g$$

Now, let $f' : U' \longrightarrow U''$ and $g' : I'' \longrightarrow (\mathsf{B}\,U)^{I'}$. Then $f'_{g'} \cdot f_g = (f' \cdot f)_{g \bullet g'}$.

Proof.

$$\begin{aligned} f'_{g'} \cdot f_g &= ((\mathsf{B}\,f') \cdot \_ \bullet g') \cdot f_g \\ &= (\mathsf{B}\,f') \cdot f_g \bullet g' \\ &= (\mathsf{B}\,f') \cdot ((\mathsf{B}\,f) \cdot \_ \bullet g) \bullet g' \\ &= (\mathsf{B}\,f' \cdot f) \cdot \_ \bullet (g \bullet g') \\ &= (f' \cdot f)_{g \bullet g'} \end{aligned}$$

□

9. CATEGORIES OF COMPONENTS. Summing up the previous discussion, a morphism between B-components can be presented as a pair $\langle h, \tau_{f,g} \rangle$, with $f : O \longrightarrow O'$, $g : I' \longrightarrow \mathsf{B}\,I$, where $h : U \longrightarrow V$ is seed-preserving and makes the diagram in §7 to commute. $\tau_{f,g}$ is the natural transformation whose U component is $(\tau_{f,g})_U = f \times U_g$ with U_g defined wrt the Kleisli composition for B. Given two such arrows $\langle h, \tau_{f,g} \rangle$ and $\langle h', \tau_{f',g'} \rangle$, their composition is $\langle h' \cdot h, \tau_{f' \cdot f, g \bullet g'} \rangle$. Finally identities are defined as $\langle \mathrm{id}_U, \tau_{\mathrm{id}_O, \eta_I} \rangle$. Components and component morphisms form, therefore, a category Sr.

A crucial point is to ensure that the proposed definition for $\tau_{f,g}$ subsumes the standard case in which the monadic effect is not present. Note that, for the maybe monad, monadic g is just the classifier (or "totalizer") of a partial map $g' : I' \rightharpoonup I$. If g' is itself total then its classifier g satisfies the equation $g = \iota_1 \cdot g'$ and U_g coincides with $(U + \mathbf{1})^{g'}$. This is indeed the case for the other monads considered above. In fact, a non monadic input morphism always emerges as a special case of a monadic one. That is to say, a total map for the maybe monad, an entire and simple relation for the powerset, etc. The following lemma proves this for the general case.

10. LEMMA Let $g' : I' \longrightarrow I$ and define $g = \eta_I \cdot g'$. Then $U_g = (\mathsf{B}\,U)^{g'}$.

Proof. Let $\phi : I \longrightarrow \mathsf{B}\ U$. Then

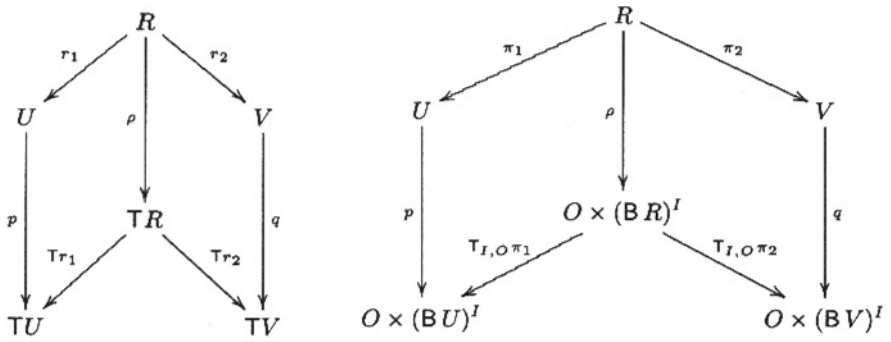
$$
\begin{aligned}
& U_g\,\phi \\
&\quad = \phi \bullet g && \{\text{ by definition }\} \\
&\quad = \phi \bullet (\eta_I \cdot g') && \{\ g \text{ is non-monadic }\} \\
&\quad = \mu \cdot \mathsf{B}\ \phi \cdot \eta_I \cdot g' && \{\ \bullet \text{ definition }\} \\
&\quad = \mu \cdot \eta_{\mathsf{B}\ U} \cdot \phi \cdot g' && \{\ \text{naturality of } \mu\ \} \\
&\quad = \phi \cdot g' && \{\ \text{monad identities }\} \\
&\quad = (\mathsf{B}\ U)^{g'}\ \phi && \{\ \text{by definition }\}
\end{aligned}
$$

□

3. COMPARING

11. BISIMULATION. When comparing software components, one intuitively identifies models which, being non isomorphic at the data level, behave in a similar way "as far as we can see". Furthermore this tends to be the key ingredient in specifications of distributed systems whose "observational contents" (or parts thereof) are *shared* by different observers.

In [14, 12] the notion of *bisimulation* was introduced in process calculi to capture this kind of observational equivalence. Later [1] gave a categorical definition of bisimulation which applies to arbitrary coalgebras (*i.e.*, bisimulation "acquired a shape"). Such a notion of T-bisimulation, for a functor T, is defined as a span $\langle R, r_1, r_2 \rangle$ whose legs lift to T-coalgebra morphisms, or, in other words, such that there is a T-coalgebra $\rho : R \longrightarrow \mathsf{T}R$ making the diagram below (on the left hand side) to commute.



In Set this can be rephrased in terms of a relation whose projections lift to T-coalgebra morphisms. The diagram on the right hand side is the corresponding instantiation for the functor $\mathsf{T}^{\mathsf{B}}_{I,O}$ underlying our model for components.

12. PROOF RULES. Let $\rho = \langle o_\rho, \overline{a_\rho} \rangle$. A simple calculation yields

$$
\begin{array}{lll}
p \cdot \pi_1 & & \\
\quad = & (O \times (\mathsf{B}\,\pi_1)^I) \cdot \rho & \{ \text{ required } \} \\
\quad = & (O \times (\mathsf{B}\,\pi_1)^I) \cdot \langle o_\rho, \overline{a_\rho} \rangle & \{ \text{ by definition } \} \\
\quad = & \langle o_\rho, (\mathsf{B}\,\pi_1)^I \cdot \overline{a_\rho} \rangle & \{ \times\text{-absorption } \} \\
\quad = & \langle o_\rho, \overline{\mathsf{B}\,\pi_1 \cdot a_\rho} \rangle & \{ \text{ exponential-absorption } \}
\end{array}
$$

and, similarly, $q \cdot \pi_2 = \langle o_\rho, \overline{\mathsf{B}\,\pi_2 \cdot a_\rho} \rangle$. A direct consequence of these equalities is the fact that, for any $\langle u, v \rangle \in U \times V$, the following equations hold:

$$
\begin{array}{rcl}
o_\rho \langle u, v \rangle & = o_p\, u = & o_q\, v \\
\overline{\mathsf{B}\,\pi_1 \cdot a_\rho} \langle u, v \rangle & = & \overline{a_p}\, u \\
\overline{\mathsf{B}\,\pi_2 \cdot a_\rho} \langle u, v \rangle & = & \overline{a_q}\, v
\end{array}
$$

Finally, we may rephrase such results as a proof rule for bisimulation, whose shape depends on the adopted behaviour monad. For the deterministic case (*i.e.*, $\mathsf{B} = \mathsf{Id}$) this yields

$$\langle u, v \rangle \in R \iff o_p\, u = o_q\, v \ \wedge\ \forall_{i \in I}\, \langle a_p \langle u, i \rangle, a_q \langle v, i \rangle \rangle \in R$$

For partial components (*i.e.*, $\mathsf{B} = \mathsf{Id} + \mathbf{1}$), on the other hand,

$$
\begin{array}{lll}
\langle u, v \rangle \in R \iff & o_p\, u = o_q\, v \ \wedge\ \forall i \in I & \\
\quad (& (a_p \langle u, i \rangle = a_q \langle v, i \rangle = \langle 2, * \rangle) & \\
\quad & \vee & \\
\quad & (a_p \langle u, i \rangle = \langle 1, u' \rangle \wedge a_q \langle v, i \rangle = \langle 1, v' \rangle \wedge \langle u', v' \rangle \in R)) &
\end{array}
$$

Finally, for the non deterministic case (*i.e.*, $\mathsf{B} = \mathcal{P}(\mathsf{Id})$) the proof rule resembles the definition of bisimulation for classical labelled transition systems,

$$
\begin{array}{lll}
\langle u, v \rangle \in R \iff & o_p\, u = o_q\, v & \\
\quad & \wedge\ \ \forall i \in I & \\
\quad (& \forall_{u' \in a_p \langle u, i \rangle} \exists_{v' \in a_q \langle v, i \rangle}\ \langle u', v' \rangle \in R & \\
\quad & \wedge & \\
\quad (& \forall_{v' \in a_q \langle v, i \rangle} \exists_{u' \in a_p \langle u, i \rangle}\ \langle u', v' \rangle \in R) &
\end{array}
$$

Notice that if p and q both denote the final coalgebra, then morphisms π_1 and π_2 coincide (by finality), ρ is an isomorphism and, therefore, bisimulation boils down to equality.

13. DEFINITION. Let p and q be components over the same interface $\langle O, I\rangle$ with $u_0 \in U$ and $v_0 \in V$ as seeds, respectively. They are said to be *bisimilar*, written $p \equiv_{\mathsf{T}_{O,I}} q$ (abbv. $p \equiv q$) iff there is a $\mathsf{T}_{O,I}$-bisimulation containing the pair $\langle u_0, v_0\rangle$.

14. ... UP TO. Now how does interface data interfere with the definition of bisimulation? Moreover, what is the effect of "monadic" input morphisms in defining bisimulation, and, therefore, relating component models? The first question leads to the definition of *bisimulation up to* $\tau_{f,g}$. The definition is well-behaved in the sense that standard results can still be rephrased in this setting. In particular, we prove in [3] that bisimulation is preserved by Sr-arrows and, moreover, the graph of a morphism $\langle h, \tau_{f,g}\rangle$ is a bisimulation up to $\tau_{f,g}$, where,

15. DEFINITION. Let p and q be components over $\langle O, I\rangle$ and $\langle O', I'\rangle$, respectively, and $\tau_{f,g} : \mathsf{T}_{I,O} \Longrightarrow \mathsf{T}_{I',O'}$ be a natural transformation. Then p and q are said to be *bisimilar up to* τ iff

$$p \equiv_\tau q \iff (\tau_{f,g} \cdot p) \equiv_{\mathsf{T}_{I',O'}} q$$

4. COMPOSING

In this section we move on to the introduction of some basic operators for composing components in category Sr. First, however, we need to define a twin category of interface spaces for components.

16. DEFINITION. For each behaviour monad B, the interface category Intf_B (abbv. Intf), has pairs of sets $\langle O, I\rangle$ as objects and pairs of functions as arrows. In particular, a morphism $l : \langle O, I\rangle \longrightarrow \langle O', I'\rangle$ is defined as $\langle f, g\rangle$, with $f : O \longrightarrow O'$ and $g : I' \longrightarrow \mathsf{B}\, I$. Composition and identities are pointwise: inherited from Set in the first component and from the Kleisli category for B in the second.

17. WRAPPING. A basic operation upon components is interface wrapping in order, for example, to restrict its use or to pre- or post- process a particular parameter. In classical process algebra this corresponds to *relabelling*. However,

the explicit consideration of input and output types in the model presented in this paper, turns it into a much more powerful operation. As illustrated below, it includes *restriction* as a particular case.

The definition of *wrapping* builds on the fundamental observation that Sr is cofibred over Intf (lemma 18). A cofibration is a functor $\mathsf{L} : \mathsf{Sr} \longrightarrow \mathsf{Intf}$ providing co-cartesian liftings of every Intf morphism $l : \langle O, I\rangle \longrightarrow \langle O', I'\rangle$ originated in the L image of each Sr-object. This means, in our context, that given a component p each interface morphism $l : \mathsf{L}p \longrightarrow \langle O', I'\rangle$ can be lifted (or *extended*) to a component morphism $l_! : p \longrightarrow q$, such that $\mathsf{L}q = \langle O', I'\rangle$ in a canonical way.

Given an interface morphism $l : \langle O, I\rangle \longrightarrow \langle O', I'\rangle$ and a component p with interface $\langle O, I\rangle$, as above, we denote p wrapped by l by $l_! p$ or, simply, $p[l]$. In the *Stack* example, for instance, one has $O = (E + \mathbf{1}) \times \mathbf{2}$ and $I = E + \mathbf{1}$. Then,

$$SimplerStack \;=\; Stack\,[\langle \pi_2, \eta_{E+\mathbf{1}}\rangle]$$

denotes a stack with no *top* attribute ($O' = \mathbf{2}$, $I' = I$). Further restricting the use of both *top* and *pop* yields

$$StrangeStack \;=\; Stack\,[\langle \pi_2, \eta_{\mathbf{1}} \cdot \iota_1\rangle]$$

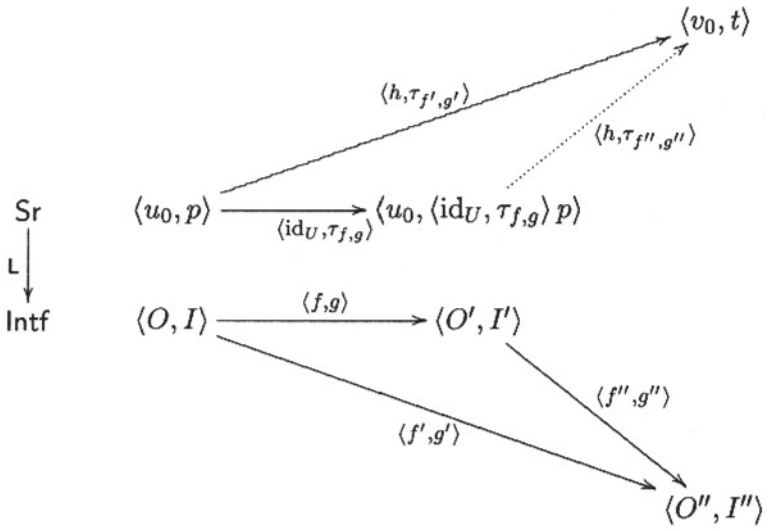
Often the input component of the wrapping morphism is a non monadic arrow, made monadic by composition with the monad unit. This is the case in the examples above. In the sequel we omit this composition for improved readability, if types are clear from the context. Furthermore, we write $p[f, g]$ as an alternative notation to $p[\langle f, g\rangle]$.

18. LEMMA. The functor $\mathsf{L} : \mathsf{Sr} \longrightarrow \mathsf{Intf}$ defined by

$$\begin{aligned} \mathsf{L}\langle u_0 \in U, p : U \longrightarrow O \times (\mathsf{B}\,U)^I\rangle &= \langle O, I\rangle \\ \mathsf{L}\langle h, \tau_{f,g}\rangle &= \langle f, g\rangle \end{aligned}$$

is a cofibration.

Proof. Consider $p = \langle u_0 \in U, p : U \longrightarrow O \times (\mathsf{B}\,U)^I\rangle$ and let $\langle f, g\rangle : \mathsf{L}p \longrightarrow \langle O', I'\rangle$ a morphism in Intf (see diagram below). We claim that its cocartesian lifting is $\langle \mathrm{id}_U, \tau_{f,g}\rangle : \langle u_0, p\rangle \longrightarrow \langle u_0, \langle \mathrm{id}_U, \tau_{f,g}\rangle\, p\rangle$.



Let $t = \langle v_0 \in V, t : V \longrightarrow O'' \times (\mathsf{B}\, V)^{I''}\rangle$ be another component and $\langle h, \tau_{f',g'}\rangle : \langle u_0, p\rangle \longrightarrow \langle v_0, t\rangle$ be an Sr-morphism. Suppose that $\langle f', g'\rangle : \langle O, I\rangle \longrightarrow \langle O'', I''\rangle$ factorizes over $\langle f, g\rangle$ in Intf through $\langle f'', g''\rangle$. Therefore composition in Intf yields $f'' \cdot f = f'$ and $g \bullet g'' = g'$. We may, then, close the upper diagram, in a unique way, with $\langle h, \tau_{f'',g''}\rangle$. In fact,

$$
\begin{aligned}
&\langle h, \tau_{f'',g''}\rangle \cdot \langle \mathrm{id}_U, \tau_{f,g}\rangle \\
&\qquad = \langle h, f'' \times U_{g''}\rangle \cdot \langle \mathrm{id}_U, f \times U_g\rangle && \{\text{ by definition }\} \\
&\qquad = \langle h, (f'' \cdot f) \times U_{g \bullet g''}\rangle && \{\text{ by lemma 8 }\} \\
&\qquad = \langle h, f' \times U_{g'}\rangle && \{\text{ by factorization in Intf }\} \\
&\qquad = \langle h, \tau_{f',g'}\rangle && \{\text{ by definition }\}
\end{aligned}
$$

□

19. PIPELINING. Another basic construction on components is *pipelining*, which corresponds to a kind of *sequential* composition. It is associative but only up to isomorphism, as the presence of internal state precludes most properties to hold up to equality.

Let $p = \langle u_0 \in U,\ p : U \longrightarrow Z \times (\mathsf{B}\, U)^I\rangle$ and $q = \langle v_0 \in V,\ q : V \longrightarrow O \times (\mathsf{B}\, V)^Z\rangle$ stand for two component models. A pipe is formed by placing them side by side and connecting the output of p to the input of q. The composed system is $p\,;q = \langle \langle u_0, v_0\rangle \in U \times V,\ p\,;q : U \times V \longrightarrow O \times (\mathsf{B}\,(U \times V))^I\rangle$

where $p\,;\,q = \langle o_{p;q}, \overline{a_{p;q}} \rangle$ is given by

$$o_{p;q} = U \times V \xrightarrow{\pi_2} V \xrightarrow{o_q} O$$

$$a_{p;q} = (U \times V) \times I \xrightarrow{\mathbf{x}} (U \times I) \times V \xrightarrow{a_p \times \mathrm{id}} \mathsf{B}\,U \times V \xrightarrow{\tau_r} \mathsf{B}\,(U \times V)$$
$$\xrightarrow{\mathsf{B}\,aux} \mathsf{B}\,\mathsf{B}\,(U \times V) \xrightarrow{\mu} \mathsf{B}\,(U \times V)$$

where

$$aux = U \times V \xrightarrow{\langle \mathrm{id}, o_p \cdot \pi_1 \rangle} (U \times V) \times Z \xrightarrow{\mathbf{a}} U \times (V \times Z)$$
$$\xrightarrow{\mathrm{id} \times a_q} U \times \mathsf{B}\,V \xrightarrow{\tau_l} \mathsf{B}\,(U \times V)$$

20. PARALLEL COMPOSITION. Unlike, *e.g.*, the cartesian product of data structures, components' parallel composition is not described by an universal construction. We have proceeded experimentally by identifying some tensors and studying their properties. We shall consider here three such structures. In all of them the composed state space is the cartesian product of the component states. They differ, however, on the allowed interaction patterns.

In the sequel let $p = \langle u_0 \in U,\ p : U \longrightarrow O \times (\mathsf{B}\,U)^I \rangle$ and $q = \langle v_0 \in V,\ q : V \longrightarrow P \times (\mathsf{B}\,V)^J \rangle$ stand for two component models.

The first two tensor products are $p \boxplus q : U \times V \longrightarrow (O \times R) \times (\mathsf{B}\,(U \times V))^{I+J}$ and $p \boxtimes q : U \times V \longrightarrow (O \times R) \times (\mathsf{B}\,(U \times V))^{I \times J}$. $p \boxplus q$ behaves either as p or q depending on input, and corresponds to what is called *external choice* in process calculi. On the other hand, $p \boxtimes q$ is a synchronous product: both processes are executed simultaneously when triggered by a pair of input values. A more "liberal" interpretation of *parallel* composition is given by another tensor product $p \circledast q : U \times V \longrightarrow (O \times R) \times (\mathsf{B}\,(U \times V))^{I \circledast J}$, where $X \circledast Y$ is given by the coproduct $X + Y + X \times Y$. The intuition is that, by putting both components side by side one gets an (observable) increase of behaviour: not only the individual observers and actions of both processes are available, but also there is the possibility of activating them concurrently (the disjointness of the two state spaces avoiding interference).

Their definition coincides on the observers

$$o_{p \boxplus q} = o_{p \boxtimes q} = o_{p \circledast q} = U \times V \xrightarrow{o_p \times o_q} O \times R$$

but does differ on actions. Therefore,

$$\begin{aligned} a_{p \boxplus q} \;=\; & (U \times V) \times (I + J) \xrightarrow{\mathbf{dr}} (U \times V) \times I + (U \times V) \times J \\ & \xrightarrow{\mathbf{x} \times \mathbf{a}} (U \times I) \times V + U \times (V \times J) \xrightarrow{a_p \times \mathrm{id} + \mathrm{id} \times a_q} \mathsf{B}\, U \times V + U \times \mathsf{B}\, V \\ & \xrightarrow{\tau_r + \tau_l} \mathsf{B}\,(U \times V) + \mathsf{B}\,(U \times V) \xrightarrow{[\mathrm{id},\mathrm{id}]} \mathsf{B}\,(U \times V) \end{aligned}$$

$$\begin{aligned} a_{p \boxtimes q} \;=\; & (U \times V) \times (I \times J) \xrightarrow{\langle\langle \pi_1 \cdot \pi_1, \pi_1 \cdot \pi_2 \rangle, \langle \pi_2 \cdot \pi_1, \pi_2 \cdot \pi_2 \rangle\rangle} (U \times I) \times (V \times J) \\ & \xrightarrow{a_p \times a_q} \mathsf{B}\, U \times \mathsf{B}\, V \xrightarrow{\delta_r} \mathsf{B}\,(U \times V) \end{aligned}$$

and

$$\begin{aligned} a_{p \circledast q} \;=\; & (U \times V) \times (I + J + I \times J) \xrightarrow{\mathbf{dr}} \\ & \longrightarrow (U \times V) \times (I + J) + (U \times V) \times (I \times J) \\ & \xrightarrow{[a_{p \boxplus q}, a_{p \boxtimes q}]} \mathsf{B}\,(U \times V) \end{aligned}$$

Moreover, units for $\boxplus$ and $\circledast$ coincide: in both cases take the component with empty input and the singleton state space, *i.e.*,

$$\boldsymbol{I}_{\boxplus} = \boldsymbol{I}_{\circledast} = \langle * \in \mathbf{1},\ \langle \mathrm{id}_{\mathbf{1}}, \mathrm{id}_{\mathbf{1}} \rangle : \mathbf{1} \longrightarrow \mathbf{1} \times (\mathsf{B}\,\mathbf{1})^{\emptyset} \rangle$$

The unit for $\boxtimes$ is $\boldsymbol{I}_{\boxtimes} = \langle * \in \mathbf{1},\ \langle \mathrm{id}_{\mathbf{1}}, \eta_{\mathbf{1}} \rangle : \mathbf{1} \longrightarrow \mathbf{1} \times (\mathsf{B}\,\mathbf{1})^{\mathbf{1}} \rangle$. Note that the use of $\boldsymbol{\delta_r}$ or $\boldsymbol{\delta_l}$ in the definition of $\boxtimes$ is irrelevant for B a commutative monad. On the other hand, choosing $\mathsf{B} = \mathsf{Id}^*$ leads to two alternative versions of synchronous product. Commutativity thus seems to be a reasonable requirement to impose on behaviour monads.

21. HOOK. Component *interaction* is not captured by any of the tensor products above. However the kind of plugging done in sequential composition may be generalized to a *hook* operation connecting some input to some output points.

Depending on the shape of the input interface, we consider an *effective* or a *delayed* hook. The first case assumes the argument is modeled by a coalgebra $p : U \longrightarrow (O \times Z) \times (\mathsf{B}\,U)^{I+Z}$. Define $(p)_Z$ as a coalgebra with the same

signature and observer which feeds back part of the output as follows:

$$a_{(p)_Z} = U \times (I + Z) \xrightarrow{a_p} \mathsf{B}\,U \xrightarrow{\mathsf{B}\,aux} \mathsf{B}\,\mathsf{B}\,U \xrightarrow{\mu} \mathsf{B}\,U$$

where

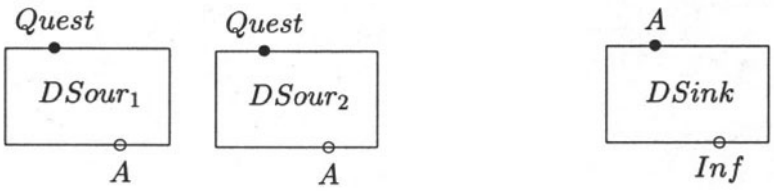
$$aux = U \xrightarrow{\langle \mathrm{id}, \pi_2 \cdot o_p \rangle} U \times Z \xrightarrow{\mathrm{id} \times \iota_2} U \times (I + Z) \xrightarrow{a_p} \mathsf{B}\,U$$

A *delayed* hook applies to cases in which the required input is a tuple whose second component comes from the (previous) state. Note that the common parameter becomes hidden. Formally, let $p : U \longrightarrow (O \times Z) \times (U + \mathbf{1})^{I \times Z}$. Define $_Z(p)$ as

$$o_{Z(p)} = \pi_1 \cdot o_p$$

$$a_{Z(p)} = U \times I \xrightarrow{\langle \mathrm{id}, \langle \pi_2, o_p \cdot \pi_1 \rangle \rangle} U \times (I \times Z) \xrightarrow{a_p} \mathsf{B}\,U$$

22. EXAMPLE. In order to illustrate the use of the connectives just introduced, consider a distributed querying system in which a question (modeled by a type $Quest$) is simultaneously placed to several independent data sources, each of which supplies a possible answer (of type A). A special component acts as an answer collector, merging all the answers produced and computing a final result by means of some pre-defined algorithm. Let us start with two data sources (the n-ary case is dealt similarly) $DSour_1$ and $DSour_2$ as well as a concentrator $DSink$ acting as a data sink. Note that $DSour_1$ and $DSour_2$ may have different definitions and need not to be deterministic. A typical choice for B in this example would be $\mathsf{B} = \mathcal{P}(- \times R)$, where R is the confidence level assigned to a given answer.



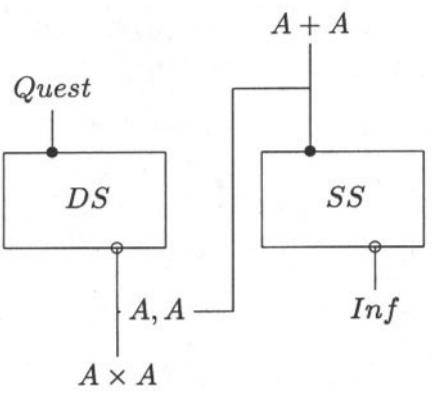
We begin by constructing the synchronous product $DSour_1 \boxtimes DSour_2$ with input $Quest \times Quest$ and output $A \times A$. As the same question is to be placed simultaneously to both data sources, this input has to be reduced by wrapping the combined process with the diagonal function, *i.e.*,

$$DS = (DSour_1 \boxtimes DSour_2)[\mathrm{id}, \Delta]$$

Next DS is composed with $DSink$ via $\boxplus$ and the output of the former fed back into the later, through a double application of *hook*. For this to be possible, however, the input of $DSink$ has to be expanded to a coproduct of A, which is achieved by wrapping with the appropriate codiagonal

$$SS = DSink\,[\mathrm{id}, \nabla]$$

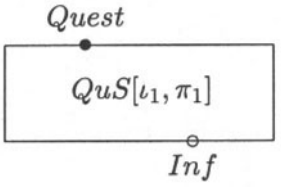
Some additional *wiring* is required to apply the *hook* operator: the output interface of $DS \boxplus SS$ is $(A \times A) \times Inf$ and, consequently has to be changed to $Inf \times (A \times A)$ by wrapping with the **s** isomorphism. The result is pictured in the diagram below.



The aggregated system is, therefore, given by

$$QuS \;=\; ((DS \boxplus SS)[\mathrm{id}, \mathbf{s}])_{A,A}$$

Finally, the intermediate answers are hidden from the environment, yielding the final querying system as a new component pictured as follows



23. FUNCTIONS. Functional components, *i.e.*, components representing pure functions, are encoded in Sr by *output mirror*. Let $\phi : A \longrightarrow B$. Its encoding as an Sr object is

$$\ulcorner \phi \urcorner \;=\; \langle b \in B, \langle \mathrm{id}_B, \overline{\eta_B \cdot \phi \cdot \pi_2} \rangle \rangle$$

whose coalgebra is of type $B \longrightarrow B \times (\mathsf{B}\, B)^A$.

Note, however, that sequential composition in Sr does not specialize to the usual functional composition. The reason is that the state space in $\ulcorner \phi \urcorner$ cannot be discarded. Moreover, the need to choose a seed value makes the representation of ϕ not unique. $\ulcorner \phi \urcorner$, for any ϕ, satisfies, however, an *almost seed irrelevance* property: given two encodings of ϕ with different seeds, for example, $p_1 = \langle b_1, p \rangle$ and $p_2 = \langle b_2, p \rangle$ ($p = \langle \mathrm{id}_B, \overline{\eta_B \cdot \phi \cdot \pi_2} \rangle$), they become bisimilar in the step following the initial one.

To see this we have first to formalize a weaker notion of bisimilarity, which could be called *next step bisimilarity*. The intuition is that two components p and q are in such a relation if each pair of successor states of p and q is contained in a $\mathsf{T}_{I,O}$-bisimulation. Given a $\mathsf{T}_{I,O}$ coalgebra p, over a state space U, and a state $u \in U$ the (possibly empty) set of successors of u is denoted by $\bullet_p^{\mathsf{B}\, X}\{u\}$ and given by (un)lifting the set

$$\{\alpha(i) |\ i \in I \ \wedge\ \alpha = \pi_2 \cdot p(u)\}$$

wrt functor B. Technically, this computation is left adjoint to the predicate lifting introduced in [6], for (extended) polynomial functors (the whole topic is addressed in [9] in the broader context of modal languages induced by coalgebras). Omitting, for the sake of brevity, the calculational details, we just present $\bullet_p^{\mathsf{B}}$ for the two familiar cases of the maybe and the powerset monads.

$$\bullet_p^{X+\mathbf{1}}\{u\} = \{x \in U |\ \iota_1 x \in \{(\pi_2 \cdot p(u))(i) |\ i \in I\}\}$$
$$\bullet_p^{\mathcal{P}(X)}\{u\} = \bigcup\{(\pi_2 \cdot p(u))(i) |\ i \in I\}$$

The announced definition is as follows:

24. DEFINITION. Let p and q be components over the same interface $\langle O, I \rangle$ and behaviour monad B, with u_0 and v_0 as seeds, respectively. They are said to be *next step bisimilar* (written $p \equiv_\bullet q$) iff

$$\begin{array}{l} \bullet_p^{\mathsf{B}}\ \{u_0\} = \bullet_q^{\mathsf{B}}\{v_0\} = \emptyset \\ \vee \\ (\forall_{u \in \bullet_p^{\mathsf{B}}\{u_0\}} \exists_{v \in \bullet_q^{\mathsf{B}}\{v_0\}} \cdot\ \langle u, p \rangle \equiv \langle v, q \rangle \\ \ \wedge \\ \forall_{v \in \bullet_q^{\mathsf{B}}\{v_0\}} \exists_{u \in \bullet_p^{\mathsf{B}}\{u_0\}} \cdot\ \langle u, p \rangle \equiv \langle v, q \rangle) \end{array}$$

25. LEMMA. Let ϕ_1 and ϕ_2 be two encodings of a function $\phi : A \longrightarrow B$ over different seeds b_1 and b_2. Then $\phi_1 \equiv_\bullet \phi_2$.

Proof. It is enough to show that the sets of successor states in both cases coincide. If they are both empty we are done; otherwise take the identity relation to trivially witness all the required bisimulations. Denoting, as in [9], the unlifting of a predicate *wrt* a functor B by $(-)_{\mathsf{B}} : \mathcal{P}(\mathsf{B}-) \longrightarrow \mathcal{P}(-)$, a simple calculation yields

$$\begin{aligned}
\bullet^{\mathsf{B}}_{\phi_1}\{b_1\} &= (\{\alpha\, a |\ a \in A \ \wedge\ \alpha = (\pi_2 \cdot \phi_1)\, b_1\})_{\mathsf{B}} \\
&= (\{\alpha\, a |\ a \in A \ \wedge\ \alpha = (\pi_2 \cdot \langle \mathrm{id}_B, \overline{\eta_B \cdot \phi \cdot \pi_2}\rangle) b_1\})_{\mathsf{B}} \\
&= (\{\alpha\, a |\ a \in A \ \wedge\ \alpha = (\overline{\eta_B \cdot \phi \cdot \pi_2})\, b_1\})_{\mathsf{B}} \\
&= (\{(\eta_B \cdot \phi \cdot \pi_2)\, \langle b_1, a\rangle |\ a \in A\})_{\mathsf{B}} \\
&= (\{(\eta_B \cdot \phi)\, a |\ a \in A\})_{\mathsf{B}} \\
&= (\{(\eta_B \cdot \phi \cdot \pi_2)\, \langle b_2, a\rangle |\ a \in A\})_{\mathsf{B}} \\
&= \bullet^{\mathsf{B}}_{\phi_2}\{b_2\}
\end{aligned}$$

□

5. CONCLUDING REMARKS

26. RELATED AND FUTURE WORK. This paper presented a coalgebraic model for software components and defined some core operators of a component algebra. Components and component morphisms are shown to form a category Sr. This definition allows for monadic input morphisms to be absorbed by the component behavioural model. Furthermore, we prove that Sr is cofibred over the interfaces.

Further properties of the operators as well as alternatives to the basic model are discussed in [3]. For example, a simple extension captures the presence of *internal* actions which allow the component to evolve without being triggered by the environment.

Complementary research, described in [2], concerns the *prototyping* of component's behaviours in the non strict functional language CHARITY [4, 18], by computing the anamorphic image of their seeds. In fact, the behaviour of a component abstracts over all internal states. For a $\mathsf{T}^{\mathsf{B}}_{I,O}$ component, this is represented (or encapsulated) in the image of its seed under the unique arrow (the anamorphism) from p to the $\mathsf{T}^{\mathsf{B}}_{I,O}$ final coalgebra. The actual way in which the anamorphism is computed (and the behaviour revealed) resorts to lazy evaluation. In each step, activation of a new continuation structure is returned upon which experimentation proceeds.

A current research topic concerns the *refinement* of this kind of coalgebras with structured input and output, which involves a weaker notion of coalge-

bra morphism. In fact, not one, but several notions of refinement arise in a construction which is, again, parametric on the behaviour monad.

27. ACKNOWLEDGEMENTS. This work has been partially supported by LOGCOMP under PRAXIS XXI contract 2/2.1/TIT/1658/95.

References

[1] P. Aczel and N. Mendler. A final coalgebra theorem. In D. Pitt, D. Rydeheard, P. Dybjer, A. Pitts, and A. Poigne, editors, *Proc. Category Theory and Computer Science*, pages 357–365. Springer Lect. Notes Comp. Sci. (389), 1988.

[2] L. S. Barbosa. Prototyping processes. In M. C. Meo and M. Vilares Ferro, editors, *Proc. of AGP'99 - Joint Conference on Declarative Programming*, pages 513–527, L'Aquila, Italy, 6-9 September 1999.

[3] L. S. Barbosa. Forthcoming thesis. DI, Universidade do Minho, 2000.

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

[5] Robin Cockett and Dwight Spencer. Strong categorical datatypes I. In R. A. G. Seely, editor, *Proceedings of Int. Summer Category Theory Meeting, Montreal, Quebec, 23–30 June 1991*, pages 141–169. AMS, CMS Conf. Proceedings 13, 1992.

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

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

[8] B. Jacobs. Objects and classes, co-algebraically. In C. Lengauer B. Freitag, C.B. Jones and H.-J. Schek, editors, *Object-Orientation with Parallelism and Persistence*, pages 83–103. Kluwer Acad. Publ., 1996.

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

[10] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *EATCS Bulletin*, 62:222–159, 1997.

[11] Cliff B. Jones. *Systematic Software Development Using* VDM. Series in Computer Science. Prentice-Hall International, 1986.

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

[13] A. Pardo. A calculational approach to strong datatypes. In *Selected Papers from 8th Nordic Workshop on Programming Theory*. Research Report 240, Oslo, 1996.

[14] D. Park. Concurrency and automata on infinite sequences. pages 561–572. Springer Lect. Notes Comp. Sci. (154), 1981.

[15] H. Reichel. An approach to object semantics based on terminal co-algebras. *Math. Struc. Comp. Sci.*, (5):129–152, 1995.

[16] J. Rutten. Universal coalgebra: A theory of systems. Technical Report CS-R9636, CWI, Amsterdam, 1996. (to appear in Theor. Comp. Sci.).

[17] J. Rutten and D. Turi. Initial algebra and final co-algebra semantics for concurrency. In *Proc. REX School: A Decade of Concurrency*, pages 530–582. Springer Lect. Notes Comp. Sci. (803), 1994.

[18] Dwight L. Spencer. *Categorical Programming with Functorial Strength.* PhD thesis, The Oregon Graduate Institute of Science and Technology, January 1993.

[19] J. M. Spivey. *The Z Notation: A Reference Manual.* Series in Computer Science. Prentice-Hall International, 1989.

[20] D. Turi and J. Rutten. On the foundations of final coalgebra semantics: non-well-founded sets, partial orders, metric spaces. *Math. Struc. Comp. Sci.*, 8(5):481–540, 1998.