Components as Coalgebras: the Refinement Dimension [⋆]

Sun Meng [a], Luís S. Barbosa [b]

[a] *LMAM, School of Mathematical Science, Peking University, China*

[b] *Department of Informatics, Minho University, Portugal*

Abstract

This paper characterizes refinement of state-based software components modeled as pointed coalgebras for some Set endofunctors. The proposed characterization is parametric on a specification of the underlying behaviour model introduced as a strong monad. This provides a basis to reason about (and transform) state-based software designs. In particular it is shown how refinement can be applied to the development of the inequational subset of a calculus of generic software components.

Key words: software components, refinement, coalgebra

1 Introduction

In the tradition of mathematical modelling in physics and chemistry, constructive formal specification methods, such as VDM [1], Z [2] or B [3], are based on the notion of a *software model*, understood as a state-based abstract machine which persists and evolves in time, according to a behavioural model capturing, for example, *partiality* or (different degrees of) *non determinism.* This can be identified with the more prosaic notion of a *software component* [4,5] advocated by the software industry as 'building block' of large, often

⋆ Research supported by FCT (the Portuguese Foundation for Science and Technology), in the context of the PURe Project under contract `POSI/ICHS/44304/2002`. Sun Meng was further supported by the National Natural Science Foundation of China under grant no. 60273001. The authors wish to thank José Oliveira for useful feedback on an earlier version of this work and the anonymous referees who also provided a number of helpful suggestions.

Email addresses: `sunmeng@water.pku.edu.cn` (Sun Meng), `lsb@di.uminho.pt` (Luís S. Barbosa).

distributed, systems. Such a component typically encapsulates a number of services through a public interface which provides a limited access to a private state space, paying tribute to the nowadays widespread object-oriented programming principles.

Regarded as state-based, dynamical systems, software components belong to the broad group of computing phenomena which are hardly definable (or simply not definable) algebraically, *i.e.*, in terms of a complete set of constructors. Their semantics is essentially *observational*, in the sense that all that can be traced of their evolution is their *interaction* with the environment. Therefore, *coalgebras*, whose theory has recently witnessed remarkable developments [6], appear as a suitable modelling tool. Such was the starting point of a coalgebraic approach to the semantics of state-based software components proposed by the authors in [7,8], under the slogan *components as coalgebras*. This research has been driven by two key ideas: first, the 'black-box' characterization of components favours an *observational* semantics; secondly, the proposed constructions should be *generic* in the sense that they should not depend on a particular notion of component behaviour. This led both to the adoption of *coalgebra* theory to capture observational semantics and to the abstract characterization of possible behaviour models (ranging from partiality to non determinism) by *strong monads* acting as parameters in the resulting calculus.

Within this approach, briefly reviewed in section 2, a set of component connectors have been identified and their properties established as *bisimilarity* equations with respect to a generic behaviour model. Actually, the corner stone of the resulting calculus is the use of coinduction to prove $\sim$-results, where $\sim$ is the appropriate bisimilarity relation, when reasoning about and transforming component-based designs. The aim of this paper is to provide a basis to extend the calculus toward the *inequational* side, while retaining its genericity, through the introduction of suitable notions of *refinement*.

But what is *component refinement*? In broad terms *refinement* can be defined as a *transformation* of an 'abstract' into a more 'concrete' design, entailing a notion of *substitution*, but not necessarily *equivalence*. There is, however, a diversity of ways of understanding both what *substitution* means, and what such a *transformation* should seek for. In *data refinement*, for example, after Hoare's landmark paper [9], the 'concrete' model is required to have *enough redundancy* to represent all the elements of the 'abstract' one. This is captured by the definition of a surjection from the former into the latter (the *abstraction* or *retrieve map*). Also *substitution* is regarded as 'complete' in the sense that the (concrete) operations accept all the input values accepted by the corresponding abstract ones, and, for the same inputs, the results produced are also the same, up to the abstraction map. This means that, if models are specified, as they usually are in *constructive* design methods, in terms of pre and post-conditions, the former are weakened and the latter strengthened, under refinement. In *object-orientation*, on the other hand, *substitution* is expressed in terms of *behaviour subtyping* [10] capturing the idea that 'concrete' objects behave similarly to objects in the 'abstract' class. Finally, refinement in *process algebras* is usually discussed in terms of a number of 'observation' preorders (see, for example, [11]), most of them justifying transformations entailing *reduction of non determinism.*

In general, refinement correctness means that the usage of a system according to its 'abstract' description is still valid if it is actually built according to the 'concrete' one. What is commonly understood by being a *valid usage* is that the corresponding observable consequences are still the same, or, in a less strict sense, a subset thereof. The exact definition, however, depends on the underlying *behaviour model*, which, in our approach to component modelling, is taken as a specification parameter. Therefore, the main contribution of this paper is a semantic characterisation of refinement for state-based components, parametric on a strong monad intended to capture components' behavioural models.

After a brief review of the equational calculus, in section 2, the paper discusses two levels of component refinement: the *interface* level, concerned with what one may call *plugging compatibility*, in section 3, and the *behavioural* one in section 4, which introduces *forward* and *backward* morphisms as refinement 'witnesses', and section 5 which builds on them to propose a family of refinement preorders. The extension of the component calculus with refinement inequations is illustrated in section 6. Section 7 proves soundness of *simulations* to establish behavioural refinement and, finally, some prospects for future work are presented in section 8.

2 Components as Coalgebras

As mentioned above, software components and connectors have been characterised in [7,8] as dynamical systems with a public interface and a private, encapsulated state. As an example consider LBuff: a connector modelling a buffered channel which occasionally loses messages, as represented in figure 1.

The put and pick operations are regarded as 'buttons' or 'ports', whose signatures are grouped together in the diagram (M stands for a message parameter type, $\mathbf{1}$ for the nullary datatype and $+$ for 'datatype sum'). One might capture LBuff dynamics by a function $a_{\mathsf{LBuff}} : \mathcal{P}(U \times O) \longleftarrow U \times I$ where U denotes the state space. This describes how LBuff reacts to input stimuli, produces output data (if any) and changes state. It can also be written in a curried form as $\overline{a}_{\mathsf{LBuff}} : \mathcal{P}(U \times O)^{I} \longleftarrow U$ that is, as a *coalgebra* of signature $U \longleftarrow \mathsf{T}\, U$ where

$$\begin{cases} \mathsf{put}: & M \longrightarrow \mathbf{1} \\ \mathsf{pick}: & \mathbf{1} \longrightarrow M \end{cases}$$

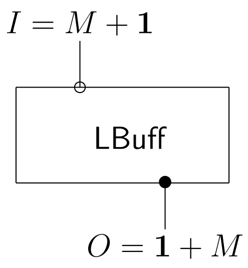

Fig. 1. The LBuff component.

functor T captures the transition 'shape':

$$\mathsf{T} \; = \mathcal{P}(\mathsf{Id} \times O)^I \tag{1}$$

Built in this 'shape' is the possibility of non deterministic evolution captured by the use of $\mathcal{P}$, the finite powerset monad. Concretely, LBuff is defined over $U = M^*$, with nil as the initial state, and dynamics given by

$$\begin{aligned} a_{\mathsf{LBuff}}\langle u, \mathsf{put}\ m\rangle &= \{\langle u, \iota_1\ *\rangle, \langle m : u, \iota_1\ *\rangle\} \\ a_{\mathsf{LBuff}}\langle u, \mathsf{pick}\rangle &= \{\langle \mathsf{tl}\ u, \iota_2\ (\mathsf{hd}\ u)\rangle\} \end{aligned}$$

where put m and pick abbreviate $\iota_1\ m$ and $\iota_2\ *$, respectively.

Non determinism, capturing the occasional loss of messages, is a possible behavioural pattern for this buffer, but, by no means, the only one. Other components will exhibit different *behaviour models*: actually *genericity* is achieved by replacing the *powerset* monad above, by an arbitrary *strong monad*[1] B. In the general case, a component p with input interface I and output interface O, denoted by $p : O \longleftarrow I$, is specified as a (pointed) coalgebra in Set

$$\langle u_p \in U_p, \overline{a}_p : \mathsf{B}(U_p \times O)^I \longleftarrow U_p\rangle \tag{2}$$

[1] A *strong monad* is a monad $\langle \mathsf{B}, \eta, \mu\rangle$ where B is a strong functor and both η and μ strong natural transformations. B being strong means there exist natural transformations $\mathsf{T}(\mathsf{Id} \times -) : \mathsf{T} \times - \Longleftarrow \mathsf{T} \times -$ and $\mathsf{T}(- \times \mathsf{Id}) : - \times \mathsf{T} \Longleftarrow - \times \mathsf{T}$ called the right and left strength, respectively, subject to certain conditions. Their effect is to *distribute* the free variable values in the context "$-$" along functor B. Strength τ_r, followed by τ_l maps $\mathsf{B}I \times \mathsf{B}J$ to $\mathsf{BB}(I \times J)$, which can, then, be flattened to $\mathsf{B}(I \times J)$ via μ. In most cases, however, the *order* of application is relevant for the outcome. The Kleisli composition of the right with the left strength, gives rise to a natural transformation whose component on objects I and J is given by $\delta r_{I,J} \; = \; \tau_{r_{I,J}} \bullet \tau_{l_{\mathsf{B}I,J}}$ Dually, $\delta l_{I,J} = \tau_{l_{I,J}} \bullet \tau_{r_{I,\mathsf{B}J}}$. Such transformations specify how the monad distributes over product and, therefore, represent a sort of sequential composition of B-computations. Whenever δ_r and δ_l coincide, the monad is said to be *commutative.*

where point u_p is taken as the 'initial' or 'seed' state. Therefore, the computation of an action will not simply produce an output and a continuation state, but a B-structure of such pairs. The monadic structure provides tools to handle such computations. Unit (η) and multiplication (μ), provide, respectively, a value embedding and a 'flatten' operation to reduce nested behavioural annotations. Strength, either in its right (τ_r) or left (τ_l) version, will cater for context information.

In such a framework, components become *arrows* in a (bicategorical) universe Cp whose objects are sets, providing types to input/output parameters (the components' *interfaces*). Component morphisms $h : q \longleftarrow p$, which impose a categorical structure on Cp homsets, are functions relating the state spaces of p and q and satisfying the following homomorphism conditions:

$$h\, u_p \;=\; u_q \qquad \text{and} \qquad \overline{a}_q \cdot h \;=\; \mathsf{B}\,(h \times O)^I \cdot \overline{a}_p \tag{3}$$

From the applicational point of view, component morphisms provide a basis for component comparison.

For each triple of objects $\langle I, K, O\rangle$, a composition law is given by functor $;_{I,K,O} : \mathsf{Cp}(I,O) \longleftarrow \mathsf{Cp}(I,K) \times \mathsf{Cp}(K,O)$ whose action on objects p and q is

$$p\,;q \;=\; \langle \langle u_p, u_q\rangle \in U_p \times U_q, \overline{a}_{p;q}\rangle \qquad \text{with}$$

$$\begin{aligned} a_{p;q} = U_p \times U_q \times I &\xrightarrow{\cong} U_p \times I \times U_q \xrightarrow{a_p \times \mathsf{id}} \mathsf{B}(U_p \times K) \times U_q \\ &\xrightarrow{\tau_r} \mathsf{B}(U_p \times K \times U_q) \xrightarrow{\cong} \mathsf{B}(U_p \times (U_q \times K)) \\ &\xrightarrow{\mathsf{B}(\mathsf{id} \times a_q)} \mathsf{B}(U_p \times \mathsf{B}(U_q \times O)) \xrightarrow{\mathsf{B}\tau_l} \mathsf{BB}(U_p \times (U_q \times O)) \\ &\xrightarrow{\cong} \mathsf{BB}(U_p \times U_q \times O) \xrightarrow{\mu} \mathsf{B}(U_p \times U_q \times O) \end{aligned}$$

This law is just a generalisation of functional composition to take into account both the presence of state and a generic behaviour model B. Similarly, for each object K, an identity law is given by a functor $\mathsf{copy}_K : \mathsf{Cp}(K,K) \longleftarrow \mathbf{1}$ whose action is the constant component $\langle * \in \mathbf{1}, \eta_{\mathbf{1}\times K}\rangle$. Note that the definitions above rely solely on the monadic structure of B. All in all, the fact that, for each strong monad B, components form a bicategory amounts not only to a canonical definition of the two basic combinators ($;$ and copy_K), but also to set up their basic laws. Recall (from *e.g.* [6]) that the graph of a morphism is a bisimulation. Therefore, the existence of a seed preserving morphism between two components makes them bisimilar, leading to the following laws, for appropriately typed components p, q and r:

$$\mathsf{copy}_I\,; p \;\sim\; p \;\sim\; p\,; \mathsf{copy}_O \tag{4}$$

$$(p\,; q)\,; r \;\sim\; p\,; (q\,; r) \tag{5}$$

The dynamics of a component specification is essentially 'one step': it describes immediate reactions to possible state/input configurations. Its temporal extension becomes the component's *behaviour*. Formally, behaviour $[\![p]\!]$ of a component p is computed by *coinductive extension*, applying the seed-value of p to the unique morphism (denoted by $[\!(\overline{a}_p)\!]$) from its dynamics $\overline{a}_p$ to the final coalgebra:

$$[\![p]\!] \;=\; [\!(\overline{a}_p)\!] u_p$$

Again behaviours organise themselves in a category Bh, whose objects are sets and arrows $b : O \longleftarrow I$ elements of $\nu_{I,O}$, the carrier of the final coalgebra $\omega_{I,O}$ for functor $\mathsf{B}(\mathsf{Id} \times O)^I$.

It should be observed that the structure of Bh mirrors whatever structure Cp possesses. In fact, the former is isomorphic to a sub-(bi)category of the latter whose arrows are components defined over the corresponding final coalgebra. Alternatively, we may think of Bh as the quotient Cp by the greatest bisimulation. However, as final coalgebras are fully abstract with respect to bisimulation, the bicategorical structure collapses. This is why properties holding in Cp up to bisimulation, do hold 'on the nose' in Bh.

In [7,8] a set of component *combinators* has been defined in a similar parametric way and their properties studied. In particular it was shown that any function $f : B \longleftarrow A$ can be lifted to Cp as

$$\ulcorner f \urcorner = \langle * \in \mathbf{1}, \eta_{(\mathbf{1} \times B)} \cdot (\mathsf{id} \times f) \rangle$$

The pre- and post-composition of a component with Cp-lifted functions can be encapsulated into an unique combinator, called *wrapping*, which resembles the *renaming* connective found in process calculi (*e.g.*, [12]). Let $p : O \longleftarrow I$ be a component and consider functions $f : I \longleftarrow I'$ and $g : O' \longleftarrow O$. Notation $p[f, g]$ denotes component p wrapped by f and g, which is typed as $I' \longrightarrow O'$.

Formally, the wrapping combinator is a functor $-[f, g] : \mathsf{Cp}(I', O') \longleftarrow \mathsf{Cp}(I, O)$ which is the identity on morphisms and maps a component $\langle u_p, \overline{a}_p \rangle$ into $\langle u_p, \overline{a}_{p[f,g]} \rangle$, where

$$a_{p[f,g]} \;= U_p \times I' \xrightarrow{\mathsf{id} \times f} U_p \times I \xrightarrow{a_p} \mathsf{B}(U_p \times O) \xrightarrow{\mathsf{B}(\mathsf{id} \times g)} \mathsf{B}(U_p \times O')$$

Function lifting and wrapping verify a number of laws of which the following two will be needed in the sequel:

$$\ulcorner f \urcorner ; \ulcorner g \urcorner \sim \ulcorner g \cdot f \urcorner \tag{6}$$
$$p[f, g] \sim \ulcorner f \urcorner ; p ; \ulcorner g \urcorner \tag{7}$$

Typical component assembly patterns are modelled by three tensors, capturing, respectively, *external choice* ($\boxplus$), *parallel* ($\boxtimes$) and *concurrent* ($\boxast$) composition. Let $p : O \longleftarrow I$ and $q : R \longleftarrow J$. When interacting with $p \boxplus q : O+R \longleftarrow I+J$, the environment chooses either to input a value of type I or one of type J, which triggers the corresponding component (p or q, respectively), producing the relevant output. In its turn, *parallel* composition $p \boxtimes q : O \times R \longleftarrow I \times J$ corresponds to a synchronous product: both components are executed simultaneously when triggered by a pair of legal input values. Note, however, that the behaviour effect, captured by monad B, propagates. For example, if B captures component failure and one of the arguments fails, the product will fail as well. Finally, *concurrent* composition $p \boxast q : O + R + O \times R \longleftarrow I + J + I \times J$ combines choice and parallel, in the sense that p and q can be executed independently or jointly, depending on the input supplied. Generalised *interaction* is catered through a sort of 'feedback' mechanism connecting a specified subset of outputs to a subset of inputs of the same component. Therefore arbitrary communication between components is achieved by first aggregating them through one of the tensors and then selecting the input and output points to be connected by the feedback operator.

3 Interface Refinement

Component *interface refinement* is concerned with type compatibility. The question is whether a component can be transformed, by suitable wiring, to replace another component with a different interface. As the structure of components' interface types encodes the available operations, this may capture situations of *extension of functionality*, in the sense that the 'concrete' component may introduce new operations. In the context of object-orientation, this is often called *design sophistication* (rather than *refinement*). In general a preorder capturing functionality extension fails to be a pre-congruence with respect to typical process combinators (see *e.g.*, [13]). If input and output parameters are structured in a signature of operations, interface refinement can also be seen as induced by a signature morphism, as in *e.g.*, [14].

As a first example, consider, from [7], the following law expressing commutativity of *choice*:

$$p \boxplus q \;\sim\; (q \boxplus p)[\mathsf{s}_+, \mathsf{s}_+] \tag{8}$$

where $\mathsf{s}_+ : J + I \longleftarrow I + J$ is the natural isomorphism capturing $+$ commutativity. The law states that $p \boxplus q$ and $q \boxplus p$ are bisimilar *up to isomorphic wiring.* This means that the observational effect of component $p \boxplus q$ can be achieved by $q \boxplus p$, provided the interface of the latter is converted to the interface of the former. Such a conversion is achieved by composition with the appropriate *wires* [2], leading to a notion of *replaceability.*

Definition 1 *Let p and q be components. We say that $p : O \longleftarrow I$ is* replaceable *by $q : O' \longleftarrow I'$, or q is a* replacement *of p, and write $p \lessdot q$ if there exist functions $w_1 : I' \longleftarrow I$ and $w_2 : O \longleftarrow O'$, to be referred to as the replacement* witnesses*, such that*

$$p \;\sim\; q[w_1, w_2] \tag{9}$$

Furthermore, components p and q are interchangeable *if each of them is a replacement of the other. Formally,*

$$p \doteqdot q \;\;\textsf{iff}\;\; p \lessdot q \;\wedge\; q \lessdot p \tag{10}$$

Lemma 2 *Replaceability ($\lessdot$) is a preorder on components*

PROOF. Clearly, $\lessdot$ is reflexive because $p \lessdot p$ is witnessed by $p \sim p[\mathsf{id}, \mathsf{id}]$. On the other hand, if $p \lessdot q$ and $q \lessdot r$ hold, there exist w_1, w_2, w_3 and w_4 such that $p \sim q[w_1, w_2]$ and $q \sim r[w_3, w_4]$. Thus, a composition result on wrapping [8] and transitivity of $\sim$, entails $p \sim r[w_1 \cdot w_3, w_4 \cdot w_2]$, *i.e.*, $p \lessdot r$.

□

Using $\lessdot$ and $\doteqdot$ some component laws in [8] can be presented in a 'wiring free' form. For example, law (8) above becomes

$$p \boxplus q \;\doteqdot\; q \boxplus p \tag{11}$$

As another example consider the law which relates *concurrent* composition with *choice*,

$$\ulcorner \iota_1 \urcorner \,;\, (p \boxast q) \sim (p \boxplus q) \,;\, \ulcorner \iota_1 \urcorner$$

which gives rise to two replacement inequations:

$$\ulcorner \iota_1 \urcorner \,;\, (p \boxast q) \;\lessdot\; p \boxplus q \quad \text{and} \quad (p \boxplus q) \,;\, \ulcorner \iota_1 \urcorner \;\lessdot\; p \boxast q$$

Finally, the statement that every component p can replace an *inert* component may be expressed as an interface refinement situation: $\textsf{inert} \;\lessdot\; p$. Note that the $\textsf{inert}$ component, which is unable to react to any external stimulus, corresponds to the lifting of the canonical mediating arrow $?_{\mathbf{1}} = !_{\emptyset} : \mathbf{1} \longleftarrow \emptyset$. Also

[2] *Wires* are components over $\mathbf{1}$ defined from identities using only the structural properties of the underlying category — *i.e.*, arrows associated to products, coproducts and exponentials. Therefore wires are natural but, of course, not always isomorphisms. Their role is to provide 'interconnection buses' between different components.

note that definition 1 does not restrict replacement situations to be witnessed by functions whose lifting to $\mathsf{Cp_B}$ results in *wires* — arbitrary functions, with the right types, can also be used. In general, law (7) justifies the following fact:

$$\ulcorner f \urcorner ; p ; \ulcorner g \urcorner \lessdot p \tag{12}$$

However, relation $\lessdot$ fails to be a pre-congruence with respect to the component operators introduced in [7]. It is easy to check that $\boxplus$, $\boxtimes$ and $\circledast$, as well as *wrapping* are preserved, *i.e.*, if $p \lessdot p'$ then, for any q, f and g, $p[f, g] \lessdot p'[f, g]$, $p \boxplus q \lessdot p' \boxplus q$ and, similarly, for the other two tensors. But things are different with respect to sequential composition and feedback. In these cases, the replaced expression may even become wrongly typed.

What $p \lessdot p'$ means is that component p can be replaced in *any* context by $p'[w_1, w_2]$, for any functions w_1, w_2 witnessing the fact. The explicit reference to them is actually required. Such is common in other approaches to interface refinement, such as [15], where witnesses are often considered part of the observation domain.

4 Forward and Backward Morphisms

Interface refinement is essentially concerned with plugging adjustment. Behaviour refinement, on the other hand, affects the internal dynamics of a component while leaving unchanged its external interface: it takes place inside each hom-category of Cp. Intuitively component p is a *behavioural refinement* of q if the behaviour patterns observed from p are a *structural restriction*, with respect to the *behavioural model* captured by monad B, of those of q. To make precise such a 'definition' we shall first describe behaviour patterns concretely as *generalized transitions.*

Actually, just as transition systems can be coded back as coalgebras, any coalgebra $\langle U, \alpha : \mathsf{T}U \longleftarrow U \rangle$ specifies a (T-shaped) transition structure over its carrier U. For extended polynomial Set endofunctors[3] such a structure may be expressed as a binary relation $_{\alpha}\!\!\longleftarrow : U \longleftarrow U$, defined in terms of the *structural membership* relation $\in_{\mathsf{T}} : U \longleftarrow \mathsf{T}\,U$, *i.e.*,

$$u' \;_{\alpha}\!\!\longleftarrow u \;\equiv\; u' \in_{\mathsf{T}} \alpha\, u$$

[3] The class inductively defined as the least collection of functors containing the identity Id and constant functors K for all objects K in the category, closed by functor composition and finite application of product, coproduct, covariant exponential and finite powerset functors.

or, in an equivalent but pointfree formulation which othen simplifies formal reasoning, as the following relational equality [4]

$$_{\alpha}\longleftarrow \;=\; \in_{\mathsf{T}} \cdot \alpha$$

where $\in_{\mathsf{T}}$ is defined by induction on the structure of T [5]:

$$\begin{array}{rll}
x \in_{\mathsf{Id}} y & \text{iff} & x = y \\
x \in_{K} y & \text{iff} & \mathsf{false} \\
x \in_{\mathsf{T}_1 \times \mathsf{T}_2} y & \text{iff} & x \in_{\mathsf{T}_1} \pi_1\, y \;\vee\; x \in_{\mathsf{T}_2} \pi_2\, y \\
x \in_{\mathsf{T}_1 + \mathsf{T}_2} y & \text{iff} & \begin{cases} y = \iota_1\, y' & \Rightarrow x \in_{\mathsf{T}_1} y' \\ y = \iota_2\, y' & \Rightarrow x \in_{\mathsf{T}_2} y' \end{cases} \\
x \in_{\mathsf{T}^K} y & \text{iff} & \exists_{k \in K}.\; x \in_{\mathsf{T}} y\, k \\
x \in_{\mathcal{P}\mathsf{T}} y & \text{iff} & \exists_{y' \in y}.\; x \in_{\mathsf{T}} y'
\end{array}$$

For any function h, relation $\in_{\mathsf{T}}$ satisfies the following naturality condition

$$h \cdot \in_{\mathsf{T}} \;=\; \in_{\mathsf{T}} \cdot \mathsf{T}\, h \tag{13}$$

which can be proved by induction on T. Applying *shunting* [6] to the left to rigth inclusion component of equation (13) leads to

$$\in_{\mathsf{T}} \;\subseteq\; h^{\circ} \cdot \in_{\mathsf{T}} \cdot \mathsf{T}h \tag{14}$$

The dynamics of a component $p : O \longleftarrow I$ is based on functor $\mathsf{B}(\mathsf{Id} \times O)^I$. Therefore a possible (and intuitive) way of regarding component p as a behavioural refinement of q is to consider that p transitions are simply *preserved* in q. For non deterministic components this is understood simply as set inclusion. But one may also want to consider additional restrictions. For example, to stipulate that if p has no transitions from a given state, q should also have no transitions from the corresponding state(s). Or one may adopt the dual point of view requiring transition *reflection* instead of preservation. In any case the basic question remains: how can such a refinement situation be identified?

[4] In the sequel both functional and relational composition will be denoted by the same symbol $\cdot$ given that the former is just a special case of the latter.

[5] Relation $\in_{\mathsf{T}}$ is actually an instance of datatype membership defined in [16] by a Galois connection.

[6] In the relational calculus [17] Galois connection $f \cdot R \subseteq S \;\equiv\; R \subseteq f^{\circ} \cdot S$, involving function f and arbitrary relations R and S, is known as the *shunting rule*. Also note that notation R° stands for the *converse* of relation R.

In data refinement, as mentioned above, there is a ‘recipe’ to identify a refinement situation: look for an *abstraction function* to witness it. In other words: look for a morphism in the relevant category, from the ‘concrete’ to the ‘abstract’ model such that the latter can be *recovered* from the former up to a suitable notion of equivalence, though, typically, not in a unique way.

In our components’ framework, however, things do not work this way. The reason is obvious: component morphisms are (seed preserving) coalgebra morphisms which are known (*e.g.*, [6]) to entail bisimilarity. Therefore we have to look for a somewhat *weaker* notion of a morphism between coalgebras.

Recall first that a T-coalgebra morphism $h : \beta \longleftarrow \alpha$ is a function from the state space of α to that of β such that

$$\mathsf{T}h \cdot \alpha \; = \; \beta \cdot h \tag{15}$$

Regarding α and β as (generalised) transition systems equation (15) becomes a relational equality (by a straightforward generalisation of an argument in [6]):

$$h \cdot \; {}_{\alpha}\!\longleftarrow \quad = \quad {}_{\beta}\!\longleftarrow \cdot h \tag{16}$$

i.e., the conjunction of inclusions

$$h \cdot \; {}_{\alpha}\!\longleftarrow \quad \subseteq \quad {}_{\beta}\!\longleftarrow \cdot h \tag{17}$$

$${}_{\beta}\!\longleftarrow \cdot h \quad \subseteq \quad h \cdot \; {}_{\alpha}\!\longleftarrow \tag{18}$$

By *shunting* inclusion (17) can also be presented in the following equivalent way:

$${}_{\alpha}\!\longleftarrow \quad \subseteq \quad h^{\circ} \cdot \; {}_{\beta}\!\longleftarrow \cdot h \tag{19}$$

Note that introducing variables inequalities (19) and (18) take the following more familiar shape:

$$u' \; {}_{\alpha}\!\longleftarrow u \;\; \Rightarrow \;\; h\,u' \; {}_{\beta}\!\longleftarrow h\,u \tag{20}$$

$$v' \; {}_{\beta}\!\longleftarrow h\,u \;\; \Rightarrow \;\; \exists_{u' \in U} \cdot \; u' \; {}_{\alpha}\!\longleftarrow u \; \wedge \; u' = h\,v' \tag{21}$$

They jointly state that, not only α dynamics, as represented by the induced transition relation, is *preserved* by h (17), but also β dynamics is *reflected* back over the same h (18). Is it possible to weaken the morphism definition to capture only one of these aspects? The answer is yes and resorts to the notion of a preorder $\leq$ on a Set endofunctor T. This was defined in [18] as a functor $\leq$ which makes the following diagram commute:

$$\begin{tikzcd}
 & \mathsf{PreOrd} \arrow[d] \\
\mathsf{Set} \arrow[ur, "\leq"] \arrow[r, "\mathsf{T}"'] & \mathsf{Set}
\end{tikzcd}
\qquad \text{concretely} \qquad
\begin{tikzcd}
 & (\mathsf{T}U, \leq_{\mathsf{T}U}) \arrow[d] \\
U \arrow[ur] \arrow[r] & \mathsf{T}U
\end{tikzcd}$$

This means that for any function $h : V \longleftarrow U$, $\mathsf{T}h$ preserves the order, *i.e.*

$$x_1 \leq_{\mathsf{T}X} x_2 \;\Rightarrow\; (\mathsf{T}h)\, x_1 \leq_{\mathsf{T}Y} (\mathsf{T}h)\, x_2 \tag{22}$$

or, in a pointfree formulation,

$$(\mathsf{T}h) \cdot \leq \;\subseteq\; \leq \cdot (\mathsf{T}\, h) \tag{23}$$

Let us denote by $\dot{\leq}$ the pointwise lifting of $\leq$ to the functional level, *i.e.*

$$f \,\dot{\leq}\, g \;\equiv\; \forall_x .\; f\, x \leq g\, x \tag{24}$$

which can also be formulated in the following pointfree way, more suitable for calculation,

$$f \,\dot{\leq}\, g \;\equiv\; f \subseteq \leq \cdot g \tag{25}$$

Clearly for any function h monotonic with respect to $\leq$ one has

$$f \,\dot{\leq}\, g \;\Rightarrow\; h \cdot f \,\dot{\leq}\, h \cdot g \tag{26}$$

because

$$\begin{array}{ll}
 & f \,\dot{\leq}\, g \\
\equiv & \quad \{ \ \dot{\leq} \text{ definition } \} \\
 & f \subseteq \leq \cdot g \\
\Rightarrow & \quad \{ \ \text{monotonicity of } \cdot \text{ wrt } \subseteq \ \} \\
 & h \cdot f \subseteq h \cdot \leq \cdot g \\
\Rightarrow & \quad \{ \ h \text{ is monotonic wrt } \leq \} \\
 & h \cdot f \subseteq \leq \cdot h \cdot g \\
\equiv & \quad \{ \ \dot{\leq} \text{ definition } \} \\
 & h \cdot f \,\dot{\leq}\, h \cdot g
\end{array}$$

Similarly,

$$f \,\dot{\leq}\, g \;\Rightarrow\; f \cdot h \,\dot{\leq}\, g \cdot h \tag{27}$$

Definition 3 *Let* T *be an extended polynomial functor on* Set *and consider two* T*-coalgebras* $\alpha : \mathsf{T}U \longleftarrow U$ *and* $\beta : \mathsf{T}V \longleftarrow V$*. A* forward *morphism* $h : \beta \longleftarrow \alpha$ *with respect to a preorder* $\leq$*, is a function from* U *to* V *such that*

$$\mathsf{T}\, h \cdot \alpha \;\dot{\leq}\; \beta \cdot h$$

Dually, h *is called a* backwards *morphism if*

$$\beta \cdot h \;\dot{\leq}\; \mathsf{T}\, h \cdot \alpha$$

The following lemma shows that such morphisms compose and can be taken as witnesses of refinement situations:

Lemma 4 *For* T *an endofunctor in* Set*,* T*-coalgebras and forward (respectively, backward) morphisms define a category.*

PROOF. In both cases, identities are the identities on the carrier and composition is inherited from Set. What remains to be shown is that the composition of forward (respectively, backward) morphisms yields again a forward (respectively, backward) morphism. So, let $h : \beta \longleftarrow \alpha$ and $k : \gamma \longleftarrow \beta$ be two forward (respectively, backward) morphisms. Then

(*forward* case)

$$
\begin{array}{ll}
& \mathsf{T}(k \cdot h) \cdot \alpha \\
= & \{ \ \mathsf{T} \text{ functor}, \cdot \text{ associative} \ \} \\
& \mathsf{T}k \cdot (\mathsf{T}h \cdot \alpha) \\
\dot{\leq} & \{ \ h \text{ forward and } (26) \ \} \\
& \mathsf{T}k \cdot (\beta \cdot h) \\
= & \{ \ \cdot \text{ associative} \ \} \\
& (\mathsf{T}k \cdot \beta) \cdot h \\
\dot{\leq} & \{ \ k \text{ forward and } (27) \ \} \\
& (\gamma \cdot k) \cdot h \\
= & \{ \ \cdot \text{ associative} \ \} \\
& \gamma \cdot (k \cdot h)
\end{array}
$$

(*backward* case)

$$
\begin{array}{ll}
& \gamma \cdot (k \cdot h) \\
= & \{ \ \cdot \text{ associative} \ \} \\
& (\gamma \cdot k) \cdot h \\
\dot{\leq} & \{ \ k \text{ backward and } (27) \ \} \\
& (\mathsf{T}k \cdot \beta) \cdot h \\
= & \{ \ \cdot \text{ associative} \ \} \\
& \mathsf{T}k \cdot (\beta \cdot h) \\
\dot{\leq} & \{ \ h \text{ backward and } (26) \ \} \\
& \mathsf{T}k \cdot \mathsf{T}h \cdot \alpha \\
= & \{ \ \mathsf{T} \text{ functor} \ \} \\
& \mathsf{T}(k \cdot h) \cdot \alpha
\end{array}
$$

□

Such a split of a coalgebra morphism into two conditions, makes it possible to capture separately transition *preservation* and *reflection*. Lemma 5 below states that forward morphisms preserve transitions whereas backwards morphisms reflect them. To prove this however the following extra condition has to be imposed on preorder $\leq$ to express its compatibility with the membership relation: for all $x \in X$ and $x_1, x_2 \in \mathsf{T}X$,

$$x \in_{\mathsf{T}} x_1 \ \wedge \ x_1 \leq \ x_2 \ \Rightarrow \ x \in_{\mathsf{T}} x_2 \tag{28}$$

or, again in a pointfree formulation,

$$\in_{\mathsf{T}} \cdot \leq \ \subseteq \ \in_{\mathsf{T}} \tag{29}$$

A preorder $\leq$ on an endofunctor T satisfying inclusion (29) will be referred to, in the sequel, as a *refinement preorder*. Then,

Lemma 5 *Let* T *be an extended polynomial functor in* Set*, and* α *and* β *two* T*-coalgebras as above. Let* ${}_{\alpha}\!\longleftarrow$ *and* ${}_{\beta}\!\longleftarrow$ *denote the corresponding transition relations. A forward (respectively, backward) morphism* $h : \beta \longleftarrow \alpha$ *preserves (respectively, reflects) such transition relations.*

PROOF. Let h be a forward morphism. Transition preservation, defined by equation (19), follows from

$$
\begin{array}{ll}
 & {}_{\alpha}\!\longleftarrow \\
= & \quad \{ \text{ definition } \} \\
 & \in_{\mathsf{T}} \cdot \alpha \\
\subseteq & \quad \{ \text{ (14), monotonicity } \} \\
 & h^{\circ} \cdot \in_{\mathsf{T}} \cdot \mathsf{T}\, h \cdot \alpha \\
\subseteq & \quad \{ \ h \text{ forward entails } \mathsf{T}h \cdot \alpha \subseteq \leq \cdot \beta \cdot h \text{, monotonicity } \} \\
 & h^{\circ} \cdot \in_{\mathsf{T}} \cdot \leq \cdot \beta \cdot h \\
\subseteq & \quad \{ \text{ compatibility with } \in_{\mathsf{T}} \text{ (29), monotonicity } \} \\
 & h^{\circ} \cdot \in_{\mathsf{T}} \cdot \beta \cdot h \\
= & \quad \{ \text{ definition } \} \\
 & h^{\circ} \cdot {}_{\beta}\!\longleftarrow \cdot h
\end{array}
$$

Let now h be a backward morphism. Transition reflection, defined by equation (18), is established as follows:

$$
\begin{array}{ll}
 & {}_{\beta}\!\longleftarrow \cdot h \\
= & \quad \{ \text{ definition } \} \\
 & \in_{\mathsf{T}} \cdot \beta \cdot h \\
\subseteq & \quad \{ \ h \text{ backwards entails } \beta \cdot h \subseteq \leq \cdot \mathsf{T}h \cdot \alpha \text{, monotonicity } \} \\
 & \in_{\mathsf{T}} \cdot \leq \cdot \mathsf{T}h \cdot \alpha \\
\subseteq & \quad \{ \text{ compatibility with } \in_{\mathsf{T}} \text{ (29), monotonicity } \} \\
 & \in_{\mathsf{T}} \cdot \mathsf{T}h \cdot \alpha \\
\equiv & \quad \{ \in_{\mathsf{T}} \text{ natural (13) } \} \\
 & h \cdot \in_{\mathsf{T}} \cdot \alpha \\
= & \quad \{ \text{ definition } \} \\
 & h \cdot {}_{\alpha}\!\longleftarrow
\end{array}
$$

$\square$

5 Behaviour Refinement

The existence of a *forward* (*backward*) morphism connecting two components p and q witnesses a refinement situation whose symmetric closure coincides, as expected, with bisimulation. In the sequel we will restrict ourselves to *forward* refinement[7] and define *behaviour refinement* by the existence of a forward morphism *up to bisimulation.* Formally,

Definition 6 *Component p is a behaviour refinement of q, written $q \trianglelefteq p$, if there exist components r and s and a (seed preserving) forward morphism h such that*

$$\begin{tikzcd} q \arrow[r, phantom, "\sim"] & s & r \arrow[l, "h"'] \arrow[r, phantom, "\sim"] & p \end{tikzcd}$$

The exact meaning of a refinement assertion $q \trianglelefteq p$ depends, of course, on the concrete refinement preorder $\leq$ adopted. But what do we know about such preorders? Condition (29) provides an upper bound (the lower bound being the quite unhelpful empty relation). Actually (29) equivales

$$\leq \;\subseteq\; \in_{\mathsf{T}} \setminus \in_{\mathsf{T}} \tag{30}$$

by direct application of the Galois connection which defines relational division[8]

$$R \cdot X \;\subseteq\; S \quad\equiv\quad X \;\subseteq\; R \setminus S \tag{31}$$

It is well known (see *e.g.*, [16] or [19]) that relation $\in_{\mathsf{T}} \setminus \in_{\mathsf{T}}$ corresponds to the lifting of $\in_{\mathsf{T}}$ to a (structural) inclusion, *i.e.*,

$$x\,(\in_{\mathsf{T}} \setminus \in_{\mathsf{T}})\,y \quad\equiv\quad \forall_{e \in_{\mathsf{T}} x}\,.\; e \in_{\mathsf{T}} y \tag{32}$$

By (30) this is the largest refinement preorder. All the interesting ones arise by suitable restrictions. Let us consider a few possibilities.

- Structural inclusion as defined above is too large to be useful in practice. Actually its definition on a constant functor is the universal relation which, in our component model, would make refinement based on $\in_{\mathsf{T}} \setminus \in_{\mathsf{T}}$ blind to the outputs produced. This suggests an additional requirement on refinement preorders for Cp components: their definition on a constant functor K must be equality on set K, *i.e.*, $x \leq_K y \;\;\mathsf{iff}\;\; x =_K y$ so that transitions with

[7] A similar study can be made about *backward* refinement, although the underlying intuition seems less familiar.

[8] A pointwise definition of this operator reads

$$x\,(R \setminus S)\,z \quad\equiv\quad \forall_y\,.\; yRx \;\Rightarrow\; ySz$$

different O-labels could not be related. Building on this idea, we arrive at a first (good) example:

$$
\begin{array}{rcl}
x \subseteq_{\mathsf{Id}} y & \text{iff} & x = y \\
x \subseteq_{K} y & \text{iff} & x =_K y \\
x \subseteq_{\mathsf{T}_1 \times \mathsf{T}_2} y & \text{iff} & \pi_1\, x \subseteq_{\mathsf{T}_1} \pi_1\, y \;\wedge\; \pi_2\, x \subseteq_{\mathsf{T}_2} \pi_2\, y \\
x \subseteq_{\mathsf{T}_1 + \mathsf{T}_2} y & \text{iff} & \begin{cases} x = \iota_1\, x' \wedge y = \iota_1\, y' & \Rightarrow x' \subseteq_{\mathsf{T}_1} y' \\ x = \iota_2\, x' \wedge y = \iota_2\, y' & \Rightarrow x' \subseteq_{\mathsf{T}_2} y' \end{cases} \\
x \subseteq_{\mathsf{T}^K} y & \text{iff} & \forall_{k \in K}.\; x\, k \subseteq_{\mathsf{T}} y\, k \\
x \subseteq_{\mathcal{P}\mathsf{T}} y & \text{iff} & \forall_{e \in x} \exists_{e' \in y}.\; e \subseteq_{\mathsf{T}} e'
\end{array}
$$

This preorder will be referred to in the sequel as *structural inclusion*. Note that *forward* refinement of non deterministic components based on $\subseteq_{\mathsf{T}}$ captures the classical notion of *non determinism reduction*.

- However, this preorder can be tuned to more specific cases. For example, the following 'failure forcing' variant — $\subseteq^E_{\mathsf{T}}$, where E stands for 'emptyset' — guarantees that the concrete component fails no more than the abstract one. It is defined as $\subseteq_{\mathsf{T}}$ by replacing the clause for the powerset functor by

$$x \subseteq^E_{\mathcal{P}\mathsf{T}} y \quad \text{iff} \quad (x = \emptyset \Rightarrow y = \emptyset) \;\wedge\; \forall_{e \in x} \exists_{e' \in y}.\; e \subseteq_{\mathsf{T}} e'$$

- Relation $\subseteq_{\mathsf{T}}$ is inadequate for partial components: refinement would collapse into bisimilarity instead of entailing an increase of definition in the implementation side. An alternative is relation $\subseteq^F_{\mathsf{T}}$ (F standing for 'failure') which adds a *maybe* clause

$$x \subseteq^F_{\mathsf{T}+\mathbf{1}} y \quad \text{iff} \quad \begin{cases} x = \iota_1\, x' \wedge y = \iota_1\, y' & \Rightarrow x' \subseteq_{\mathsf{T}} y' \\ x = \iota_2\, * & \Rightarrow \mathsf{true} \end{cases}$$

taking precendence over the general sum clause.

For illustration purposes consider again component LBuff introduced in section 2, and a *deterministic* buffered channel Buff specified as a coalgebra $(M^* \times (\mathbf{1} + M))^{M+\mathbf{1}} \longleftarrow M^*$, with nil as the initial state and dynamics given by

$$
\begin{array}{rcl}
a_{\mathsf{Buff}}\langle u, \mathsf{put}\; m\rangle & = & \langle m : u, \iota_1\, *\rangle \\
a_{\mathsf{Buff}}\langle u, \mathsf{pick}\rangle & = & \langle \mathsf{tl}\; u, \iota_2\; (\mathsf{hd}\; u)\rangle
\end{array}
$$

To establish $\mathsf{LBuff} \trianglelefteq \mathsf{Buff}$ entails the need to embed the latter into the space of non deterministic systems. This is achieved by a (natural) transformation from $(\mathsf{Id} \times O)^I$ to $\mathcal{P}(\mathsf{Id} \times O)^I$ canonically extending function $\mathsf{sing}\, x = \{x\}$ which is a monad morphism from the *identity* to the *powerset* monads — the behaviour models underlying Buff and LBuff, respectively. Then, it is immediate to verify that the identity function on state space M^* is a *forward* morphism, with respect to structural inclusion, *i.e.*,

$$(\mathsf{id} \times O) \cdot \overline{a}_{\mathsf{Buff}} \subseteq_{(\mathsf{B}\ (\mathsf{Id} \times O))^I} \overline{a}_{\mathsf{LBuff}}$$

6 Refinement Laws in the Component Calculus

As mentioned in the introduction, the main motivation underlying this research has been the development of *inequational* laws in the context of the component calculus proposed in [7]. Although there is not enough space in this paper to introduce the calculus in full detail, this section concentrates on two case studies where refinement results do emerge and prove useful: canonical arrow lifting and parallel composition.

6.1 Lifting

Lifting canonical Set arrows to Cp is a simple way to explore the structure of Cp itself. For instance, consider the lifting of $?_I : I \longleftarrow \emptyset$. Clearly, $?_I$ keeps its naturality as, for any $p : O \longleftarrow I$, the following diagram commutes up to bisimulation,

$$\begin{tikzcd} I \arrow[r, "p"] & O \\ \emptyset \arrow[u, "\ulcorner ?_I \urcorner"] \arrow[ur, "\ulcorner ?_O \urcorner"'] & \end{tikzcd}$$

because both $\ulcorner ?_I \urcorner$ and $\ulcorner ?_O \urcorner$ are *inert* components: the absence of input makes reaction impossible. Formally equation $\ulcorner ?_I \urcorner ; p \sim \ulcorner ?_O \urcorner$ lifts to an equality in Bh, which makes $\emptyset$ the *initial* object there.

A different situation emerges when considering the lifting of $!_I : \mathbf{1} \longleftarrow I$ because in general, *i.e.*, for an arbitrary B, the following diagram fails to commute.

$$\begin{tikzcd} I \arrow[r, "p"] \arrow[d, "\ulcorner !_I \urcorner"'] & O \arrow[dl, "\ulcorner !_O \urcorner"] \\ \mathbf{1} & \end{tikzcd}$$

The reason is that the Cp lifting of the final arrow (as the lifting of any other function) cannot fail, whereas the $\ulcorner !_I \urcorner ; p$ may fail (whenever p does). Function $! : \mathbf{1} \longleftarrow U_p \times \mathbf{1}$ is, however, a forward morphism, with respect to $\subseteq^F_{\mathsf{T}}$ for partial components, or to both $\subseteq_{\mathsf{T}}$ and $\subseteq^E_{\mathsf{T}}$ for non deterministic ones. For this last case, note that $\overline{a}_{\ulcorner !_O \urcorner} \cdot ! = \lambda\, i \in I.\ \{*\}$, whereas $\mathsf{B}(! \times \mathsf{id})^I \cdot \overline{a}_{p;\ulcorner !_O \urcorner}\ \langle u, * \rangle$ equals

$$\lambda\, i \in I\ .\ \begin{cases} \{*\} & \mathsf{iff}\ (\overline{a}_p\, u)\,(i) \neq \emptyset \\ \emptyset & \mathsf{iff}\ (\overline{a}_p\, u)\,(i) = \emptyset \end{cases}$$

Therefore, $\ulcorner !_I \urcorner \trianglelefteq p\,;\ulcorner !_O \urcorner$. Also notice that $\mathbf{1}$ does not become the final object in Bh, but in the case of deterministic components (*i.e.*, for $\mathsf{B} = \mathsf{Id}$).

6.2 Parallel Composition

The next case study concerns the development of the theory of *parallel* composition. This combinator, denoted by $p \boxtimes q$, corresponds to a synchronous product: both components are executed simultaneously when triggered by a pair of legal input values. Note, however, that the behaviour effect, captured by monad B, propagates. For example, if B can express component failure and one of the arguments fails, the product will fail as well. Formally, $\boxtimes$ is defined on objects as $I \boxtimes J \;=\; I \times J$ and a family of functors

$$\boxtimes_{IOJR} : \mathsf{Cp}(I \times J, O \times R) \longleftarrow \mathsf{Cp}(I, O) \times \mathsf{Cp}(J, R)$$

which yields

$$p \boxtimes q \;=\; \langle \langle u_p, u_q \rangle \in U_p \times U_q, \overline{a}_{p \boxtimes q} \rangle$$

where

$$\begin{aligned} a_{p\boxtimes q} = \quad & U_p \times U_q \times (I \times J) \xrightarrow{\ \mathsf{m}\ } U_p \times I \times (U_q \times J) \\ & \xrightarrow{\ a_p \times a_q\ } \mathsf{B}\ (U_p \times O) \times \mathsf{B}\ (U_q \times R) \\ & \xrightarrow{\ \delta_l\ } \mathsf{B}\ (U_p \times O \times (U_q \times R)) \\ & \xrightarrow{\ \mathsf{B\ m}\ } \mathsf{B}\ (U_p \times U_q \times (O \times R)) \end{aligned}$$

and maps every pair of arrows $\langle h_1, h_2 \rangle$ into $h_1 \times h_2$. Notice that $\mathsf{m} : (A \times C) \times (B \times D) \longleftarrow (A \times B) \times (C \times D)$ is simply a re-arrangement natural isomorphism.

In [8] several equational results were proved for $\boxtimes$, capturing, in particular, its monoidal structure and lax functoriality, the latter entailing distribution with respect to sequential composition. Thus, for example,

$$\begin{array}{rll} \textit{lax} & (p \boxtimes p')\,;(q \boxtimes q') \;\sim\; (p\,;q) \boxtimes (p'\,;q') & (33) \\ & \mathsf{copy}_{K \boxtimes K'} \;\sim\; \mathsf{copy}_K \boxtimes \mathsf{copy}_{K'} & (34) \\ \textit{functions} & \ulcorner f \urcorner \boxtimes \ulcorner g \urcorner \;\sim\; \ulcorner f \times g \urcorner & (35) \\ \textit{assoc} & (p \boxtimes q) \boxtimes r \;\doteqdot\; p \boxtimes (q \boxtimes r) & (36) \\ \textit{id} & \mathsf{idle} \boxtimes p \;\doteqdot\; p & (37) \end{array}$$

$$\begin{array}{llr} \textit{zero} & \mathsf{nil} \boxtimes p \doteqdot \mathsf{nil} & (38) \\ \textit{comm} & p \boxtimes q \doteqdot q \boxtimes p \quad \text{if } \mathsf{B} \text{ is } \textit{commutative} & (39) \end{array}$$

Notice that the last four laws hold only up to suitable re-wiring and are, therefore, presented in terms of the interface interchangeability relation $\doteqdot$ introduced in section 3. For example, $(p \boxtimes q) \boxtimes r \sim (p \boxtimes (q \boxtimes r))[\mathsf{a}, \mathsf{a}^\circ]$, where $\mathsf{a} : A \times (B \times C) \longleftarrow (A \times B) \times C$ is the associativity isomorphism and a° its converse.

To introduce refinement let us concentrate on the question of whether $\boxtimes$ can act as a universal product construction in the calculus. The answer, as discussed below, is, in general, negative, but using behavioural refinement several useful results can still be proved. Just as in Set one defines the universal arrow in a cartesian product diagram as the *split* $\langle f, g \rangle$ of two functions[9], let us start by defining the split of two components as

$$\langle p, q \rangle \;=\; \ulcorner \triangle \urcorner \,;\, (p \boxtimes q) \quad \text{where} \quad \triangle = \langle \mathsf{id}, \mathsf{id} \rangle$$

We shall investigate both the suitability of this definition, expressed by the commutativity of diagram

$$\begin{tikzcd} & I \arrow[dl, "p"'] \arrow[d, "\langle p,q \rangle"] \arrow[dr, "q"] & \\ O & O \boxtimes R \arrow[l, "\ulcorner \pi_1 \urcorner"] \arrow[r, "\ulcorner \pi_2 \urcorner"'] & R \end{tikzcd} \tag{40}$$

as well as its uniqueness. The crucial point to look at is whether the diagonal arrow $\triangle$ keeps its naturality when lifted to Cp, *i.e.*,

$$\ulcorner \triangle \urcorner \,;\, (p \boxtimes p) \;\sim\; p \,;\, \ulcorner \triangle \urcorner \tag{41}$$

or, equivalently, by (7),

$$(p \boxtimes p)[\triangle, \mathsf{id}] \;\sim\; p[\mathsf{id}, \triangle] \tag{42}$$

The obvious candidate to establish bisimilarity is $\triangle$: $U_p \times U_p \longleftarrow U_p$, which is clearly seed preserving. To show that $\triangle$ is a component morphism amounts to verifying the commutativity of

$$\begin{tikzcd} (U_p \times U_p) \times I \arrow[r, "a_{(p \boxtimes p)[\triangle, \mathsf{id}]}"] & \mathsf{B}(U_p \times U_p \times (O \times O)) \\ U_p \times I \arrow[u, "\triangle \times \mathsf{id}"] \arrow[r, "a_{p[\mathsf{id}, \triangle]}"] & \mathsf{B}(U_p \times U_q \times (O \times O)) \arrow[u, "\mathsf{B} \triangle \times \mathsf{id}"'] \end{tikzcd}$$

Then,

[9] A concrete, pointwise definition reads $\langle f, g \rangle\ x = \langle f\ x, g\ x \rangle$.

$$
\begin{array}{ll}
& \mathsf{B}(\triangle \times \mathsf{id}) \cdot a_{p[\mathsf{id},\triangle]} \\
= & \quad \{ \text{ wrapping definition } \} \\
& \mathsf{B}(\triangle \times \mathsf{id}) \cdot \mathsf{B}(\mathsf{id} \times \triangle) \cdot a_p \\
= & \quad \{ \text{ routine: } \mathsf{m} \cdot \triangle = \triangle \times \triangle \} \\
& \mathsf{Bm} \cdot \mathsf{B} \triangle \cdot a_p \\
\overset{?}{=} & \quad \{ \star \} \\
& \mathsf{Bm} \cdot \delta_l \cdot \triangle \cdot a_p \\
= & \quad \{ \triangle \text{ natural } \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_p) \cdot \triangle \\
= & \quad \{ \text{ routine: } \mathsf{m} \cdot (\triangle \times \triangle) = \triangle \} \\
& \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_p) \cdot \mathsf{m} \cdot (\triangle \times \triangle) \\
= & \quad \{ \boxtimes \text{ definition } \} \\
& a_{p \boxtimes p} \cdot (\triangle \times \triangle) \\
= & \quad \{ \text{ wrapping definition } \} \\
& a_{(p \boxtimes p)[\triangle,\mathsf{id}]} \cdot (\triangle \times \mathsf{id})
\end{array}
$$

Clearly this is valid equational reasoning when $\mathsf{B} = \mathsf{Id}$ or $\mathsf{B} = \mathsf{Id} + \mathbf{1}$. In general, however, the step marked with a $\star$ is problematic. For example, for $\mathsf{B} = \mathcal{P}$, one gets for any set s

$$\mathcal{P}\triangle\ s \;=\; \{\langle x, x\rangle \mid x \in s\} \;\subseteq\; \{\langle x, y\rangle \mid x, y \in s\} \;=\; \delta_l\ \langle s, s\rangle \;=\; (\delta_l \cdot \triangle)\ s$$

Therefore, symbol $\overset{?}{=}$ in the calculation above becomes $\subseteq$. Keeping in mind that structural inclusion $\subseteq_{\mathsf{T}}$ for functor $\mathsf{T} = \mathcal{P}(\mathsf{Id} \times O)^I$ amounts to

$$x \subseteq_{\mathcal{P}(\mathsf{Id} \times O)^I} y \;\equiv\; \forall_{i \in I}\ .\ \forall_{e \in x\ i}\ .\ \exists_{e' \in y\ i}\ .\ e = e' \;\equiv\; \forall_{i \in I}\ .\ x\ i \subseteq y\ i$$

by choosing $\subseteq_{\mathsf{T}}$ as the refinement preorder and taking $\mathsf{B} = \mathcal{P}$, we arrive at

$$\ulcorner \triangle \urcorner ; (p \boxtimes p) \;\trianglelefteq\; p ; \ulcorner \triangle \urcorner \tag{43}$$

As a consequence, a *fusion* law for component spliting emerges just as another refinement law:

$$r ; \langle p, q \rangle \;\trianglelefteq\; \langle r ; p, r ; q \rangle \tag{44}$$

which can be verified by the following calculation

$$
\begin{array}{ll}
& r ; \langle p, q \rangle \\
\sim & \quad \{ \text{ definition } \} \\
& r ; (\ulcorner \triangle \urcorner ; (p \boxtimes q))
\end{array}
$$

$\sim \quad \{ \text{ ; associative } \}$

$(r \,;\ulcorner \triangle \urcorner) \,; (p \boxtimes q)$

$\trianglelefteq \quad \{ \text{ law (43), ; associative } \}$

$\ulcorner \triangle \urcorner ; ((r \boxtimes r) \,; (p \boxtimes q))$

$\sim \quad \{ \text{ law (34) } \}$

$\ulcorner \triangle \urcorner ; ((r \,; p) \boxtimes (r \,; q))$

$\sim \quad \{ \text{ definition } \}$

$\langle r \,; p, r \,; q \rangle$

Let us go back to diagram (40) to discuss its commutativity. Note that each triangle represents a *cancellation* law which we may try to verify by checking whether $\pi_1 : U_p \longleftarrow U_p \times U_q$ is a component morphism. A tentative calculation goes as follows:

$\mathsf{B}(\pi_1 \times \mathsf{id}) \cdot a_{\langle p,q \rangle ;\ulcorner \pi_1 \urcorner}$

$= \quad \{ \text{ definitions } \}$

$\mathsf{B}(\pi_1 \times \pi_1) \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \triangle)$

$= \quad \{ \text{ routine: } \pi_1 \times \pi_1 = \pi_1 \cdot \mathsf{m} \}$

$\mathsf{B}\pi_1 \cdot \mathsf{Bm} \cdot \mathsf{Bm} \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \triangle)$

$= \quad \{ \mathsf{m}^\circ = \mathsf{m} \}$

$\mathsf{B}\pi_1 \cdot \delta_l \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \triangle)$

$\overset{?}{=} \quad \{ \star \}$

$\pi_1 \cdot (a_p \times a_q) \cdot \mathsf{m} \cdot (\mathsf{id} \times \triangle)$

$= \quad \{ \times \text{ cancellation } \}$

$a_p \cdot \pi_1 \cdot \mathsf{m} \cdot (\mathsf{id} \times \triangle)$

$= \quad \{ \text{ routine: } \pi_1 \times \pi_1 = \pi_1 \cdot \mathsf{m} \}$

$a_p \cdot (\pi_1 \times \pi_1) \cdot (\mathsf{id} \times \triangle)$

$= \quad \{ \text{ routine: } \pi_1 \cdot \triangle = \mathsf{id} \}$

$a_p \cdot (\pi_1 \times \mathsf{id})$

Consider the validity of the step marked with a $\star$. Clearly this amounts to equality for $\mathsf{B} = \mathsf{Id}$. But what happens for the *maybe* monad? Let $(\mathsf{Id} \times \mathsf{Id}) + \mathbf{1} : (\mathsf{Id} + \mathbf{1}) \times (\mathsf{Id} + \mathbf{1}) \Longleftarrow (\mathsf{Id} + \mathbf{1}) \times (\mathsf{Id} + \mathbf{1})$. An easy computation leads to $((\pi_1 + \mathbf{1}) \cdot \delta_l) \ \langle \iota_1 a, \iota_2 \star \rangle = \iota_2 \star$ whereas $\pi_1 \ \langle \iota_1 a, \iota_2 \star \rangle = \iota_1 a$. In this case symbol $\overset{?}{=}$ must be replaced by $\subseteq^F_{\mathsf{B}(\mathsf{Id} \times O)}$, leading again to a weak version of cancellation as a refinement law

$$p \ \trianglelefteq \ \langle p, q \rangle \,; \ulcorner \pi_1 \urcorner \tag{45}$$

The result also holds for $\mathsf{B} = \mathcal{P}$ and structural inclusion as the refinement preorder. In such a case, however, refinement boils down to bisimulation unless the left argument is the empty set: for example, $(\mathsf{B}\pi_1 \cdot \delta_l)\langle X, \emptyset\rangle = \emptyset \neq X = \pi_1\langle X, \emptyset\rangle$. Therefore diagram (40) strictly commutes for the non empty powerset monad, a monad which expresses non deterministic behaviour excluding the possibility of *failure*. This seems to be the general rule concerning the existence of *splits* for components based on commutative monads: the exclusion of failure. The result fails because the eventual failure of q propagates, leading to the failure of $\langle p, q\rangle$.

It should be stressed, however, that even in cases where cancellation fails (and consequently, construction $\langle p, q\rangle$ can hardly be called a *split*) the following *reflection* and *absorption* laws hold, the latter only if B is a commutative monad:

$$\langle \ulcorner\pi_1\urcorner, \ulcorner\pi_2\urcorner\rangle \;\sim\; \mathsf{copy}_{O\times R} \tag{46}$$
$$\langle p, q\rangle \,;\, (p' \boxtimes q') \;\sim\; \langle p \,;\, p', q \,;\, q'\rangle \tag{47}$$

Let us check (46) in the first place:

$$\begin{array}{ll}
 & \langle \ulcorner\pi_1\urcorner, \ulcorner\pi_2\urcorner\rangle \\
\sim & \quad \{ \text{ definition } \} \\
 & \ulcorner\triangle\urcorner \,;\, (\ulcorner\pi_1\urcorner \boxtimes \ulcorner\pi_2\urcorner) \\
\sim & \quad \{ \text{ law (35) } \} \\
 & \ulcorner\triangle\urcorner \,;\, \ulcorner\pi_1 \times \pi_2\urcorner \\
\sim & \quad \{ \text{ law (6) } \} \\
 & \ulcorner(\pi_1 \times \pi_2)\cdot \triangle\urcorner \\
\sim & \quad \{ \times \text{ absorption and identity in } \mathsf{Cp} \} \\
 & \mathsf{copy}_{O\times R}
\end{array}$$

Concerning (47):

$$\begin{array}{ll}
 & \langle p, q\rangle \,;\, (p' \boxtimes q') \\
\sim & \quad \{ \text{ definition } \} \\
 & \ulcorner\triangle\urcorner \,;\, (p \boxtimes q) \,;\, (p' \boxtimes q') \\
\sim & \quad \{ \text{ laws (5) and (34) } \} \\
 & \ulcorner\triangle\urcorner \,;\, ((p \,;\, p') \boxtimes (q \,;\, q')) \\
\sim & \quad \{ \text{ definition } \} \\
 & \langle p \,;\, p', q \,;\, q'\rangle
\end{array}$$

Notice that law (34) used above requires B to be commutative.

In summary, a *split* construction for components has been characterized even if resorting, in the general case, to refinement results. The following lemma answers our initial question:

Lemma 7 *Combinator $\boxtimes$ only lifts to a categorical product (up to bisimulation in* Cp *or strictly in the corresponding category of behaviours) for the identity monad,* i.e.*, for* total*,* deterministic *components.*

PROOF. Let us suppose that $\langle p, q\rangle$ is definable such that $\langle p, q\rangle ; \ulcorner \pi_1 \urcorner \sim p$ and $\langle p, q\rangle ; \ulcorner \pi_2 \urcorner \sim q$ and that there exists another component r satisfying the same equalities. Then, by $\sim$ transitivity, $r ; \ulcorner \pi_1 \urcorner \sim \langle p, q\rangle ; \ulcorner \pi_1 \urcorner$ and similarly for q and π_2. Thus

$$
\begin{array}{ll}
 & \langle p, q\rangle \\
\sim & \quad \{ \ \scriptstyle \boxtimes \text{ cancellation for } \mathsf{B} = \mathsf{Id} \ \} \\
 & \langle \langle p, q\rangle ; \ulcorner \pi_1 \urcorner, \langle p, q\rangle ; \ulcorner \pi_2 \urcorner \rangle \\
\sim & \quad \{ \ \scriptstyle \text{assumption} \ \} \\
 & \langle r ; \ulcorner \pi_1 \urcorner, r ; \ulcorner \pi_2 \urcorner \rangle \\
\sim & \quad \{ \ \scriptstyle \boxtimes \text{ fusion (44)} \ \} \\
 & r ; \langle \ulcorner \pi_1 \urcorner, \ulcorner \pi_2 \urcorner \rangle \\
\sim & \quad \{ \ \scriptstyle \boxtimes \text{ reflection (46)} \ \} \\
 & r ; \mathsf{copy}_{O \times R} \\
\sim & \quad \{ \ \scriptstyle \text{law (4)} \ \} \\
 & r
\end{array}
$$

$\square$

7 Simulations

This section introduces an alternative proof technique for establishing behaviour refinement: the identification of a *simulation* relation $R : U_q \longleftarrow U_p$ on the state spaces of the 'concrete' (p) and the 'abstract' (q) components. Again, the notion of a simulation depends on the adopted refinement preorder $\leq$. To proceed in a *generic* way, we adopt an equally generic definition of simulation due to Jacobs and Hughes in [18]:

Definition 8 *Given a* Set *endofunctor* T *and a refinement preorder* $\leq$*, a lax relation lifting is an operation* $Rel_{\leq}(\mathsf{T})$ *mapping relation* R *to* $\leq \cdot Rel(\mathsf{T})(R) \cdot \leq$*, where* $Rel(\mathsf{T})(R)$ *is the lifting of* R *to* T *(defined, as usual, as the* T*-image of inclusion* $\langle r_1, r_2\rangle : U \times V \longleftarrow R$*,* i.e.*,* $\langle \mathsf{T}r_1, \mathsf{T}r_2\rangle : \mathsf{T}U \times \mathsf{T}V \longleftarrow \mathsf{T}R$*).*

Given T*-coalgebras* α *and* β*, a simulation is a* $Rel_{\leq}(\mathsf{T})$*-coalgebra over* α *and* β*,* i.e.*, a relation* R *such that, for all* $u \in U, v \in V$*,* $\langle u, v\rangle \in R \Rightarrow \langle \alpha\, u, \beta\, v\rangle \in Rel_{\leq}(\mathsf{T})(R)$*.*

In order to prove that simulations are a sound proof technique to establish behaviour refinement we consider separately functional and non functional simulations. In any case, however, they are assumed to be entire relations[10].

Lemma 9 *Let* p *and* q *be* T*-components over state spaces* U *and* V*, respectively. For a given refinement preorder* $\leq$*, if there exists a simulation* $R : V \longleftarrow U$ *which is both functional and entire, then* p *is a forward refinement of* q*.*

PROOF. By assumption, simulation R is the graph of a function. Now, define a forward morphism $h : V \longleftarrow U$ as $h\, u = v \;\equiv\; \langle u, v\rangle \in R$. Because R is a simulation, for every pair $\langle u, v\rangle \in R$, there should exist $x \in \mathsf{T}U$, $y \in \mathsf{T}V$, such that $p\, u \leq_{\mathsf{T}U} x$, $y \leq_{\mathsf{T}V} q\, v$, and $\langle x, y\rangle \in Rel(\mathsf{T})(R)$, i.e., $y = (\mathsf{T}h)\, x$. Inclusion (23) and $p\, u \leq_{\mathsf{T}U} x$ entail $(\mathsf{T}h \cdot p)\, u \leq_{\mathsf{T}V} (\mathsf{T}h)\, x$, and thus $(\mathsf{T}h \cdot p)\, u \leq_{\mathsf{T}V} q\, v$. Since R is entire, h is defined for all $u \in U$, making the following diagram to commute:

$$\begin{tikzcd}
u \arrow[rr, "h"] \arrow[d, "p"'] & & h\ u \arrow[d, "q"] \\
p\ u\ (\leq_{\mathsf{T}U}\ p\ u) \arrow[r, "\mathsf{T}\, h"] & \mathsf{T}h\ (p\ u) \arrow[r, dotted, no head, "\leq"] & q\ h\ u
\end{tikzcd}$$

$\square$

Consider, now, the non-functional case (*e.g.* whenever two bisimilar but not equal abstract states are represented by a single concrete state). To prove soundness in this case, the state space of the 'concrete' component p is artificially inflated with an auxiliary value such that a forward morphism can be found. The technique is similar to the use of *ghost*-variables in [20].

Definition 10 *Given a coalgebra* $\langle U, \alpha : \mathsf{T}U \longleftarrow U\rangle$ *and a set* W*, define the extension of* α *to* W *as a coalgebra* $\widehat{\alpha}$ *over* $\widehat{U} = U \times W$ *such that* $\mathsf{T}\pi_1 \cdot \widehat{\alpha} = \alpha \cdot \pi_1$*.*

Clearly this auxiliary state space does not interfere with the behaviour of α: π_1 being a coalgebra morphism, the two coalgebras are bisimilar.

Given components p, $q : O \longleftarrow I$ and a non-functional simulation R an auxiliary 'component' $\widehat{p}$ can be defined taking R as the state space (which, because R is entire, is just an extension of p in the sense of the definition

[10] A relation $R : V \longleftarrow U$ is *functional* if every $u \in U$ is related to at most one $v \in V$ and *entire* if for all $u \in U$, there exists some $v \in V$ such that $\langle u, v\rangle \in R$.

above) and the rule $\langle u', v' \rangle \in_{\mathsf{T}} a_{\widehat{p}}\langle\langle u, v\rangle, i\rangle$ iff $u' \in_{\mathsf{T}} a_p\langle u, i\rangle \wedge v' \in_{\mathsf{T}} a_q\langle v, i\rangle$, for all $i \in I$, as its dynamics. With this construction we prove that

Lemma 11 *(Soundness) To prove $q \trianglelefteq p$ it is sufficient to exhibit an entire simulation R such that $\langle u_p, u_q\rangle \in R$.*

PROOF. If R is functional the result follows from lemma 9. Otherwise construct $\widehat{p}$ as above: clearly p is bisimilar to $\widehat{p}$ and the graph of projection π_2 from its state space to V defines a simulation between $\widehat{p}$ and q. By definition, $p \sim \widehat{p}$ and the existence of a (seed-preserving) forward morphism from $\widehat{p}$ to q entails $q \trianglelefteq p$.

$\square$

Finally notice that, although $\trianglelefteq$ is transitive, it is not always the case that simulations are closed under (relational) composition.

To illustrate the simulation proof technique we shall prove now another refinement law in the component calculus.

Lemma 12 *For any refinement preorder $\leq$, relation $\trianglelefteq$ is monotonic with respect to parallel composition,* i.e.*,*

$$q \boxtimes t \;\trianglelefteq\; p \boxtimes r \tag{48}$$

whenever $q \trianglelefteq p$ and $t \trianglelefteq r$.

PROOF. Let R_1 and R_2 be the simulation relations witnessing facts $q \trianglelefteq p$ and $t \trianglelefteq r$, respectively. Our aim is to build a relation $R \subseteq (U_p \times U_r) \times (U_q \times U_t)$ such that

$$\langle\langle u, v\rangle, \langle u', v'\rangle\rangle \in R \Rightarrow \langle \overline{a}_{p\boxtimes r}\,\langle u, v\rangle, \overline{a}_{q\boxtimes t}\,\langle u', v'\rangle\rangle \;\in \leq \cdot Rel_{\leq}(\mathsf{B}(\mathsf{Id}\times(O\times O'))^{I\times I'})(R)\cdot \leq$$

Define

$$R = \{\langle\langle u, v\rangle, \langle u', v'\rangle\rangle \mid \langle u, u'\rangle \in R_1 \wedge \langle v, v'\rangle \in R_2\}$$

let a pair $\langle\langle u, v\rangle, \langle u', v'\rangle\rangle \in R$. This implies:

$$
\begin{aligned}
& \langle \overline{a}_{p \boxtimes r} \, \langle u, v \rangle, \overline{a}_{q \boxtimes t} \, \langle u', v' \rangle \rangle \;\in\; \leq \cdot Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times (O \times O')^{I \times I'}))(R) \cdot \leq \\
\equiv \quad & \{ \text{ for all } \langle i, j \rangle \in I \times I' \} \\
& \langle a_{p \boxtimes r} \, \langle \langle u, v \rangle, \langle i, j \rangle \rangle, a_{q \boxtimes t} \, \langle \langle u', v' \rangle, \langle i, j \rangle \rangle \rangle \;\in \\
& \qquad \leq \cdot Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times (O \times O')))(R) \cdot \leq \\
\equiv \quad & \{ \boxtimes \text{ definition} \} \\
& \langle \mathsf{Bm} \cdot \delta_l \, \langle a_p \, \langle u, i \rangle, a_r \, \langle v, j \rangle \rangle, \mathsf{Bm} \cdot \delta_l \, \langle a_q \, \langle u', i \rangle, a_t \, \langle v', j \rangle \rangle \rangle \;\in \\
& \qquad \leq \cdot Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times (O \times O')))(R) \cdot \leq \\
\equiv \quad & \{ \text{ definition of } R, \text{ composite } \mathsf{Bm} \cdot \delta_l \text{ is } \textit{structural} \} \\
& \langle \langle a_p \, \langle u, i \rangle, a_r \, \langle v, j \rangle \rangle, \langle a_q \, \langle u', i \rangle, a_t \, \langle v', j \rangle \rangle \rangle \;\in \\
& \qquad \leq \times \leq \cdot \, (Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times O))(R_1) \times Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times O'))(R_2)) \, \cdot \leq \times \leq \\
\equiv \quad & \{ \text{ product of binary relations} \} \\
& \langle a_p \, \langle u, i \rangle, a_q \, \langle u', i \rangle \rangle \;\in\; \leq \cdot Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times O))(R_1) \cdot \leq \;\; \wedge \\
& \qquad \langle a_r \, \langle v, j \rangle, a_t \, \langle v', j \rangle \rangle \;\in\; \leq \cdot Rel_{\leq}(\mathsf{B}(\mathsf{Id} \times O))(R_2) \cdot \leq \\
\equiv \quad & \{ \text{ assumption} \} \\
& \text{TRUE}
\end{aligned}
$$

□

Note that similar *monotonicity* laws can be obtained along the same lines for both *pipeline* composition and the remaining tensors.

8 Conclusion and Future Work

This paper introduced two levels of refinement for (state-based) components. In particular, the notion of *behavioural* refinement parametric on a model of behaviour captured by a strong monad B is, to the best of our knowledge, new. It is *generic* enough to capture a number of situations, depending on both B and the refinement preorder adopted. Non determinism reduction is just one possibility among many others. Also note that Poll's notion of *behavioural subtyping* in [14], at the model level, emerges as a particular instantiation.

A comparison with the mainstream literature on refinement in state based formalisms (*e.g.*, [21] or [22]) places the approach proposed here as a generalisation of a particular path in such theories. Generalisation in the sense that components' behaviour is taken as a parameter (formalised as a strong commutative monad B) whereas others adopt from the outset a relational specification framework (which corresponds to instantiating B with the finite powerset monad). On the other hand we limit ourselves to functions, instead of relations, as witnesses of refinement situations. Because our starting point was a notion of software component modelled as a coalgebra in Set, morphisms are just functions between coalgebra carriers. In order to be able to capture more general approaches to refinement the results presented in this paper have to be lifted to a broader base category — that of coalgebras over Rel, the category of sets and binary relations. We believe that a suitable generalisation of forward and backward refinement in a coalgebraic setting, combined with the economy of pointfree style calculations, will provide new insights in structuring and classifying existing approaches to refinement of state based specifications.

We are currently working on such a generalisation building on a theory of *generic transposition* proposed in [19]. This extends the usual power-transpose, a device aimed at developing relational algebra *via* the algebra of functions [23], to different classes of datatype arrows. Such transposed arrows have the structure of F-coalgebras, for F a datatype with membership, framed in a monadic structure wherever F is a monad.

On the application side our current concern is the full development of a refinement calculus for software components and its application to the proof of consistency of static and dynamic UML diagrams in the context of [24]. Whether this approach scales up to be useful in the classification and transformation of software *architectures* [25] remains a research question.

References

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

[2] J. M. Spivey, The Z Notation: A Reference Manual (2nd ed), Series in Computer Science, Prentice-Hall International, 1992.

[3] J. R. Abrial, The B Book: Assigning Programs to Meanings, Cambridge University Press, 1996.

[4] C. Szyperski, Component Software, Beyond Object-Oriented Programming, Addison-Wesley, 1998.

[5] P. Wadler, K. Weihe, Component-based programming under different paradigms, Tech. rep., Dagstuhl Seminar 99081 (February 1999).

[6] J. Rutten, Universal coalgebra: A theory of systems, Theor. Comp. Sci. 249 (1) (2000) 3–80, (Revised version of CWI Techn. Rep. CS-R9652, 1996).

[7] L. S. Barbosa, J. N. Oliveira, State-based components made generic, in: H. P. Gumm (Ed.), CMCS'03, Elect. Notes in Theor. Comp. Sci., Vol. 82.1, 2003.

[8] L. S. Barbosa, Towards a Calculus of State-based Software Components, Journal of Universal Computer Science 9 (8) (2003) 891–909.

[9] C. A. R. Hoare, Proof of correctness of data representations, Acta Informatica 1 (1972) 271–281.

[10] B. Liskov, Data abstraction and hierarchy, SIGPLAN Notices 23 (3).

[11] M. Fokkinga, R. Eshuis, Comparing refinements for failure and bisimulation semantics, Tech. rep., Faculty of Computing Science, Enschede (2000).

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

[13] C. Bolton, J. Davies, Using relational and behavioural semantics in the verification of object models, in: S. F. Smith, C. L. Talcott (Eds.), FMOODS'2000 - Formal Methods for Open Object-Oriented Distributed Systems, Kluwer Academic Publishers, Stanford, USA, 2000, pp. 163–182.

[14] E. Poll, A coalgebraic semantics of subtyping, Theorectical Informatica and Apllications 35 (1) (2001) 61–82.

[15] A. Mikhajlova, E. Sekerinski, Class refinement and interface refinement in object-oriented program, in: Proc. 4th Int. Formal Methods Europe Symposium, Springer Lect. Notes Comp. Sci. (1313), 1997, pp. 82–101.

[16] P. F. Hoogendijk, A generic theory of datatypes, Ph.D. thesis, Department of Computing Science, Eindhoven University of Technology (1996).

[17] R. C. Backhouse, P. F. Hoogendijk, Elements of a relational theory of datatypes, in: B. Möller, H. Partsch, S. Schuman (Eds.), Formal Program Development, Springer Lect. Notes Comp. Sci. (755), 1993, pp. 7–42.

[18] B. Jacobs, J. Hughes, Simulations in coalgebra, in: H. P. Gumm (Ed.), CMCS'03, Elect. Notes in Theor. Comp. Sci., Vol. 82.1, Warsaw, 2003.

[19] J. N. Oliveira, C. J. Rodrigues, Transposing relations: From *Maybe* functions to hash tables, in: D. Kozen (Ed.), 7th International Conference on Mathematics of Program Construction, Springer Lect. Notes Comp. Sci. (3125), 2004, pp. 334–356.

[20] M. Abadi, L. Lamport, The existence of refinement mappings, Theor. Comp. Sci. 82 (2) (1991) 253–284.

[21] W.-P. d. Roever, K. Engelhardt, Data Refinement: Model-Oriented Proof Methods and their Comparison, Vol. 47 of Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1998.

[22] J. Derrick, E. Boiten, Refinement in Z and Object-Z: Foundations and Advanced Applications, FACIT, Springer-Verlag, 2001.

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

[24] S. Meng, B. K. Aichernig, L. S. Barbosa, Z. Naixiao, A Coalgebraic Semantic Framework for Component Based Development in UML, in: L. Birkedal (Ed.), Proc. Int. Conf. on Category Theory and Computer Science (CTCS'04), ENTCS (to appear), Elsevier, 2004.

[25] R. Allen, D. Garlan, A formal basis for architectural connection, ACM TOSEM 6 (3) (1997) 213–249.