The United Nations
University

UNU/IIST

International Institute for
Software Technology

$\mathrm{Coalg}_{\mathrm{KPF}}$: Towards a Coalgebraic Calculus for Component-Based Systems

Sun Meng and Bernhard K. Aichernig

UNU/IIST Report No. 271

$\mathcal{R}$

UNU/IIST and UNU/IIST Reports

UNU/IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macau, and was founded in 1991. It started operations in July 1992. UNU/IIST is jointly funded by the Governor of Macau and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endowment Fund. As well as providing two-thirds of the endowment fund, the Macau authorities also supply UNU/IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU/IIST is to assist developing countries in the application and development of software technology.

UNU/IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU/IIST, and
7. Dissemination, in which UNU/IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU/IIST is on formal methods for software development. UNU/IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU/IIST produces a report series. Reports are either Research $\boxed{\text{R}}$, Technical $\boxed{\text{T}}$, Compendia $\boxed{\text{C}}$ or Administrative $\boxed{\text{A}}$. They are records of UNU/IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU/IIST at P.O. Box 3058, Macau or visit UNU/IIST's home page: http://www.iist.unu.edu, if you would like to know more about UNU/IIST and its report series.

Chris George, Acting Director

The United Nations University

UNU/IIST

International Institute for
Software Technology

P.O. Box 3058
Macau

$\mathrm{Coalg}_{\mathrm{KPF}}$: Towards a Coalgebraic Calculus for Component-Based Systems

Sun Meng and Bernhard K. Aichernig

Abstract

Coalgebras of Kripke Polynomial Functors have been widely used in modelling various kinds of systems. In this paper, we give a category $\mathbf{Coalg}_{KPF}$ which consists of coalgebras of KPFs. Then we present a set of constructions like sequential and parallel composition in a subcategory of $\mathbf{Coalg}_{KPF}$ for a restricted family of KPFs by exploiting the canonical operations in category theory. A family of algebraic laws for the properties being satisfied by these operations is provided.

Sun Meng is a Fellow at UNU/IIST on leave from the School of Mathematical Science of Beijing University, China, where he is a Ph.D candidate. His research interest include category theory, coalgebra theory, Object-Oriented method, formal method in software development, and formal semantics for modeling languages. His email address is sm@iist.unu.edu.

Bernhard K. Aichernig is a Research Fellow at UNU/IIST. He is also an assistant professor at the Institute for software Technology at the Graz University of Technology, Austria. His research interests include the synergies of testing and formal development methods, techniques of refinement, and requirements engineering supported by formal specification languages. His email address is bka@iist.unu.edu.

Contents

1 Introduction **1**

2 Preliminaries **2**

3 The Category Coalg$_{\mathbf{KPF}}$ **6**

4 The Coalgebraic Calculus **8**

4.1 Restriction, View, Exclusion and Hiding . . . 9

4.2 Relabelling . . . 11

4.3 Sequential Composition . . . 12

4.4 Choice . . . 16

4.5 Parallel Composition . . . 19

5 Conclusion and Future Work **22**

1 Introduction

The component-based software development paradigm [38], as an emerging field with promising solutions for dealing with the rapidly changing requirements of software applications, retains from object-orientation the basic principle of encapsulation of data and code, but shifts the emphasis from inheritance to composition to avoid the interference between the former and encapsulation and, is getting accepted in software industry as the cornerstone of software engineering. However, as happened before with object-orientation, component technology has become popular before consensual definitions and principles, let alone formal foundations, have been put forward.

Over the last few years there has been a growing interest in the theory of coalgebras, motivated by the fact that a great variety of state-based dynamical systems, like transition systems, automata, process calculi and classes in object-oriented languages can be captured uniformly in an implementation independent way as coalgebras, see [22, 36] as references for an introduction to coalgebra theory and [6, 16, 17, 18, 20, 21, 33, 35] for related work.

Coalgebraic frameworks can be integrated with algebraic techniques together to obtain a framework for the specification of systems having both an observational and computational properties. Related work can be found in [10, 11, 12] which is based on the work of unifying operational semantics and denotational semantics in [40, 41] and uses equations and coequations to formalize the equivalence of computations and invariants on the structure and evolution of systems. The coinduction principle makes proving an equation being satisfied up to observations can be reduced to build up a bisimulation relationship between the two terms involved in the equation.

An alternative calculational approach for proofs is used in [5, 6] where proofs of properties can be done in an equational and pointfree calculational style, therefore circumventing the explicit construction of bisimulations.

Coalgebras has been applied successfully as the semantic framework for both observations and computations, and is proposed as the semantic model for components in [4]. However, while considering the large, complex systems such as air-traffic control system, the interaction and coordination between different components becomes a more critical problem. Motivated by the calculational approach proposed in [5, 6], in this paper, we present a cofibred category of coalgebras over a family of interface functors which can describe the behavioural patterns of a large variety of systems and give a semantic basis for composition and communication between different coalgebras by exploiting the constructions in this category, including a set of operations corresponding to the encapsulation principles, relabelling, sequential composition, external choice, and parallel composition, etc. which are the typical combinators in process calculus such as CCS [26] and CSP [14]. Such operators model the global structure of the whole target system and therefore provide facilities for constructing a large system from its components. A family of algebraic laws are provided for describing the properties satisfied by these operations.

An advantage of this approach is that the structural aspects of the behavior models of systems (for example, inactive vs active, deterministic vs nondeterministic, total vs partial, etc.), described by the signature functors, are separated clearly from the interaction structure which defines the interaction and synchronization among systems (or subsystems, components) and is described by the operations in the category of coalgebras.

Since the behavior semantics of each component can be described by the corresponding (final) coalgebra, the semantics of a larger system can be determined by the operations in the category of coalgebras which give the semantics of the interactions among its components and the coalgebras of the components, that means, the semantics of its syntatic constituents. This approach specifies the *compositionality* and *structural transparency* of systems: a system composed of components and connections among them can be considered as a component again and connected to other components, and thereby the internal state space of the system is hidden again to outside.

The categorical notions also relate coalgebras with different computational models naturally. [42] provides the abstract characterisations of different computational models and formal means of translating between them by exploiting category theory. We can identify different models with special coalgebras and use functors to include them into the category of coalgebras. For a more complete account on category theory, [8] can be used as references.

The structure of this paper is organized as follows: In Section 2, we review some basic concepts in coalgebra theory and give some examples of coalgebras to make the paper self-contained. A category of coalgebras for Kripke polynomial functors is defined in Section 3. In Section 4, we define a set of operations on coalgebra in the category and provide a set of algebraic laws for the operations to describe the properties that they should satisfy. Finally conclusion and discussions about future work are given in Section 5.

2 Preliminaries

Let $\mathbf{Set}$ be the category of sets and functions. The functors we use in this paper to describe component signatures is a particular collection of functors which are called *Kripke polynomial functors* (KPFs). Such functors are the endofunctors over the category $\mathbf{Set}$ and are finitely built up from the following:

$$F(X) = C \mid X \mid F_1(X) \times F_2(X) \mid F_1(X) + F_2(X) \mid C \to F_1(X) \mid \mathcal{P}_f(F_1(X)) \tag{1}$$

where C is an arbitrary non-empty constant set, F_1, F_2 are two previously defined KPFs. The *product* of two sets A, B is a set $A \times B$ in $\mathbf{Set}$ together with two projections $\pi_1 : A \times B \to A, \pi_2 : A \times B \to B$ The *coproduct (sum)* of A and B is an object $A + B$ together with two injections $\iota_1 : A \to A + B, \iota_2 : B \to A + B$. The *exponent* A^B (or $B \to A$) is used for the collection of functions from a set B to A. The powerset functor $\mathcal{P}$ sends a set to the set of its subsets: $\mathcal{P}(S) = \{V \mid V \subseteq S\}$, and a function $f : S \to T$ is mapped to $\mathcal{P}(f) : \mathcal{P}(S) \to \mathcal{P}(T)$ which is defined as $\mathcal{P}(f)(V) = f(V)$ for any $V \subseteq S$. $\mathcal{P}_f$ is the finite powerset functor $\mathcal{P}(S) = \{V \mid V \subseteq S$ and V is finite$\}$. Finiteness property is needed because the final coalgebra for ordinary powerset functor does not exist.

Let F be an arbitrary functor (In fact, all the functors being used in this paper are KPFs as defined in (1)). A F-*Coalgebra* is a pair $c = (U, \alpha)$ consisting of a set U and a function $\alpha : U \to F(U)$. Like for algebra, the set U is called the *carrier* of the coalgebra, and the function α is called the *(transition) structure* or *operation* of the coalgebra. And because coalgebras are often used to describe state-based dynamic systems, the carrier U is also called the *state space* of the coalgebra. Sometimes we may also use a triple $(U, \alpha, u_0 \in U)$ for a coalgebra where u_0 is called its *initial state* (especially in the interpretation of classes in object-oriented languages, see e.g. [16, 17] for related work).

Suppose $c = (U, \alpha)$ and $c' = (U', \alpha')$ be two F-coalgebras, then a F-*homomorphism* from c to c' is a function $f : U \to U'$, such that the following diagram commutes.

$$\begin{tikzcd} U \arrow[r, "f"] \arrow[d, "\alpha"'] & U' \arrow[d, "\alpha'"] \\ F(U) \arrow[r, "F(f)"'] & F(U') \end{tikzcd}$$

That is, $\alpha' \circ f = F(f) \circ \alpha$. Concerning the initial states u_0 and u'_0 respectively, f is called *initial state preserving* if $f(u_0) = u'_0$.

The F-coalgebras and F-homomorphisms together form a category being widely used in the research of coalgebra theory, which we denote by $\mathbf{Coalg}_F$.

An important notion in coalgebra theory is that of *behavior equality*, which is formally described by the concept of *bisimulation*, a notion also being widely used in concurrency theory [26, 27] and transition systems [1] to capture the notion of observational equivalence. Informally, two states of a F-coalgebra (or of two different F-coalgebras) are related by a bisimulation relationship if they can not be distinguished by observations and this is preserved along all possible transitions. Its formal definition is given as follows.

Let $F : \mathbf{Set} \to \mathbf{Set}$ be an arbitrary functor, $c = (U, \alpha)$ and $c' = (U', \alpha')$ be two F-coalgebras. A relation $R \subseteq U \times U'$ is called a F-*bisimulation* (between c and c') if there exists a F-coalgebra $(R, \alpha_R : R \to F(R))$ such that the projections from R to U and U' are F-homomorphisms:

$$\begin{tikzcd} U \arrow[d, "\alpha"'] & R \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\alpha_R"] & U' \arrow[d, "\alpha'"] \\ F(U) & F(R) \arrow[l, "F(\pi_1)"] \arrow[r, "F(\pi_2)"'] & F(U') \end{tikzcd}$$

[23] provides an alternative definition for bisimulation of coalgebras by exploiting the concept of relation lifting and shows that for polynomial functors both approaches yield equivalent notions of bisimulation.

Two elements $u \in U$ and $u' \in U'$ are *bisimilar* if there is a bisimulation relation $R \subseteq U \times U'$ with $(u, u') \in R$. In this case we write it as $u \approx u'$.

Several examples of coalgebras will be mentioned in the following.

Example 2.1 Consider *labelled transition systems* (LTS) $(S, \rightarrow, A)$ as an example, which consists of a set S of states, a transition relationship $\rightarrow \subseteq S \times A \times S$ and a set A of labels [31]. Define a KPF $B = \mathcal{P}(A \times Id)$ [1], for any set S,

$$B(S) = \mathcal{P}(A \times S) = \{V \mid V \subseteq A \times S\}$$

Then an A-LTS can be equivalently represented as a B-coalgebra (S, α) where

$$\alpha : S \rightarrow B(S) \qquad s \mapsto \{\langle a, s' \rangle \mid s \xrightarrow{a} s'\}$$

Now we consider a simple vending machine being used in [14] as another example.

Example 2.2 *(X3 in [14] Section 1.1.3)*A vending machine $VMCT$ [2] that can serve a customer with either chocolate or toffee on each transaction is described by the process expression

$$VMCT = \mu X.coin \rightarrow (choc \rightarrow X \mid toffee \rightarrow X)$$

Such a vending machine can be represented as a coalgebra (S, α) of functor $B = A \overset{\sim}{\rightarrow} Id$, [3] where S has two inhabitants $\{s_1, s_2\}$, $A = \{coin, choc, toffee\}$ is the set of actions. The transition α is defined as follows:

$$\begin{aligned}
&\alpha(s_1) : A \overset{\sim}{\rightarrow} S, \qquad dom(\alpha(s_1)) = \{coin\} \\
&\alpha(s_1)(coin) = s_2 \\
&\alpha(s_2) : A \overset{\sim}{\rightarrow} S, \qquad dom(\alpha(s_2)) = \{choc, toffee\} \\
&\alpha(s_2)(choc) = \alpha(s_2)(toffee) = s_1
\end{aligned}$$

Classes in object-oriented programming languages, which are built around a hidden state space and can only be observed and modified via specified observers (attributes) and operations (methods), can be described by coalgebras.

Example 2.3 Consider a class of banking account ACCOUNT which is represented as a RSL [39] specification as follows:

[1] Such a functor gives an "active" interpretation of the system behavior, in which a transition $s \xrightarrow{a} s'$ is informally characterised as "state s evolves to s' by performing an action a". A "reactive" interpretation can be got by define the functor as $B = (\mathcal{P}(Id))^A$ where $s \xrightarrow{a} s'$ means "s reacts to an external stimulus a and evolves to s'".

[2] From this example we can see how recursive expressions can be defined in the coalgebraic setting.

[3] here $\overset{\sim}{\rightarrow}$ is used for partial functions, which describes the possibility of deadlock and can be captured by the functor $B = A \rightarrow Id + 1$.

scheme ACCOUNT =
  class
    type
      Account
    value
      balance : Account → **Nat**,
      credit : Account × **Nat** → Account,
      debit : Account × **Nat** $\xrightarrow{\sim}$ Account,
      account : Account
    axiom
      ∀ a : Account, m : **Nat** •
        balance(credit(a,m)) ≡ balance(a) + m,
      ∀ a : Account, m : **Nat** •
        balance(debit(a,m)) ≡ balance(a) − m
        pre balance(a) ≥ m,
      balance(account) ≡ 0
  end

The attributes and methods of this class can be combined into a single function:

$$\langle balance, credit, debit\rangle : S \to \mathbb{N} \times S^{\mathbb{N}} \times (S+1)^{\mathbb{N}}$$

which forms a coalgebra for functor $B = \mathbb{N} \times Id^{\mathbb{N}} \times (Id+1)^{\mathbb{N}}$ on the state space S while the constructor $account$ can be described by an initial state s_0 (see [16, 17, 33, 34] for a more detailed interpretation on the coalgebraic description of classes).

We have seen that the functors can be used to describe the signature of computation and a coalgebra of the functor gives an implementation for it. However, to interpret a system correctly, not only the computation inside a system component, but also the coordination between different components need to be described by the semantic model. Therefore, the relationship between different functors need to be considered. So we will define a "total" category $\mathbf{Coalg}_{KPF}$ which encompasses the categories $\mathbf{Coalg}_F$ for all the possible signature functors F, and also admits morphisms between coalgebras for different functors in the next section.

Finally, before we come to the category $\mathbf{Coalg}_{KPF}$, we introduce another notion: *monad*, which plays an important role in our definition for the operations in Section 4. Monads, sometimes called triples [7, 9] are closely related to the notion of adjunctions in category theory and being used as a way of modelling computational types by Moggi in [28, 29].

A monad B over a category $\mathbf{C}$ is a triple (B, η, μ) where B is an endofunctor on $\mathbf{C}$, $\eta : id_{\mathbf{C}} \to B$ and

$\mu : B^2 \to B$ are two natural transformations such that the following diagrams commute:

$$\begin{tikzcd} B^3U \arrow[r, "\mu_{BU}"] \arrow[d, "B_{\mu_U}"'] & B^2U \arrow[d, "\mu_U"] \\ B^2U \arrow[r, "\mu_U"] & BU \end{tikzcd} \qquad \begin{tikzcd} BU \arrow[r, "\eta_{BU}"] \arrow[dr, "id_{BU}"'] & B^2U \arrow[d, "\mu_U"] & BU \arrow[l, "B_{\eta_U}"'] \arrow[dl, "id_{BU}"] \\ & BU & \end{tikzcd}$$

Thinking of B as a behavior pattern, then η_U represents the minimal structure whereby a value $u \in U$ is embedded in $\mathsf{B}U$. On the other hand, μ providing a way to view a B effect of another B effect as still a B effect.

In order to handle the presence of the environment of a system and distribute it along the computation of the system, monads used in this paper are required to be strong. A monad is *strong* if it comes equipped with a natural transformation $\tau_r^{\mathsf{B}} : \mathsf{B} \times - \longrightarrow \mathsf{B}(Id \times -)$, called a *right strength* verifying some coherence conditions [13]. A *left strength* can be dually given by $\tau_l^{\mathsf{B}} : - \times \mathsf{B} \longrightarrow \mathsf{B}(- \times Id)$. In [30], a definition of strength for any polynomial functor is presented by reasoning inductively on the construction of polynomial functors.

For two arbitrary monads B_1 and B_2, a simple composition of their corresponding functors does not result necessarily in a new monad. In order to define a new monad it is necessary to verify the existence of a natural transformation $\gamma : \mathsf{B}_2\mathsf{B}_1 \longrightarrow \mathsf{B}_1\mathsf{B}_2$ satisfying a number of coherence conditions, see [4] for a more detailed discussion.

Given two monads $\mathsf{B}_1 = (B_1, \eta_1, \mu_1)$ and $\mathsf{B}_2 = (B_2, \eta_2, \mu_2)$ on the category $\mathbf{Set}$, a morphism between them is a strong natural transformation $\alpha : \mathsf{B}_1 \longrightarrow \mathsf{B}_2$ making the following diagrams commute:

$$\begin{tikzcd} & \mathbf{Id} \arrow[dl, "\eta_1"'] \arrow[dr, "\eta_2"] & \\ B_1 \arrow[rr, "\alpha"'] & & B_2 \end{tikzcd} \qquad \begin{tikzcd} B_1B_1 \arrow[r, "\alpha^2"] \arrow[d, "\mu_1"'] & B_2B_2 \arrow[d, "\mu_2"] \\ B_1 \arrow[r, "\alpha"'] & B_2 \end{tikzcd}$$

Finally, note that monad morphisms compose and there exists a trivial identity morphism for each monad. Therefore, monads and their morphisms for a category.

3 The Category $\mathbf{Coalg_{KPF}}$

We have introduced the notions in coalgebra theory previously. In this section, we will define a cofibred category of coalgebras for $\mathbf{Coalg}_{KPF}$ over the category $\mathbf{KPF}$ of Kripke Polynomial Functors, which is the category of coalgebras for a family of functors generated by applying (1).

The functors we use in this paper to describe coalgebraic signatures are the endofunctors on the category $\mathbf{Set}$, and together with the natural transformations between them form a category $[\mathbf{Set}, \mathbf{Set}]$. For every object of this category, that is, an endofunctor F on the category $\mathbf{Set}$, we can obtain the category $\mathbf{Coalg}_F$ of all F-coalgebras.

The category $\mathbf{Coalg}_{KPF}$ is defined as the "total" category which encompasses the possible categories $\mathbf{Coalg}_F$ for the polynomial functor F, and also admits morphisms between coalgebras for different functors.

Proposition 3.1 *Let $\mathbf{KPF}$ be the subcategory of the category $[\mathbf{Set}, \mathbf{Set}]$, and consists of the polynomial functors and natural transformations between them. Then for two such functors F and G in $\mathbf{KPF}$, a natural transformation $\eta : F \to G$ allows us to view every F-coalgebra $\alpha : U \to F(U)$ as a G-coalgebra $\eta_U \circ \alpha : U \to G(U)$. Take coalgebras for the functors in $\mathbf{KPF}$ as objects and for two coalgebras $(U, \alpha : U \to F(U))$ and $(V, \beta : V \to G(V))$, take (σ, η) as the arrow between them such that the following diagram commutes:*

$$\begin{tikzcd} U \arrow[rr, "\sigma"] \arrow[d, "\alpha"'] & & V \arrow[d, "\beta"] \\ F(U) \arrow[r, "\eta_U"] & G(U) \arrow[r, "G(\sigma)"] & G(V) \end{tikzcd}$$

Then we can get the category $\mathbf{Coalg}_{KPF}$.

Proof: The composition of two arrows $f_1 = (\sigma_1, \eta_1) : c \to d$ and $f_2 = (\sigma_2, \eta_2) : d \to e$ is $f_2 \circ f_1 = (\sigma_2 \circ \sigma_1, \eta_1 \circ \eta_2) : c \to e$. According to the property of natural transformation, we can easily get that the *Associative Law* is satisfied. It is even easier to show that there is a trivial identity morphism for each coalgebra $c = (U, \alpha : U \to F(U), u_0)$ which is defined by (id_U, id_F) where id_U is the identity function on the state space U and id_F is the identity natural transformation from F to itself, such that the *Identity Law* is satisfied.

We can know that the category $\mathbf{Coalg}_{KPF}$ is a cofibration over the category $\mathbf{KPF}$ where we have a functor $p : \mathbf{Coalg}_{KPF} \to \mathbf{KPF}$ which maps every F-coalgebra to the functor F. The cofibration provides us a single total category $\mathbf{Coalg}_{KPF}$ which contains the transition structures corresponding to different functors and allow us to relate coalgebras of different functors within just one category.

Here we recall some standard terminology in category theory [8, 19]. For a total category $\mathbf{C_F}$ and the base category $\mathbf{F}$ of this cofibration, an object $c \in \mathbf{C_F}$ and an arrow $f : c \to d$ with $pc = F$ and $pf = \lambda$ are called *over* F and *over* λ respectively.

For two F-coalgebras of the same functor F, the morphism between them as defined previously is $f = (\sigma, \langle\rangle_F)$ which can be identified with the F-homomorphism σ. Therefore, for every functor F, the category of F-coalgebras is a subcategory of $\mathbf{Coalg}_{KPF}$. Generally, for any set of polynomial functors $\{F_i\}_{i=1,\ldots,n}$, the category of all the F_i-coalgebras forms a subcategory of $\mathbf{Coalg}_{KPF}$. In the following section, we will specialise our research to one such particular subcategory of a special class of functors and describe how larger coalgebras can be built up from applying the operators defined in the category to primitive coalgebras.

4 The Coalgebraic Calculus

In this section, we will define a set of operations on coalgebras in a subcategory of the category $\mathbf{Coalg}_{KPF}$ via some fundamental categorical constructions, where a categorical analysis of coalgebras is given and then the operations are analyzed categorically. The properties of these operations are presented by a set of algebraic laws. Note that most of the laws presented here hold only up to isomorphism, since the presence of internal state prevents most of the properties to hold up to equality.

Our first concern is precisely the shape of the functor F being used as the signature for coalgebras in the subcategory. In fact, different shapes of functor F describes the behavior signatures for different systems. The one we are interested in is with the form

$$F^{\mathsf{B}} = A \times \mathsf{B}(O \times -)^I \tag{2}$$

which is parameterized by a (strong) monad B being used to capture the behavior pattern of the corresponding coalgebras (For example, nondeterminism behavior can be represented by the (finite) powerset monad $\mathsf{B} = \mathcal{P}(Id)$, and partiality can be represented by the monad $\mathsf{B} = Id + 1$). Such functors are extremely simple. However, it seems that coalgebras of such functors provide models for a broad range of systems. Especially, the polynomial functors used by Jacobs [16, 17] for the semantics of classes in object oriented programming languages can also be represented as the functor F^{B}. The sets A, I, O in (2) are used for the values of observations (attributes), inputs and outputs correspondingly and all of them can be empty. And every F^{B}-coalgebra $c = (U, \alpha : U \to F^{\mathsf{B}}(U))$ can be represented as a pair $\langle o_c, a_c \rangle$ where $o_c : U \to A$ is the *observer* and $a_c : U \times I \to \mathsf{B}(O \times U)$ is the *action* (method or operation).

Generally, a coalgebraic specification [21] may include more than one KPFs for signatures (see the specification in Example 2.3 as an example). Therefore, the functor (2) will be extended to the following

$$F = \prod_k A_k \times \prod_l \mathsf{B}_l(O_l \times -)^{I_l} \tag{3}$$

to match more patterns. However, for simplicity, we will still use the functor with the form (2) for the functor F in (3) without confusion where

$$A = \prod_k A_k, \qquad I = \sum_l \langle l, I_l \rangle, \qquad O = \sum_l O_l$$

and the behavior of a F-coalgebra is described by

$$\mathsf{B}(O \times -)^I = \text{case } I \text{ of } [\langle l, I_l \rangle \mapsto \mathsf{B}_l(O_l \times -)^{I_l}]$$

Here we use k, l to mark different observers and actions.

If we do not consider the attributes, coalgebras can be viewed as the collections of state transition functions which can be equally represented by LTSs, and corresponds to the Kripke models/frames in modal logic [24, 25, 32]. However, the presence of attributes extends the one-dimension transition model into a two-dimension model where both state transitions and observations on states are taken into account, as the following diagram shows:

$$
u_0 \xrightarrow{a_0} u_1 \xrightarrow{a_1} \ldots \xrightarrow{a_n} u_{n+1}
\qquad
\begin{tikzcd}
u_0 \arrow[r, "a_0"] \arrow[d, "o_0"] & u_1 \arrow[r, "a_1"] \arrow[d, "o_1"] & \ldots \arrow[r, "a_n"] & u_{n+1} \arrow[d, "o_{n+1}"] \\
at_0 & at_1 & \ldots & at_{n+1}
\end{tikzcd}
$$

where $at_i \in A, i = 0, 1, \ldots, n+1$ are the observable attribute values corresponding to the states respectively.

4.1 Restriction, View, Exclusion and Hiding

The first class of operations on coalgebras to be introduced includes the restriction [26, 14, 42] of behaviour of the action parts, the restriction of the visibility of the observers w.r.t. a seperation of views (public and private) and their complementation. Note that here the restriction of behaviour is defined as restriction to a subset of certain behaviours, which is complementary to the definition of restriction on a subset of actions to be concealed in process calculus such as CCS [26] (being defined as exclusion in this paper and corresponding to the private methods in object oriented languages). Given a coalgebra and an ingredient functor of its signature functor, the operation of restriction removes all transitions and observations that are not specified in the ingredient functor.

Definition 4.1 (Restriction and View) *Let $c = (U, \alpha : U \to F(U), u_0)$ be a F-coalgebra. Its observer and action components are $o_c : U \to A$ and $a_c : U \to H(U)$.*[4]

- *Given an action coalgebra $a' = (U, \alpha' : U \to H'(U), u_0)$ where H' satisfies that $\exists H'', s.t. H = H' \times H''$, for a coalgebra morphism $f = (id_U, \gamma) : a_c \to a'$ where $id_U : U \to U$, and $\gamma : H \to H'$ is a projection of functors, the restriction of c by f is defined as*

  $$res(c, f) \triangleq (o, f \circ a)$$

  which is written as $c \upharpoonright f$ for convenience;

- *Given an observer $o' : U \to A'$ where A' satisfies that $\exists A'', s.t. A = A' \times A''$, for a coalgebra morphism $g = (id_U, \pi) : o_c \to o'$ where $id_U : U \to U$, and $\pi : A \to A'$ is a projection, the view of c by g is defined as*

  $$view(c, g) \triangleq (g \circ o, a)$$

  which is written as $c \upharpoonleft g$.

[4] Here and in the following we use $A = \prod_k A_k$ for the type of observation values, and H for the behavior functor with the form $\prod_l \mathsf{B}_l(O_l \times -)^{I_l}$ as defined in (3).

The notation of restriction and view here together gives the formal interpretation of encapsulation, which shows the distinction of public and private properties in OO development [5]. In general, the transition structure of a F-coalgebra is restricted to a subset of allowed actions and observations which can only be accessed from outside and corresponds to a given coalgebraic subsignature (a functor F' which satisfies $\exists F'', s.t.F = F' \times F''$, that is, there exists some path from F to F' which can be got from projections, we use $F' \subseteq F$ to represent it[6]).

For an action signature $H' \subseteq H$, and a morphism $f : a_c \to a'$ as shown in the definition, a restriction $c \upharpoonright f$ can also be written as $c \upharpoonright H'$, and we call it a H'-restriction. Similarily, a view $c \upharpoonleft g$ induced by a morphism g from the constant functor A to $A' \subseteq A$ is also called an A'-view and written as $c \upharpoonleft A'$.

A natural consideration is to give the complementary operations of restriction and view, which are defined as the exclusion of actions, and hiding of observers, i.e. to make them invisible from outside.

Definition 4.2 (Exclusion and Hiding) *Let $c = (U, \alpha : U \to F(U), u_0)$ be a F-coalgebra. Its observer and action components are $o_c : U \to A$ and $a_c : U \to H(U)$.*

- *Given an action coalgebra $a' = (U, \alpha' : U \to H'(U), u_0)$ where H' satisfies that $\exists H'', s.t.H = H' \times H''$, for a coalgebra morphism $f = (id_U, \gamma) : a_c \to a'$ where $id_U : U \to U$, and $\gamma : H \to H'$ is a projection, the exclusion of c by f is defined as*

$$exc(c, f) \triangleq res(c, \bar{f})$$

where $\bar{f}$ represents the coalgebra morphism from a_c to the action coalgebra $a'' = (U, \alpha'' : U \to H''(U), u_0)$ which includes all the actions not in a', and we write it as $c \setminus f$ or $c \downharpoonright f$;

- *Given an observer $o' : U \to A'$ where A' satisfies that $\exists A'', s.t.A = A' \times A''$, for a coalgebra morphism $g = (id_U, \pi) : o_c \to o'$ where $id_U : U \to U$, and $\pi : A \to A'$ is a projection from the constant functor A to A', the hiding of c by g is defined as*

$$hide(c, g) \triangleq view(c, \bar{g})$$

where $\bar{g}$ represents the coalgebra morphism from o_c to the observer coalgebra $o'' : U \to A''$ which includes all the observations not in o' and we write it as $c \downharpoonleft f$.

The exclusion and hiding in Definition 4.2 can also be represented as $c \downharpoonright H'$ and $c \downharpoonleft A'$ respectively.

Now we give some basic properties of these operations, stated as algebraic laws. Suppose c be a F-coalgebra, where $F = A \times H$, A and H correspond to the observations and actions seperately.

Law 4.1 *Let $H'' \subseteq H' \subseteq H$. Then $c \upharpoonright H'' = (c \upharpoonright H') \upharpoonright H''$.*

[5] We seperate view from restriction here only to make the difference between observations and actions explicit, and similar for the seperation between exclusion and hiding.

[6] Note that the equation $F = F' \times F''$ is up to isomorphism.

Proof: Let $f = (id, \gamma)$ be the coalgebra morphism corresponding to the restriction from H to H', and $f' = (id, \gamma')$ be the coalgebra morphism corresponding to the restriction from H' to H''. Then

$$\begin{aligned} c \upharpoonright H'' =& c \upharpoonright (f' \circ f) \\ =& (c \upharpoonright f) \upharpoonright f' \\ =& (c \upharpoonright H') \upharpoonright H'' \end{aligned}$$

Law 4.2 *Let $A'' \subseteq A' \subseteq A$. Then $c \upharpoonleft A'' = (c \upharpoonleft A') \upharpoonleft A''$.*

Proof: Similar like Law 4.1.

Law 4.3 *Let $H' \subseteq H$ and $A' \subseteq A$. Then $(c \upharpoonleft A') \upharpoonright H' = (c \upharpoonright H') \upharpoonleft A'$.*

Proof: By the definition, let f, g are the corresponding restriction and view morphism, then

$$\begin{aligned} (c \upharpoonleft A') \upharpoonright H' =& (g \circ o, a) \upharpoonright f \\ =& (g \circ o, f \circ a) \\ =& (o, f \circ a) \upharpoonleft g \\ =& (c \upharpoonright f) \upharpoonleft g \\ =& (c \upharpoonright H') \upharpoonleft A' \end{aligned}$$

The proofs of the following laws are similar as those of Law 4.3 and Law 4.1.

Law 4.4 *Let $H' \subseteq H$ and $A' \subseteq A$. Then $(c \downharpoonleft A') \downharpoonright H' = (c \downharpoonright H') \downharpoonleft A'$.*

Law 4.5 *Let $H'' \subseteq H' \subseteq H$. Then $c \downharpoonright H'' = (c \downharpoonright H') \downharpoonright H''$.*

Law 4.6 *Let $A'' \subseteq A' \subseteq A$. Then $c \downharpoonleft A'' = (c \downharpoonleft A') \downharpoonleft A''$.*

4.2 Relabelling

In CCS, one can use relabelling operation $E[f]$ to make a copy of a process by applying the relabelling function f. In the coalgebraic context, a relabelling combinator between coalgebras can be defined as

Definition 4.3 **(Relabelling)** *Given a F-coalgebra $c = (U, \alpha : U \to F(U), u_0)$ and a natural transformation $\gamma : F \to F'$, the cocartesian lifting of γ gives rise to a F'-coalgebra*

$$c' = (U, \alpha' : U \to F(U) \to F'(U), u_0)$$

where the composition of α and γ_U just renames the output of α in $F(U)$ according to γ. The new coalgebra is written as $c' = c[\gamma]$ and called as the relabelling of c under γ.

It can be easily proved that relabelling preserves identity and composition.

Law 4.7 $c[id_F] = c$ *for* $c = (U, \alpha : U \to F(U), u_0)$.

Law 4.8 $(c[\gamma])[\gamma'] = c[\gamma \cdot \gamma']$.

4.3 Sequential Composition

A characteristic combinator which combines components into a system is sequential composition ("|" in CCS [26] or ";" in CSP [14] and Unifying Theory of Programming [15] [7]). Here we adopt the notation ";" for the sequential composition of coalgebras of functors with the form as defined in (2), where communication may happen between coalgebras. The output of the first component is used as the input of the second component in their communication. The corresponding definition in the coalgebra setting is as follows:

Definition 4.4 *Let* $c = (U_c, \alpha_c : U_c \to A_c \times \mathsf{B}_c(Z \times U_c)^I)$ *and* $d = (U_d, \alpha_d : U_d \to A_d \times \mathsf{B}_d(O \times U_d)^Z)$ *be two coalgebras. A sequential composition of them is formed by the following expression:*

$$c; d = (U, \alpha : U \to A \times \mathsf{B}(O \times U)^I)$$

where $U = U_c \times U_d$, $A = A_c \times A_d$, $\mathsf{B} = \mathsf{B}_c\mathsf{B}_d$[8].

$c; d = \langle o_{c;d}, a_{c;d} \rangle$ is a coalgebra which is given by

$$o_{c;d} : U_c \times U_d \xrightarrow{\langle o_c, o_d \rangle} A_c \times A_d$$

and

$$\begin{aligned}
a_{c;d} : (U_c \times U_d) \times I \quad & \xrightarrow{\times} \quad U_c \times I \times U_d \\
& \xrightarrow{a_c \times id_{U_d}} \quad \mathsf{B}_c(Z \times U_c) \times U_d \\
& \xrightarrow{\tau_r^{\mathsf{B}_c}} \quad \mathsf{B}_c(Z \times U_c \times U_d) \\
& \xrightarrow{\times} \quad \mathsf{B}_c(U_c \times U_d \times Z) \\
& \xrightarrow{\mathsf{B}_c(id_{U_c} \times a_d)} \quad \mathsf{B}_c(U_c \times \mathsf{B}_d(O \times U_d)) \\
& \xrightarrow{\tau_l^{\mathsf{B}_d}} \quad \mathsf{B}_c(\mathsf{B}_d(U_c \times O \times U_d)) \\
& \xrightarrow{\times} \quad \mathsf{B}_c(\mathsf{B}_d(O \times U_c \times U_d))
\end{aligned}$$

[7] In fact, these two operations are not exactly same. But they are same for those well-terminating agents. See Chapter 9 in [26] for more discussion on their relations.

[8] Note that it should be guaranteed that the two monads B_c and B_d are able to be composed in this definition.

Law 4.9 $(c;d);e = c;(d;e)$ *for coalgebras* $c = (U_c, \alpha_c : U_c \to A_c \times \mathsf{B}_c(Z \times U_c)^I)$, $d = (U_d, \alpha_d : U_d \to A_d \times \mathsf{B}_d(X \times U_d)^Z)$ *and* $e = (U_e, \alpha_e : U_e \to A_e \times \mathsf{B}_e(O \times U_e)^X)$.

Proof: We can easily know that the state spaces of the coalgebras on both sides are $U_c \times U_d \times U_e$, and the observations on both sides are equal. The equality of actions can be proved by applying the definition of the action for sequential composition seperately.

$$a_{(c;d);e} = (U_c \times U_d \times U_e) \times I \longrightarrow \mathsf{B}_c(\mathsf{B}_d(\mathsf{B}_e(O \times U_c \times U_d \times U_e))) = a_{c;(d;e)}$$

The left and right unit of sequential composition are presented in the following two laws.

Law 4.10 $\mathbf{1}^l_;; c \sim c$ *where* $\mathbf{1}^l_; = (\mathbf{1} = \{*\}, \langle id_1, id_I \rangle : \mathbf{1} \to \mathbf{1} \times (I \times \mathbf{1})^I)$ *for any* $c = (U, \alpha : U \to A \times \mathsf{B}(O \times U)^I)$.

Proof: $U_{\mathbf{1}^l_;;c} = \{*\} \times U$ which can be identified with U. Observations on both sides can also be identified because there is only one trivial observer in $\mathbf{1}^l_;$. The actions can be equalized as follows:

$$\begin{array}{rcl} a_{\mathbf{1}_;;c} = (\{*\} \times U) \times I & \xrightarrow{\times} & (\{*\} \times I) \times U \\ & \xrightarrow{a_{\mathbf{1}^l_;}} & (I \times \{*\}) \times U \\ & \xrightarrow{\times} & \{*\} \times (U \times I) \\ & \xrightarrow{a_c} & \{*\} \times \mathsf{B}(O \times U) \cong \mathsf{B}(O \times U) = a_c \end{array}$$

Law 4.11 $c; \mathbf{1}^r_; \sim c$ *where* $\mathbf{1}^r_; = (\mathbf{1} = \{*\}, \langle id_1, id_O \rangle : \mathbf{1} \to \mathbf{1} \times (O \times \mathbf{1})^O)$ *for any* $c = (U, \alpha : U \to A \times \mathsf{B}(O \times U)^I)$.

Proof: Similar as Law 4.10.

The following law for the distribution of relabelling on sequential composition can be easily proved.

Law 4.12 $(c;d)[f] = c[f];d[f]$.

Sequential composition of coalgebras provides a natural interpretation of the pipe-and-filter software architecture where each component has a set of inputs and a set of outputs. A component reads data streams from input channels and produces data streams to output channels. Every component in such a system is illustrated as a filter which has a local transformation from inputs to outputs. The best known examples of such systems include the programs written in the Unix shell, compilers and the examples in functional programming.

As an example, consider two components READER and WRITER and a connector between them as follows:

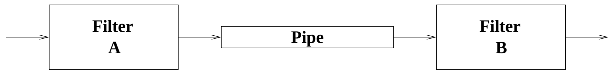

Figure 1: Pipe and Filters

```
scheme READER =
  class
    type
      Input,
      Mid
    channel
      input : Input,
      add : Mid
    value
      transformr: Input → Mid,
      reader : Unit → in input out add Unit
    axiom
      reader() ≡ let v=input? in add!(transformr(v)) end; reader()
  end
```

```
scheme WRITER =
  class
    type
      Mid,
      Output
    channel
      output : Output,
      get : Mid
    value
      transformw: Mid → Output,
      writer : Unit → in get out output Unit
    axiom
      writer() ≡ let v=get? in output!(transformw(v)) end; writer()
  end
```

```
scheme CONNECTOR =
  class
    type
      Mid
    channel
```

```
        add , get : Mid
    value
        connector : Unit → in add out get Unit
    axiom
        connector() ≡ let v=add? in get!(v) end; connector()
end
```

The connector is both the left unit for the WRITER component and the right unit for the READER component under the sequential composition operation. The composition of READER and WRITER via CONNECTOR forms another component READER_WRITER whose behavior is the sequential composition of its components.

```
scheme READER_WRITER =
  class
    type
        Input, Output
    channel
        input : Input,
        output : Output
    value
        behavior : Unit → in input out output Unit
    axiom
        behavior() ≡
            local
                type
                    Mid
                channel
                    add : Mid,
                    get : Mid
                value
                    transformr: Input → Mid,
                    transformw: Mid → Output,
                    connector : Unit → in add out get Unit,
                    reader : Unit → in input out add Unit,
                    writer : Unit → in get out output Unit
                axiom
                    connector() ≡ let v=add? in get!(v) end,
                    reader() ≡ let v=input? in add!(transformr(v)) end,
                    writer() ≡ let v=get? in output!(transformw(v)) end
                in
                    reader(); connector(); writer(); behavior()
            end
  end
```

4.4 Choice

We have known that the nondeterminism behavior (*internal choice*) can be described in the coalgebraic setting by a powerset behavior monad $\mathcal{P}$, which allows a system modeled by the coalgebra can behave like one of a set of alternative possibilities in the codomain set. Which alternative it behaves like does not depend on the what it communicates with the environment. Since the result for an input forms a set, the familiar algebraic properties being satisfied by nondeterministic choice can be easily got from the set operations, including *symmetric*, *associative*, *idempotent* properties and moreover, the *distribution* of all the operations (including those we will define later) through nondeterministic choice.

To give a universal interpretation of the *external choice* operation, which plays a role like the conditional of sequential programs [15], we eximine the coproduct operation in our category of coalgebras.

Definition 4.5 *Let* $c = (U_c, \alpha_c : U_c \to A_c \times \mathsf{B}_c(O_c \times U_c)^{I_c}, u_c^0)$ *and* $d = (U_d, \alpha_d : U_d \to A_d \times \mathsf{B}_d(O_d \times U_d)^{I_d}, u_d^0)$ *be two coalgebras where* $I_c \cap I_d = \emptyset$*. A coproduct (sum) of them* $c \boxplus d$ *is defined as the coalgebra* $e = (U, \alpha : U \to A \times \mathsf{B}(O \times U)^I, u^0)$*, where*

- $U = (U_c \times \{u_d^0\}) \cup (\{u_c^0\} \times U_d)$ *which can be identified with the union of the two state spaces* $U_c \cup U_d$*;*
- $A = A_c \cup A_d$*,* $I = I_c \cup I_d$*;*
- $\forall u \in U, o_e(u) = \pi_1(\alpha(u)) = o_c(\pi_1(u)) + o_d(\pi_2(u)) \in A_c \cup A_d$*;*
- *the action* $a_e : U \to \mathsf{B}(O \times U)^I$ *at the initial state* u_0 *depends on the input* I *and is defined as follows:*

$$\begin{array}{rcl} a_e : U \times I = U \times (I_c + I_d) & \xrightarrow{distribute} & U \times I_c + U \times I_d \\ & \xrightarrow{\times} & (U_c \times I_c) \times \{u_d^0\} + \{u_c^0\} \times (U_d \times I_d) \\ & \xrightarrow{a_c \times id + id \times a_d} & \mathsf{B}_c(O_c \times U_c) \times \{u_d^0\} + \{u_c^0\} \times \mathsf{B}_d(O_d \times U_d) \\ & \xrightarrow{\tau_r^{\mathsf{B}_c} + \tau_l^{\mathsf{B}_d}} & \mathsf{B}_c(O_c \times U_c \times \{u_d^0\}) + \mathsf{B}_d(\{u_c^0\} \times O_d \times U_d) \\ & \xrightarrow{\times} & \mathsf{B}_c(O_c \times U_c \times \{u_d^0\}) + \mathsf{B}_d(O_d \times \{u_c^0\} \times U_d) \end{array}$$

which alternative should be the result of the action depends on the input set, that means, the behavior of the sum coalgebra is decided by the input from its environment.

$$a_e : U \to \mathsf{B}_c(O_c \times U_c \times \{u_d^0\})^{I_c} + \mathsf{B}_d(O_d \times \{u_c^0\} \times U_d)^{I_d}$$

after the execution of the action a_e *on the initial state, the following behavior is same as either* a_c *or* a_d*.*

It can be easily proved that the operation $\boxplus$ defined in Definition 4.5 is the coproduct in the specified category of coalgebras. The observations on coalgebra $c \boxplus d$ consists of the observations of both coalgebras (but only the observation of one component can be changed in the coalgebra $c \boxplus d$, and that of the other one is fixed), and its actions are same as either c or d, depending on the input I. The unique morphism in the universal property of coproducts can be defined as $[p, q]$ for any other coalgebra e' and $p : c \to e'$, $q : d \to e'$ as coalgebra morphisms.

Moreover, a unit for the choice operation is the coalgebra with a singleton state space as follows:

$$\mathbf{1}_{\boxplus} = (\mathbf{1} = \{*\}, \langle id_{\mathbf{1}}, id_{\mathbf{1}} \rangle : \mathbf{1} \to \mathbf{1} \times \mathsf{B}(\mathbf{1})^{\emptyset})$$

Algebraic laws on the choice operation are given as follows:

Law 4.13 $c \boxplus \mathbf{1}_{\boxplus} \sim c$.

Proof: First, we can easily know that the state space on both sides are isomorphic. Since the input set of $\mathbf{1}_{\boxplus}$ is empty, the only input can be accepted of $c \boxplus \mathbf{1}_{\boxplus}$ is that of c. Therefore, the coalgebra constructed by the choice will be acting as c as a result. The equality on the observation parts up to isomorphism is obvious.

It is well known that $U \times \{v\}$ for any set U and singleton set $\{v\}$ can be identified with the set U naturally. Therefore, we have the following law:

Law 4.14 $c \boxplus c \sim c$.

Proof: By Definition 4.5.

The next laws can be easily proved by applying Definition 4.4 and 4.5 seperately.

Law 4.15 $c \boxplus d \sim d \boxplus c$.

Law 4.16 $(c \boxplus d) \boxplus e \sim c \boxplus (d \boxplus e)$ *if* $I_c \cap I_d = I_c \cap I_e = I_d \cap I_e = \{\}$.

Law 4.17 $(c \boxplus d); e \sim (c; e) \boxplus (d; e)$.

Law 4.18 $e; (c \boxplus d) \sim (e; c) \boxplus (e; d)$.

In Law 4.17 and 4.18, the conditions of sequential composition (interface compatibility) must be satisfied. That is, in Law 4.17 (4.18), the type of output (input) values of c and d must be same as the type of input (output) values of e so that the sequential compositions happening in this law are all valid.

Consider a concurrent database specification as an example using the operations defined in this paper.

```
scheme DATABASE =
  class
    type
      Key, Data,
      Database = Key →m Data
    channel
      empty : Unit,
      insert : Key × Data,
      remove, defined, lookup : Key,
      defined_res : Bool,
      lookup_res : Data
    value
      database : Database → in any out any Unit,
      not_found : Data
    axiom
      ∀ db: Database •
        database(db) ≡
          empty? ; database([ ])
          []
          let (k,d) = insert? in database(db†[ k↦d ]) end
          []
          let k = remove? in database(db\{k}) end
          []
          let k = defined? in defined_res!(k ∈ dom db) ; database(db) end
          []
          let k = lookup? in
            if k ∈ dom db then lookup_res!(db(k)) ; database(db)
            else lookup_res!not_found ; database(db) end
      end
  end
```

Here the value database is specified as a coalgebra which is a external choice [] among several channels (corresponding to the + construction defined in Section 4), which are described either as observations or as actions of the coalgebra (also can be seen as coalgebras). For example:

channel insert : Key $\times$ Data
value insert : Database $\times$ Key $\times$ Data $\rightarrow$ Database

corresponds to the channel and value declaration of the action *insert* seperately, which is also a coalgebra of the functor Ins : $X \rightarrow X^{\mathrm{Key}\times\mathrm{Data}}$.

4.5 Parallel Composition

Coalgebras has been used succesfully as a suitable framework for the description of state-based systems. One central operation for such systems is parallel composition, which sets the components (subsystems) of a system in communication with each other via actions of synchronisations or value-passing communications. For example, in CCS [26], processes communicate over channels which link the ports of the involved processes. Synchronisations are formed by combining inputs and outputs on the same channel. There may be other kinds of synchronisations. For example, the actions of two components may be executed simultaneously, but no relation on the inputs and outputs of them can be specified. Moreover, the actions of two parallel components may also be executed asynchronously.

Parallel composition can not be derived simply from a universal construction in the category of coalgebras because of the different possible behavior patterns (synchrony or asynchrony, etc.). We only present some experimental operations here for parallel composition between coalgebras.

The first operation we consider is a restricted synchronous product of coalgebras:

Definition 4.6 *Let* $c = (U_c, \alpha_c : U_c \to A_c \times \mathsf{B}_c(O_c \times U_c)^{I_c}, u_c^0)$ *and* $d = (U_d, \alpha_d : U_d \to A_d \times \mathsf{B}_d(O_d \times U_d)^{I_d}, u_d^0)$ *be two coalgebras, their synchronous product is defined as the coalgebra*

$$c \boxtimes d = (U, \alpha : U \to A \times \mathsf{B}(O \times U)^I)$$

where $U = U_c \times U_d$, $A = A_c \times A_d$, $I = I_c \times I_d$, *the action* $a : U \to \mathsf{B}(O \times U)^I$ *is defined as follows:*

$$\begin{aligned} a_{c \boxtimes d} : (U_c \times U_d) \times (I_c \times I_d) & \xrightarrow{\langle\langle \pi_1 \cdot \pi_1, \pi_1 \cdot \pi_2\rangle, \langle \pi_2 \cdot \pi_1, \pi_2 \cdot \pi_2\rangle\rangle} (U_c \times I_c) \times (U_d \times I_d) \\ & \xrightarrow{\langle a_c, a_d\rangle} \mathsf{B}_c(U_c \times O_c) \times \mathsf{B}_d(U_d \times O_d) \\ & \xrightarrow{\tau_r^{\mathsf{B}_c}} \mathsf{B}_c(U_c \times O_c \times \mathsf{B}_d(U_d \times O_d)) \\ & \xrightarrow{\mathsf{B}_c \tau_l^{\mathsf{B}_d}} \mathsf{B}_c(\mathsf{B}_d(U_c \times O_c \times U_d \times O_d)) \\ & \xrightarrow{\mathsf{B}_c \mathsf{B}_d \mathsf{m}} \mathsf{B}_c(\mathsf{B}_d(U \times O)) \end{aligned}$$

Such an operation is too restricted because it need the actions of both its components being triggered by a pair of input values and executed simultaneously. However, in real systems, not all behaviors of parallel composed components are executed exactly at the same time. Therefore, we give the following another more "free" interpretation for parallel composition:

Definition 4.7 *Let* $c = (U_c, \alpha_c : U_c \to A_c \times \mathsf{B}_c(O_c \times U_c)^{I_c}, u_c^0)$ *and* $d = (U_d, \alpha_d : U_d \to A_d \times \mathsf{B}_d(O_d \times U_d)^{I_d}, u_d^0)$ *be two coalgebras, their free product is defined as the coalgebra*

$$c \otimes d = (U, \alpha : U \to A \times \mathsf{B}(O \times U)^I)$$

where $U = U_c \times U_d$, $A = A_c \times A_d$, $I = I_c + I_d + I_c \times I_d$, *the action* $a : U \to \mathsf{B}(O \times U)^I$ *is defined as follows:*

$$\begin{array}{rcl} a : (U_c \times U_d) \times (I_c + I_d + I_c \times I_d) & \xrightarrow{distribute} & (U_c \times U_d) \times (I_c + I_d) + (U_c \times U_d) \times (I_c \times I_d) \\ & \xrightarrow{[a_{c \boxplus d}, a_{c \boxtimes d}]} & \mathsf{B}_c(O_c \times U) + \mathsf{B}_d(O_d \times U) + \mathsf{B}_c\mathsf{B}_d(O_c \times O_d \times U) \end{array}$$

Generally, parallel composition of coalgebras can be derived by applying restriction, view, and relabelling operations previously defined to the free product of them. Moreover, a synchronous product is a special case of the result by applying such operations in which all the possible operations are synchronisations of the operations of coalgebras as components.

A behavior unit for $\boxtimes$ is given by the following law.

Law 4.19 $c \boxtimes \mathbf{1}_{\boxtimes} \sim c$, *for the unit* $\mathbf{1}_{\boxtimes} = (\mathbf{1}, \langle id_{\mathbf{1}}, \epsilon_{\mathbf{1}} \rangle : \mathbf{1} \to \mathbf{1} \times (\mathbf{1})^S)$ *for any set* S *and any coalgebra* $c = (U, \alpha : U \to A \times \mathsf{B}_c(O \times U)^I)$.

Proof: By Definition 4.6, the equality of observations can be got directly, and because the unit $\mathbf{1}_{\boxtimes}$ has no action, the action $a_{c \boxtimes \mathbf{1}_{\boxtimes}}$ is

$$a_{c \boxtimes \mathbf{1}_{\boxtimes}} : (U \times I) \times (\mathbf{1} \times S)(\cong U \times I) \to \mathsf{B}_c(O \times U) \times \mathbf{1}(\cong \mathsf{B}_c(O \times U))$$

Then the result can be got directly.

The following laws can be easily proved by applying Definition 4.5 and 4.6.

Law 4.20 $c \boxtimes d \sim d \boxtimes c$.

Law 4.21 $(c \boxtimes d) \boxtimes e \sim c \boxtimes (d \boxtimes e)$.

Law 4.22 $(c \boxplus d) \boxtimes e \sim (c \boxtimes e) \boxplus (d \boxtimes e)$.

Law 4.23 $c \otimes \mathbf{1}_{\otimes} \sim c$, *for* $\mathbf{1}_{\otimes} = \mathbf{1}_{\boxplus}$ *as the unit of* $\otimes$ *and any coalgebra* c.

Law 4.24 $c \otimes d \sim d \otimes c$.

Law 4.25 $(c \otimes d) \otimes e \sim c \otimes (d \otimes e)$.

Law 4.26 $(c \boxplus d) \otimes e \sim (c \otimes e) \boxplus (d \otimes e)$.

The following is an example of a component READER_WRITER2 which has *input* and *output* as two interfaces and describes a paralleled composition of three processes (can be considered as coalgebras, see [3, 43] for more details): *reader*, *connector* and *writer*.

```
scheme READER_WRITER2 =
  class
    type
      Input, Output
    channel
      input : Input,
      output : Output
    value
      behavior : Unit → in input out output Unit
    axiom
      behavior() ≡
        local
          type
            Mid
          channel
            add : Mid,
            get : Mid
          value
            transformr : Input → Mid,
            transformw : Mid → Output,
            connector : Unit → in add out get Unit,
            reader : Unit → in input out add Unit,
            writer : Unit → in get out output Unit
          axiom
            connector() ≡ let v = add? in get!v end; connector(),
            reader() ≡ let v = input? in add!(transformr(v)) end; reader(),
            writer() ≡ let v = get? in output!(transformw(v)) end; writer()
          in
            reader()||connector()||writer()
        end
  end
```

The behavior of this component is similar as that of the component READER_WRITER. However, here the parallel composition $||$ corresponds to the free product $\otimes$ defined in this section and the three processes can happen in any possible sequence while the three components of READER_WRITER must behave in a specified sequence as the specification shows.

5 Conclusion and Future Work

In this paper we present a category $\mathbf{Coalg}_{KPF}$ for a specified class of KPFs as a generalization for the category $\mathbf{Coalg}_F$ being widely exploited in the research of coalgebra theory. Based on this category, a set of constructions are introduced to describe the interaction and coordination between components in a system. The properties of such constructions are described via algebraic laws.

The coalgebraic calculational model provides a clear seperation between computation and coordination as two orthogonal aspects of a system and describes the coordination between components in a highly abstract way. Our work in this area is still on-going. Future work includes the extension of timed properties to the model and the RSL representation for the calculus. Another plan is to find some instances of coordination between components and apply this calculus to give their specification and verification as the complementary work to [37].

Acknowledgement. The conversations with Dr. Luís Barbosa during his visit to UNU/IIST have greatly increased our understanding of the calculational approach in the research of coalgebra theory and motivated this work. This paper is partially supported by the National Natural Science Foundation of China under Grant No. 60273001.

References

[1] A. Arnold. *Finite Transition Systems*. Prentice Hall, 1994.

[2] L. S. Barbosa. Components as processes: An exercise in coalgebraic modeling. In Scott F. Smith and Carolyn L. Talcott, editor, *Formal Methods for Open Object-Based Distributed Systems IV,IFIF TC6/WG6.1 Fourth International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS 2000), September 6-8, 2000, Stanford, California, USA*, volume 177 of *IFIP Conference Proceedings*. Kluwer, 2000.

[3] L. S. Barbosa. *Components as Coalgebras*. PhD thesis, Universidade do Minho, Braga, Portugal, 2001.

[4] L. S. Barbosa. Coalgebraic structures in program construction. In *6th Brazilian Symposium on Programming Languages, SBLP'2002*, 2002.

[5] L. S. Barbosa and J. N. F. de Oliveira. Coinductive Interpreters for Process Calculi. In *Sixth International Symposium on Functional and Logic Programming*, volume 2441 of *LNCS*. Springer, 2002.

[6] M. Barr and C. Wells. *Toposes, Triples and Theories*. Springer, 1985.

[7] M. Barr and C. Wells. *Category Theory for Computing Science, Third Edition*. Les Publications CRM, 1999.

[8] F. Borceux. *Handbook of Categorical Algebra*. Cambridge University Press, 1994.

[9] C. Cîrstea. An algebra-coalgebra framework for system specification. In H. Reichel, editor, *Electronic Notes in Theoretical Computer Science*, volume 33. Elsevier Science Publishers, 2000.

[10] C. Cîrstea. *Integrating Observations and Computations in the Specification of State-Based Dynamical Systems*. PhD thesis, Corpus Christi College and St. John's College, University of Oxford, 2000.

[11] C. Cîrstea. A coalgebraic equational approach to specifying observational structures. *Theoretical Computer Science*, 280:35–68, 2002.

[12] R. Cockett and D. Spencer. Strong categorical datatypes I. In R. A. G. Seely, editor, *International Meeting on Category Theory 1991*, Canadian Mathematical Society Proceedings. AMS, 1992.

[13] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.

[14] C. A. R. Hoare and J. He. *Unifying Theories of Programming*. Prentice Hall International, 1998.

[15] B. Jacobs. Inheritance and cofree constructions. In P. Cointe, editor, *European Conference on Object-Oriented Programming*, volume 1098 of *LNCS*, pages 210–231. Springer, Berlin, 1996.

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

[17] B. Jacobs. Coalgebraic reasoning about classes in object-oriented languages. In *Electronic Notes in Theoretical Computer Science*, volume 11. Elsevier Science Publishers, 1998.

[18] B. Jacobs. *Categorical Logic and Type Theory*, volume 141 of *Studies in Logic and The Foundations of Mathematics*. Elsevier, 1999.

[19] B. Jacobs. Object-oriented hybrid systems of coalgebras plus monoid actions. *Theoretical Computer Science*, 239:41–95, 2000.

[20] B. Jacobs. Exercises in coalgebraic specification. In R. Backhouse, R. Crole, and J. Gibbons, editors, *Algebraic and Coalgebraic Methods in the Mathematics of Program Construction*, volume 2297 of *LNCS*, pages 237–280. Springer, 2002.

[21] B. Jacobs and J. Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.

[22] B. Jacobs and H. Tews. Assertional and Behavioural Refinement in Coalgebraic Specification. In *Electronic Notes in Theoretical Computer Science*, volume 47. Elsevier Science Publishers, 2001.

[23] A. Kurz. *Logics for Coalgebras and Applications to Computer Science*. PhD thesis, Universität München, 2000.

[24] A. Kurz. Specifying coalgebras with modal logic. *Theoretical Computer Science*, 260:119–138, 2001.

[25] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[26] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.

[27] E. Moggi. Computational lambda-calculus and monads. In *Proceedings 4th Annual IEEE Symp. on Logic in Computer Science, LICS'89, Pacific Grove, CA, USA, 5–8 June 1989*, pages 14–23. IEEE Computer Society Press, Washington, DC, 1989.

[28] E. Moggi. Notions of computation and monads. *Information and Computation*, 93:55–92, 1991.

[29] A. Pardo. A calculational approach to strong datatypes. Research Report 240, Department of Informatics, University of Oslo, 1996. In M. Haveraaen and O. Owe eds. Selected Papers 8th Nordic Workshop on Programming Theory, NWPT'96, Oslo, Norway.

[30] G. D. Plotkin. A Structural Approach to Operational Semantics. Technical Report DAIMI FN-19, Computer Science Department, Aarhus University, 1981.

[31] S. Popkorn. *First Steps in Modal Logic*. Cambridge University Press, 1994.

[32] H. Reichel. An approach to object semantics based on terminal co-algebras. *Mathematical Structures in Computer Science*, 5:129–152, 1995.

[33] J. Rothe, H. Tews, and B. Jacobs. The coalgebraic class specification language CCSL. *Journal of Universal Computer Science*, 7(2):175–193, 2001.

[34] J. Rutten. A calculus of transition systems (towards universal coalgebra). In A. Ponse, M. de Rijke, and Y. Venema, editors, *Modal Logic and Process Algebra, a bisimulation perspective*, pages 231–256. CSLI Lecture Notes No. 53, CSLI Publications, Standford, 1995.

[35] J. Rutten. Universal coalgebra: a theory of systems. *Theoretical Computer Science*, 249:3–80, 2000.

[36] M. Sun and B. Aichernig. Component-based Coalgebraic Specification and Verification in RSL. Technical Report 267, UNU/IIST, October 2002.

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

[38] The RAISE Language Group. *The RAISE Specification Language*. Prentice Hall International, 1992.

[39] D. Turi. *Functorial Operational Semantics and its Denatational Dual*. PhD thesis, Free University, Amsterdam, 1996.

[40] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *Proceedings 12th Ann. IEEE Symp. on Logic in Computer Science, LICS'97, Warsaw, Poland, 29 June – 2 July 1997*, pages 280–291. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[41] G. Winskel and M. Nielsen. Categories in concurrency. In A. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, pages 299–354. Publications of the Newton Institute, Cambridge University Press, 1997.

[42] U. Wolter. CSP, partial automata, and coalgebras. *Theoretical Computer Science*, 280:3–34, 2002.