

CONSTRUCTING SPECIFICATIONS AND MODULES IN A KZ-DOCTRINE

GILLIAN HILL

*Department of Computer Science, City University
Northampton Square, London, EC1 VOH B*

E-mail: gah@cs.city.ac.uk

and

*Department of Computing, Imperial College
180 Queen's Gate, London SW7 2BZ, United Kingdom*

E-mail: gah@doc.ic.ac.uk

ABSTRACT

Using the structure of a KZ-monad we create a general categorical workspace in which diagrams can be formally constructed. In particular this abstract framework of category theory is shown to provide a precise semantics for constructing the specifications of complex systems from their component parts.

1 Introduction

The purely equational 2-dimensional categorical algebra underlying the process of freely completing categories under a given suitable class of colimits is presented by Kock in [8]. A common feature of the category which freely completes another category is that it has diagrams (functors) as its objects. Using properties of adjointness Kock identifies a particular monad which is called either a ‘monad with the Kock property’ or a ‘KZ-doctrine’. Kock shows that this type of monad, which has structures which are adjoint to units, expresses precisely the properties of free cocompleteness and colimit formation in a category.

In this paper the properties of a KZ-doctrine are presented before they are shown to provide a general categorical workspace for the construction of diagrams. These ideas are then extended to provide a specific categorical workspace within which specifications of system components can be both structured and implemented to configure the architecture of a final executable system. We show that the mathematical structure of a KZ-doctrine, although complex, provides a powerful framework that expresses precisely the configuration of systems. We have already introduced a categorical framework for the configuration of

complex systems in [4], and designed a language for configuration in [6]. The possible relationships between the component parts of systems are identified at an intuitive level in [5] in order to choose appropriate high-level combinators for configuration. The high-level combinators are then formally defined in terms of more primitive combinators that operate on recursively defined configured objects of sorts in the set $\{specification, module\}$. Each configured object is a theory presentation within a logical framework for systems configuration that has been presented in [7]. Specifications of configured objects are represented in the categorical workspace by recursively defined and structured diagrams, whereas modules are represented by the singleton diagrams created by colimit formation from their specifications. Our aim is to present the category-theoretic framework for the configuration of modular systems as a mathematical workspace that is independent of any particular specification approach, design methodology or programming paradigm.

This paper is arranged as follows. In Section 2 we present the background to Kock’s work and in Section 3 we build on Kock’s definition of the KZ-doctrine to provide a formal framework for the construction of diagrams. In Section 4 these ideas are applied to a particular categorical workspace for the configuration of complex systems from specifications and modules, and an example is given of the configuration of an implemented module from a more abstract specification. Finally conclusions are drawn in Section 5.

2 Background

In this section we explain the concepts and notation from 2-dimensional category theory before defining the property of adjointness and the structure of a monad. By working within a 2-dimensional category we gain greater expressive power based on the horizontal and vertical composition of natural transformations and functors.

2.1 A 2-dimensional Category

We use [9, 17] to present the background to Kock’s work which is based on the structure of a 2-dimensional category. Since constructions are made on categories, functors, as the structure-preserving ‘homo-morphisms’ between categories, are defined within the category of all categories. Functors are arrows (morphisms) which model the objects and arrows of one category in another category. The modelling is made by functions on the objects and arrows of the first category in such a way that a single entity of these objects, and the arrows between them, is constructed in the second category.

From considering functors as arrows between any two categories, we can

define a category of functors with arrows called *natural transformations*. The naturality condition for the family of arrows that defines the transformation is expressed by a commutative diagram in the second (or target) category.

Categories and functors themselves form a category **Cat**, which is a *large* category with objects that are *small* categories. Within **Cat** both the vertical and the horizontal composition of two natural transformations can be defined with a rule that relates these compositions by an identity. This is the basis of 2-dimensional category theory. It is important to note that the objects for vertical composition are the functors, whereas for horizontal composition the objects are the categories.

Notation 2.1.1 In [8] Kock uses the following standard notation from 2-dimensional category theory:

1. 0-cell denotes an object, 1-cell denotes a morphism, and 2-cell denotes a natural transformation.
2. The vertical composition of 2-cells is denoted by \cdot , while the horizontal composition of 1-cells and 2-cells is denoted by $*$.

2.2 Adjointness as a Universal Property

The property of adjointness is a powerful way of expressing the properties of free objects and of universal constructions in category theory. Adjointness is defined between a pair of functors and always produces a universal arrow. The actual structure formed by the left and right adjoint functors and the categories at the end of the functors is called an *adjunction* and can be characterised, as in [17], by a pair of functors and a pair of natural transformations that satisfy the following identity transformations:

Proposition 2.2.1 Suppose $U : \mathcal{B} \rightarrow \mathcal{A}$, $F : \mathcal{A} \rightarrow \mathcal{B}$ are functors and $\eta : 1_{\mathcal{A}} \Rightarrow UF$, $\epsilon : FU \Rightarrow 1_{\mathcal{B}}$ are natural transformations satisfying

$$\begin{aligned} U_{\epsilon B} \cdot \eta_{UB} &= 1_{UB} & B \in \mathcal{B} \\ \epsilon_{FA} \cdot F\eta_A &= 1_{FA} & A \in \mathcal{A} \end{aligned}$$

Then F is *left adjoint* to U with *unit* η and *counit* ϵ , and the adjunction is denoted by $\langle F, U, \eta, \epsilon \rangle : \mathcal{A} \dashv \mathcal{B}$. □

Notation 2.2.2

1. Kock denotes the adjointness between 1-cells by $f \dashv g$ meaning that f is left adjoint to g .
2. Identity 2-cells may be denoted by *id*, so that the context must make it clear what 1-cell it is the identity 2-cell of.

3. The 2-cell front-adjunction is denoted by $\eta : id \Rightarrow g * f$ and the 2-cell back-adjunction is denoted by $\epsilon : f * g \Rightarrow id$.
4. When ϵ is an identity 2-cell the fact that f is then a reflection (left) adjoint to g is denoted by $f \dashv_r g$.
5. When η is an identity 2-cell the fact that g is then a coreflection (right) adjoint of f is denoted by $f \dashv_{co} g$.

2.3 Monads and Algebras

Adjointness, as a universal property, can be related to the theory of universal algebra by a structure called a monad. In fact every monad, which is like a monoid but with the extra structure of natural transformations between functors, can be defined by a suitable pair of adjoint functors.

As suggested initially, however, an alternative way of introducing the structure of a monad is to consider a monad as a bridge between adjointness and universal algebras. First we show how a monad can be defined in general from an adjunction. Then a link is made from a monad to a particular category of T -algebras. Finally the bridge can be recrossed to define an adjunction from the category of T -algebras from which a monad can be derived.

Definition 2.3.1 (A Monad defined by an Adjunction) Let $\langle F, U, \eta, \epsilon \rangle : \mathcal{A} \rightarrow \mathcal{B}$ denote an adjunction. Then the monad defined by $\langle F, U, \eta, \epsilon \rangle$ is the monad in \mathcal{A} defined by $\langle UF, \eta, U\epsilon F \rangle$. The functors $F : \mathcal{A} \rightarrow \mathcal{B}$ and $U : \mathcal{B} \rightarrow \mathcal{A}$ have the endofunctor $T = UF$ as their composite. The unit of the adjunction is the unit of the monad $\eta : I \Rightarrow T$. The counit $\epsilon : FU \Rightarrow I_{\mathcal{B}}$ of the adjunction with horizontal composition gives the multiplication of the monad $\mu = U\epsilon F : UFUF \Rightarrow UF = T$. \square

Information about the adjunction $F \dashv U$ is retained in \mathcal{A} in the form of the endofunctor T with natural transformations η and μ .

From the monad $\mathbf{T} = \langle T, \eta, \mu \rangle$, in \mathcal{A} , a category $\mathcal{A}^{\mathbf{T}}$ of \mathbf{T} -algebras, called the *Eilenberg-Moore category of the monad \mathbf{T}* , can be constructed. This link from the monad, itself determined by the structure of free monoids, to its algebras (reconstructed as arbitrary monoids) illustrates the importance of the information held in the free algebras.

Definition 2.3.2 (A \mathbf{T} -Algebra) Let $\mathbf{T} = \langle T, \eta, \mu \rangle$ denote a monad in \mathcal{A} . Then a \mathbf{T} -algebra $\langle A, h \rangle$ is a pair consisting of an object $A \in \mathcal{A}$ (the underlying object of the algebra) and an arrow $h : TA \rightarrow A$ of \mathcal{A} (called the structure map of the algebra) which makes both the following diagrams commute: \square

$$\begin{array}{ccc}
T^2 A & \xrightarrow{Th} & TA \\
\mu_A \downarrow & & \downarrow h \\
TA & \xrightarrow{h} & A
\end{array}
\qquad
\begin{array}{ccc}
A & \xrightarrow{\eta_x} & TA \\
\searrow 1 & & \downarrow h \\
& & A
\end{array}$$

From the category $\mathcal{A}^{\mathbf{T}}$ the adjunction $\mathcal{A} \dashv \mathcal{A}^{\mathbf{T}}$ can then be defined, and from the adjunction the monad \mathbf{T} in \mathcal{A} can finally be reconstructed.

3 Constructing Diagrams Within a KZ-doctrine

We explain the particular characteristics of a monad with the Kock property before giving Kock's definition of a KZ-doctrine. We then show how diagrams can be constructed and 'flattened' within a formal calculus.

3.1 Monads with the Kock Property

Particular monads that are defined on a 2-category and possess the Kock property are presented by Kock as *KZ-doctrines* in [8]. The key feature of these monads is that they possess extra structure due to adjointness. For example, the structure map of the \mathbf{T} -algebra, for the KZ-doctrine \mathbf{T} , is a left adjoint to the unit of the monad. We present only those ideas from [8] that we need for the formal construction of diagrams and use our own notation in places.

The feature that is attractive for our own work of constructing diagrams in a categorical workspace is that the category $F(\mathcal{C})$ which freely completes a category \mathcal{C} has diagrams in \mathcal{C} as its objects. In fact $F(\mathcal{C})$ freely *co*-completes \mathcal{C} which has, among its objects, the appropriate colimits for the diagrams in $F(\mathcal{C})$. It follows that, if \mathcal{C} is cocomplete, there is a functor $\lim_{\rightarrow} : F(\mathcal{C}) \rightarrow \mathcal{C}$ which associates a colimit $\lim_{\rightarrow}(D)$ to a diagram D in $F(\mathcal{C})$. Uniqueness of colimits is expressed by making \lim_{\rightarrow} left adjoint to the functor $y_{\mathcal{C}} : \mathcal{C} \rightarrow F(\mathcal{C})$ that assigns a singleton diagram to an object $C \in \mathcal{C}$.

As we have seen the universal property of freeness can also be expressed by the structure of a monad. To express the freeness of $F(\mathcal{C})$, Kock makes F left adjoint to the forgetful functor $U : \mathbf{SCat} \rightarrow \mathbf{Cat}$, where \mathbf{SCat} is the category of categories which admits the appropriate kind of colimits and whose functors preserve them. The composite endofunctor $T = U \circ F : \mathbf{Cat} \rightarrow \mathbf{Cat}$ and the units, $y_{\mathcal{C}}$ where $\mathcal{C} \in \mathbf{Cat}$, form the monad \mathbf{T} . The link from the monad \mathbf{T} over \mathbf{Cat} to the category of \mathbf{T} -algebras $\mathbf{Cat}^{\mathbf{T}}$ provides the equivalence between $\mathbf{Cat}^{\mathbf{T}}$ and \mathbf{SCat} . It also provides the key feature that the structure map for $\mathbf{Cat}^{\mathbf{T}}$ is a left adjoint for the unit $y_{\mathcal{C}} : \mathcal{C} \rightarrow T(\mathcal{C})$.

Kock's central result is that all the properties of the KZ-doctrine and its algebras can be derived from a single natural transformation called λ . He presents a formal calculus where T is any endo-2-functor on any 2-category (that we denote by \mathcal{A}), The particular case we are interested in is where T is a functor on an object \mathcal{C} which is a category in the 2-category \mathbf{Cat} .

Definition 3.1.1 (A KZ-Doctrine) Let \mathcal{A} be a 2-category and let $I : \mathcal{A} \rightarrow \mathcal{A}$ denote the identity functor. A *KZ-doctrine* on \mathcal{A} consists of an endo-2-functor $T : \mathcal{A} \rightarrow \mathcal{A}$, and two 2-natural transformations $y : I \rightarrow T$, $m : T \circ T \rightarrow T$, and, for each $A \in \mathcal{A}$, a 2-cell $\lambda_A : T(y_A) \Rightarrow y_{T(A)}$,

$$T(A) \begin{array}{c} \xrightarrow{T(y_A)} \\ \Downarrow \lambda_A \\ \xrightarrow{y_{T(A)}} \end{array} T^2(A)$$

satisfying the following axioms:

- **T0** y is a strict two-sided unit for m (their composition is the identity),
 $m_A * T(y_A) = m_A * y_{T(A)} = id_{T(A)}$
- **T1** $\lambda_A * y_A$ is an identity 2-cell
- **T2** $m_A * \lambda_A$ is an identity 2-cell
- **T3** $m_A * T(m_A) * \lambda_{TA}$ is an identity 2-cell.

□

The construction of the natural transformation λ depends on another adjoint situation between the natural transformations y and m . Because y_A is an inclusion functor (it assigns a diagram to each object in \mathcal{A}) its left adjoint is a *reflection*.

Proposition 3.1.2 We have the reflection adjointness $m_A \dashv_r y_{TA}$, where $A \in \mathcal{A}$, with $T(m_A) * \lambda_{TA}$ as front adjunction. □

Now we recall the process of freely cocompleting any category \mathcal{C} in the particular category \mathbf{Cat} . The functor $F : \mathbf{Cat} \rightarrow \mathbf{SCat}$ is left adjoint to the forgetful adjoint functor $U : \mathbf{SCat} \rightarrow \mathbf{Cat}$. The unit of the adjunction, that assigns to an object $C \in \mathcal{C}$ its singleton diagram, is $y_C : \mathcal{C} \rightarrow UFC = TC$. For a category $\mathcal{C} \in \mathbf{Cat}$ to be in \mathbf{SCat} , the subcategory of \mathbf{Cat} , it must have the appropriate kind of colimits. The colimit formation, ξ , is carried out by the component, ξ_C , of a natural transformation ξ which is left adjoint to y_C . In particular Kock shows that not only does any freely cocompleted category, $TC \in \mathbf{Cat}$, admit a reflection left adjoint $m_C : T^2\mathcal{C} \rightarrow TC$ for y_{TC} but also that m_C is a coreflection right adjoint for $T(y_C)$. These adjunctions will be important when we construct specifications as diagrams. We shall use m_C ,

which is an arrow in \mathbf{SCat} , to form the colimits of diagrams over objects in $T\mathcal{C}$. The cocompletion process $T = UF$ which we use to build the new specification diagrams is proved formally by Kock to have the structure of a KZ-doctrine.

However, a KZ-doctrine $\mathbf{T} = (T, y, m, \lambda)$ on a 2-category \mathcal{A} is shown by Sassone et al in [13] to be a weaker structure than a monad. We have already seen that the monad, as an algebraic structure, acts as a bridge to universal algebras. Because KZ-doctrines are defining the completion of categories by colimits and the notion of colimit is defined only up to isomorphism, the stronger form of universality that gives rise to a monad is not possible. The free object for cocompleteness is identified up to equivalence, and not up to isomorphism as is usual for free constructions. The simplicity of expressing this weaker structure as a KZ-doctrine is that the information about the structure is contained in the natural transformation λ .

The unit of the KZ-doctrine \mathbf{T} is y and the multiplication is m . Taking \mathbf{Cat} as the particular underlying category, Kock identifies an equivalence between the Eilenberg-Moore category of \mathbf{T} -algebras and \mathbf{SCat} . A further adjoint situation can be expressed in general between the structure map of the algebra for \mathbf{T} and the unit of the monad.

Proposition 3.1.3 Let $\mathbf{T} = (T, y, m, \lambda)$ be a KZ-doctrine on the 2-category \mathcal{A} . The structure map $h : T(A) \rightarrow A$, where A is an object of \mathcal{A} , is a retraction (left inverse) left adjoint for $y_A : A \rightarrow T(A)$. Both the structure map h and the unit η for $h \dashv_r y_A$ are unique up to isomorphism. □

By Proposition 3.1.2 we have for $A \in \mathcal{A}$ the adjointness relation $m_A \dashv_r y_{TA}$, and so it follows that TA has the structure of a \mathbf{T} -algebra with structure map m_A . Kock summarises the adjointness within a KZ-doctrine by the following theorem.

Theorem 3.1.4 For any KZ-doctrine $\mathbf{T} = (T, y, m, \lambda)$ on a 2-category \mathcal{A} we have, for any $A \in \mathcal{A}$, $T(y_A) \dashv_{co} m_A \dashv_r y_{TA}$. □

3.2 Constructing Diagrams

Now we apply the abstract properties of a KZ-doctrine $\mathbf{T} = (T, y, m, \lambda)$ on the particular 2-category \mathbf{Cat} to the general task of formally constructing diagrams. We begin with diagrams over objects in a category $\mathcal{C} \in \mathbf{Cat}$ and define these diagrams as objects in the freely cocompleted category $T\mathcal{C}$ of diagrams over \mathcal{C} . The construction of more complex diagrams can be completed in the category $T^2\mathcal{C}$ whose objects are the diagrams in $T\mathcal{C}$. We define the morphisms between the diagrams over \mathcal{C} as the morphisms in $T\mathcal{C}$; the morphisms between diagrams in $T\mathcal{C}$ are then defined as morphisms in $T^2\mathcal{C}$.

Because a KZ-doctrine is a weaker structure than a monad it may be necessary, as Meseguer and Sassone have shown in [11, 13], to identify certain diagrams by defining an equivalence relation over the diagrams as functors. Their technique is to construct, by the Yoneda embedding, the category $\mathbf{Set}^{\mathcal{C}^{op}}$ as the free completion of \mathcal{C} by arbitrary colimits.

Definition 3.2.1 (Morphisms in $T\mathcal{C}$) Let $T\mathcal{C}$ be the free completion of the category $\mathcal{C} \in \mathbf{Cat}$. The objects of $T\mathcal{C}$ are the diagrams (functors) $D : \mathbf{I} \rightarrow \mathcal{C}$. Let $D_1 : \mathbf{I}_1 \rightarrow \mathcal{C}$ and $D_2 : \mathbf{I}_2 \rightarrow \mathcal{C}$ be objects in $T\mathcal{C}$ and $\varphi : \mathbf{I}_1 \rightarrow \mathbf{I}_2$ be a functor between the index categories \mathbf{I}_1 and \mathbf{I}_2 . Then a morphism between objects in $T\mathcal{C}$ is the natural transformation between the functors D_1 and D_2 in \mathcal{C} denoted by $f_{\mathcal{C}} : D_1 \Rightarrow D_2 * \varphi$. \square

The objects in $T^2\mathcal{C}$ are diagrams over diagrams in \mathcal{C} with morphisms defined between them.

Definition 3.2.2 (Morphisms in $T^2\mathcal{C}$) Let $D_1 : \mathbf{I}_1 \rightarrow T\mathcal{C}$ and $D_2 : \mathbf{I}_1 \rightarrow T\mathcal{C}$ be diagrams in $T\mathcal{C}$. Then D_1 and D_2 are objects in $T^2\mathcal{C}$. Let $\varphi : \mathbf{I}_1 \rightarrow \mathbf{I}_2$ be a functor. Then a morphism between objects in $T^2\mathcal{C}$ is the natural transformation $\alpha : D_1 \Rightarrow D_2 * \varphi$. \square

Now we consider what the ‘diagrams of diagrams’ in $T^2\mathcal{C}$ look like. All the components of each diagram in $T\mathcal{C}$ with the arrows between them are preserved in the diagrams which are the objects in $T^2\mathcal{C}$. In turn all the components of the diagrams in \mathcal{C} with arrows between them are preserved in the diagrams which are objects in $T\mathcal{C}$. Those diagrams which have been structured from diagrams in \mathcal{C} will include the encapsulation around the objects in \mathcal{C} which identifies their membership of the diagram in \mathcal{C} . Also included in $T\mathcal{C}$ are the singleton diagrams assigned by $y_{\mathcal{C}}$ to each of the objects in \mathcal{C} . Similarly in $T^2\mathcal{C}$ are the singleton diagrams assigned by $y_{T\mathcal{C}}$ to each of the objects in $T\mathcal{C}$; these will have two shells around them to identify them as ‘singleton diagrams of singleton diagrams’.

Using Definition 3.1.1 we illustrate in Figure 1 the construction of objects in $T^2\mathcal{C}$ by both $Ty_{\mathcal{C}}$ and $y_{T\mathcal{C}}$ with λ as the 2-cell between these functors. The effect of λ is to form a ‘flat’ diagram from a diagram structured by singleton diagrams in $T^2\mathcal{C}$. For the construction by $Ty_{\mathcal{C}}$, each of the objects that belong to a diagram in \mathcal{C} are first assigned by $y_{\mathcal{C}}$ to their singleton diagrams in $T\mathcal{C}$. Under the endofunctor T on $T\mathcal{C}$, a diagram is then constructed, as an object in $T^2\mathcal{C}$, from these singleton diagrams. We denote the construction of the singleton diagrams that are part of the diagrams in $T^2\mathcal{C}$ by square brackets. For the construction by $y_{T\mathcal{C}}$, the diagram in \mathcal{C} first becomes an object in $T\mathcal{C}$. Composition with the component $y_{T\mathcal{C}}$ then constructs a singleton diagram in $T^2\mathcal{C}$ as an object from the diagram in $T\mathcal{C}$. The flattening effect of λ on the construction by $y_{T\mathcal{C}}$ results from the colimit construction with the flattened diagram as the colimit of the ‘diagram of diagrams’.

$$\begin{array}{ccc}
D : \mathbf{I} \rightarrow \mathcal{C} & & T^2\mathcal{C} \\
& & [A] \\
& \xrightarrow{T y_{\mathcal{C}}} & \downarrow y_{\mathcal{C}}(f) \\
& & [B] \\
& \downarrow \lambda & \\
& & A \\
& \xrightarrow{y_{T\mathcal{C}}} & \downarrow f \\
& & B
\end{array}$$

Figure 1: Constructing diagrams in $T^2\mathcal{C}$

The adjointness between structures and units for a KZ-doctrine allows flexibility in moving between constructing diagrams and forming their colimits in \mathcal{C} , $T\mathcal{C}$ and $T^2\mathcal{C}$. For example, colimit formation in $T\mathcal{C}$ of a diagram in $T^2\mathcal{C}$ can be expressed by $m_{\mathcal{C}}$ using the adjointness $m_{\mathcal{C}} \dashv y_{T\mathcal{C}}$. The 2-cell λ is shown by Kock to be derived from this adjoint situation. The singleton diagram of a structured diagram flattened by $m_{\mathcal{C}}$ is constructed by $y_{T\mathcal{C}}$ in the category $T^2\mathcal{C}$. It is the unit of the adjunction $\eta_{\mathcal{C}} : I \rightarrow y_{T\mathcal{C}} * m_{\mathcal{C}}$ that, as a morphism in $T^2\mathcal{C}$, assigns each structured diagram of diagrams to its singleton diagram of trivial shape.

4 The Categorical Workspace for Configuration

We now extend these general ideas on the formal construction of diagrams to provide a specific categorical workspace within which the specifications of complex systems can be configured from their component parts.

The activity of specification is in our view that of building theories, that are presented in specifications, so we extend the ideas of Turski, Maibaum and Veloso in [14, 16]. Interpretation between theories has been formalized in a categorical framework by Maibaum and Fiadeiro in [10], and category theory was also used to define an abstract specification theory for refining specifications by Ury and Gergely in [15]. Earlier work by Burstall, Goguen and Sannella in [1, 2, 12] used the construction of colimits for diagrams during theory-building. By using based theory morphisms, they extended theories by including the

diagram of the extension in the extended theory. In [3] Goguen has presented an axiomatisation of the notion of extension as ‘inclusion’ between theories.

We have broadened the constructions on specifications, as named theories, to be based on interpretation between theories with the extension of theories as a special case of interpretation which does not involve a change of language. An extension between theories that is conservative is shown in [4, 6, 5] to preserve the properties of the extended theory without allowing the deduction of new properties about the extended theory in the extension. Morphisms that represent conservative extension play an important role within the mathematical workspace that we provide for software engineers to use.

In our workspace for configuration, system components are represented by recursively defined objects of sorts in the set $\{specification, module\}$. Specifications retain the history of their configuration whereas modules are simply the reusable instances of specifications. The abstract category-theoretic semantics of configuration is based on morphisms between theories in a logical category. A requirement for the logic underlying the workspace for configuration is that it should possess the Craig interpolation property. This property ensures the preservation of structure during configuration. We identify the primitive combinators for configuring objects as extension, conservative extension and interpretation; they operate on both specifications and modules and are represented by morphisms between recursively defined diagrams. The horizontal and vertical high-level combinators, defined precisely in terms of the primitive combinators, construct new specifications as the diagrams of more complex objects by completing the colimit diagrams from the component parts. The combinator for creating modules (in the third dimension of our workspace) constructs the colimit object of a specification and only keeps that as a singleton diagram.

4.1 A Categorical Workspace

Now we apply Kock’s work on KZ-doctrines to the specific task of constructing a powerful and expressive framework for both structuring and implementing specifications and modules to configure complex systems. We define a configuration workspace, based on the structure of a particular KZ-doctrine for configuration, and illustrate how the specifications and modules of system components are configured by morphisms that represent the primitive combinators of the configuration language.

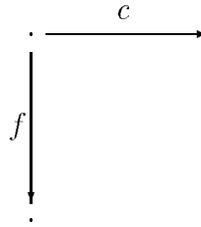
Since the high-level combinators for parameterization and implementation are defined only in terms of the primitive combinators for interpretation and conservative extension, we need to identify those morphisms in the category of configured objects which are conservative and essential for building safely. The property of pushout completion that we require represents the Craig interpola-

tion property of the underlying logic.

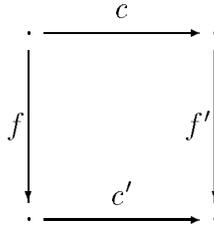
Definition 4.1.1 (A Category of Configured Objects) Let \mathcal{C} be a small category such that any element of $Obj \mathcal{C}$ is a configured object and is denoted by O . If $f : O_1 \rightarrow O_2$ is a morphism of \mathcal{C} then f is called an *interpretation*. Let \mathbf{Cons} be a sub-category of \mathcal{C} such that $Obj \mathbf{Cons} = Obj \mathcal{C}$. A morphism $c : O_1 \rightarrow O_2$ is a *conservative extension* of O_1 iff $c \in Mor \mathbf{Cons}$. The category, \mathcal{C} , of configured objects is finitely cocomplete. \square

Definition 4.1.2 (A Diagram in a Category) Let \mathcal{G} be a finite category. A *diagram* (of shape \mathcal{G}) in a category \mathcal{C} is a functor $D : \mathcal{G} \rightarrow \mathcal{C}$. \square

Definition 4.1.3 (Pushout Completion Property) Let \mathcal{C} be the category of configured objects. Then \mathcal{C} has the *pushout completion property* if for any diagram



where $c \in Mor \mathbf{Cons}$, there exists its colimit



such that c' also belongs to $Mor \mathbf{Cons}$. \square

Definition 4.1.4 (The KZ-Doctrine for Configuration) Let an abstract KZ-doctrine on the 2-category \mathbf{Cat} be denoted by $\mathbf{T} = (T, y, m, \lambda)$. The *KZ-doctrine for configuration* has the structure of \mathbf{T} and in addition associates to each small category its completion by finite colimits. The finitely cocomplete categories with functors that preserve the finite colimits are the algebras for the KZ-doctrine. The KZ-doctrine for configuration is denoted by

$$\mathbf{KZ}_{Conf} = ((-)_\mathcal{D}, make\text{-}diagram, flatten\text{-}diagram, \lambda)$$

\square

Definition 4.1.5 (The Configuration Workspace) Let \mathcal{C} be a small category in the 2-category \mathbf{Cat} with the pushout completion property. The *configuration workspace*, denoted by \mathcal{W} , consists of the categories $\mathcal{C}, \mathcal{C}_{\mathcal{D}}, \dots, \mathcal{C}_{\mathcal{D}^n}$ and the adjoint functors between them that are the components of \mathcal{C} and belong to the KZ-doctrine $\mathbf{KZ}_{\mathcal{C}_{conf}}$. The category $\mathcal{C}_{\mathcal{D}}$ is the free-cocompletion of \mathcal{C} . \square

4.2 Configuring Specifications and Modules

The category \mathcal{C} within our framework for configuration has as its objects the unstructured theory presentations of all those components that can be used in the configuration of complex systems. The actual configuration of the specifications and modules for these objects, however, takes place by structuring specifications, as diagrams, within the workspace of the KZ-doctrine.

Definition 4.2.1 (A Specification) Let O denote a configured object in $Obj \mathcal{C}$. Then a *specification* for O , is given by a diagram, denoted by D , which is an object in the category of diagrams $\mathcal{C}_{\mathcal{D}^n}$ where $n \geq 1$. A specification with no previous history of configuration is defined to be a primitive specification and is given by a singleton diagram in the category $\mathcal{C}_{\mathcal{D}}$. A specification that has been configured by $n - 1$ applications of a high-level combinator for structuring or implementation is given by a diagram in $\mathcal{C}_{\mathcal{D}^n}$. All specifications of all objects in \mathcal{C} are included in the limiting category of the configuration workspace, $\lim_{n \rightarrow \infty} \mathcal{C}_{\mathcal{D}^n}$. \square

Definition 4.2.2 (Shapes for the High-Level Combinators) The operation of each high-level combinator is defined in terms of the primitive combinators by the *shape* of a simple graph. The diagram of each configured specification has an underlying shaped graph given by the high-level combinator used in the previous configuration. The operation of each high-level combinator is defined on each simple shaped graph, \mathcal{G} , that belongs to the set of shaped graphs \mathcal{G} . The extended shaped graph that results from the operation is labelled by a diagram as a functor into the appropriate category of diagrams, $\mathcal{C}_{\mathcal{D}^n}$, in the configuration workspace. \square

Definition 4.2.3 (Configuring a Specification in the Workspace) The *configuration of a new specification from the specifications or modules in the workspace* takes place in the following stages:

1. The operation of each high-level combinator on configured objects is expressed by a diagram $D : \mathcal{G} \rightarrow \mathcal{C}_{\mathcal{D}^n}$. This diagram, with specifications or modules at the nodes and arrows that represent the primitive combinators, is an object in $\mathcal{C}_{\mathcal{D}^{n+1}}$. The category $\mathcal{C}_{\mathcal{D}^n}$ is chosen because the maximum number of configurations made for any of the specifications at the nodes of the diagram is n . Any specifications at the nodes that were

configured in fewer than n configurations will be included in $\mathcal{C}_{\mathcal{D}^n}$ by the appropriate component of *make-diagram*.

2. Using the appropriate number of applications of *flatten-diagram*, the colimit of D is formed as a flattened diagram, D' , which is an object in $\mathcal{C}_{\mathcal{D}}$. Although the encapsulation around the objects that are at the nodes of D (themselves diagrams) is removed, all the components of the diagrams and the arrows between them are preserved.
3. The colimit object for the flattened diagram D' is then associated with D' by the functor $\lim_{\leftarrow} : \mathcal{C}_{\mathcal{D}} \rightarrow \mathcal{C}$ which is left adjoint to $\text{make-diagram}_{\mathcal{C}}$. The colimit cocone is constructed over a diagram in \mathcal{C} with the colimit object at its vertex.
4. Using the appropriate number of applications of *make-diagram*, the singleton diagram of the colimit object $\lim_{\leftarrow}(D')$ is formed in $\mathcal{C}_{\mathcal{D}^n}$, where it is at the vertex of D and at the end of the arrows in the cocone, constructed in \mathcal{C} , to the colimit object. Constructing the colimit object and the initial step of forming it into a singleton diagram in $\mathcal{C}_{\mathcal{D}}$ is carried out by the unit of the adjoint situation $\lim_{\leftarrow} \dashv \text{make-diagram}_{\mathcal{C}}$.
5. The diagram in $\mathcal{C}_{\mathcal{D}^n}$, formed by D and the arrows to the singleton diagram of the colimit diagram at its vertex, represents the new configured specification as an object in $\mathcal{C}_{\mathcal{D}^{n+1}}$. The name of the new specification is given by the name of the colimit singleton diagram at its vertex.

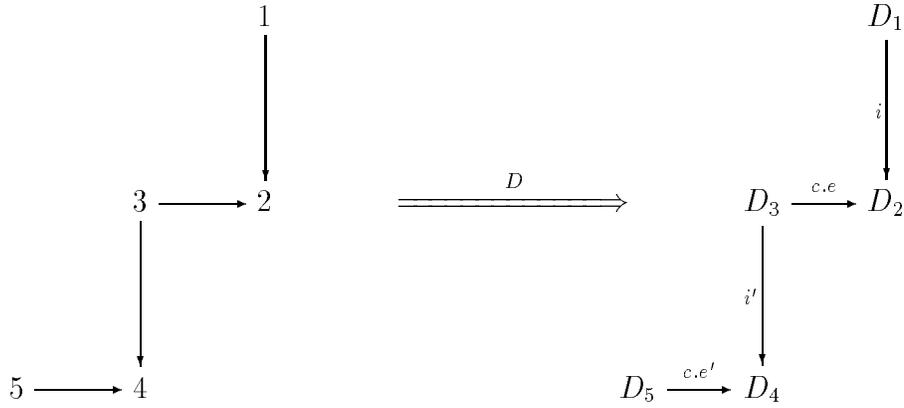
□

From each specification a unique module instance can be created by taking the colimit of the flattened diagram for that specification, as an object in $\mathcal{C}_{\mathcal{D}}$.

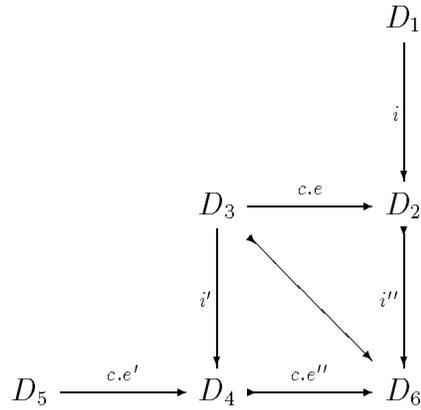
Definition 4.2.4 (Creating a Module Diagram) Let D' denote the diagram for a specification in the category $\mathcal{C}_{\mathcal{D}^n}$ where $n \geq 1$. The *creation of a module diagram* from D' is made initially by $(n - 1)$ applications of the functor $\text{flatten-diagram}_{\mathcal{C}}$ to form an unstructured diagram, D , in $\mathcal{C}_{\mathcal{D}}$. A *module diagram*, denoted by M , is then defined by $M = \text{make-diagram}_{\mathcal{C}}(\lim_{\leftarrow}(D))$, and is an object in the category $\mathcal{C}_{\mathcal{D}}$. The colimit of D , constructed by \lim_{\leftarrow} , is one of an isomorphic class of objects in \mathcal{C} . The singleton diagram of that colimit object, constructed by $\text{make-diagram}_{\mathcal{C}}$, gives a uniquely named module diagram in $\mathcal{C}_{\mathcal{D}}$. □

For simplicity we configure from primitive specifications which are singleton diagrams and objects in the category $\mathcal{C}_{\mathcal{D}}$. Also we only illustrate the stage of configuring a specification by the vertical combinator for implementation, which is defined by the pair of primitive combinators interpretation and conservative

extension, denoted by $(i, c . e)$. We choose the shape for the application of the combinator for implementation on an already implemented shape. The diagram for that shape, as an object in the category $\mathcal{C}_{\mathcal{D}\mathcal{D}}$, is the functor D and is represented by a double arrow for clarity:



The colimit cocone over the diagram for the composition of implementations is:



Example 4.2.5 The implementation of a set by a sequence can be made by interpretation from **spec set** to **spec extseq** which is constructed as the conservative extension of **spec seq**. The implementation of a set by an array can be configured by the composition of the implementation of a set by a sequence, expressed in **spec set_impl_by_seq**, with the implementation of a sequence by an array, expressed in **spec seq_impl_by_array**. The diagram is:

$$\begin{array}{ccc}
& & set^s \\
& & \downarrow i_{extseq^s}^{set^s} \\
& seq^s & \xrightarrow{c \cdot e_{extseq^s}^{seq^s}} extseq^s \\
& \downarrow i_{extarray^s}^{seq^s} & \\
array^s & \xrightarrow{c \cdot e_{extarray^s}^{array^s}} & extarray^s
\end{array}$$

Forming the colimit for this diagram involves the construction of a pushout with the colimit object $extseq^s \amalg extarray^s$. In `spec set_imp_by_array` the implementation is expressed by the text `spec set impl. by spec array`. \square

Each unique module instance of the implemented specification is created by taking a colimit of the diagram for that specification and forming a singleton diagram in the category $\mathcal{C}_{\mathcal{D}}$.

5 Conclusions

We have applied the structure of a KZ-doctrine to the formal construction of diagrams in a 2-category. This method of building diagrams is then shown to apply to the specific problem of providing a formal framework for the configuration of complex systems in the real world of software engineering. It is important to note that software engineers who use the configuration language are not required to understand the abstract definitions of the categorical framework. We have aimed to make our work original in two directions. First we believe that our choice of the KZ-doctrine provides a more general framework for building specifications as diagrams than previously used. Secondly we extend our earlier work on configuring parameterised specifications to the configuration of implemented specifications and modules by the primitive combinators of interpretation and conservative extension.

Acknowledgments

Steve Vickers suggested using the KZ-doctrine to provide a formal framework for system configuration and gave helpful advice. The many discussions with Tom Maibaum have been very stimulating. Chris Hankin and Sarah Liebert

have made constructive criticisms on the paper. I also want to thank Jose Meseguer for his comments on an early draft of this paper as well as his valuable suggestions for further work. Paul Taylor's macros were used for the diagrams.

References

- [1] R. M. Burstall and J. A. Goguen. Putting theories together to make specifications. In *Proceedings of the 5th. International Joint Conference on Artificial Intelligence*, pages 1045–1058, Cambridge, Mass., 1977.
- [2] R. M. Burstall and J. A. Goguen. The semantics of Clear, a specification language. In *Abstract Software Specifications, LNCS 86*. Springer-Verlag, 1979.
- [3] R. Diaconescu, J. Goguen, and P. Stefanias. Logical support for modularisation. Technical report, Programming Research Group, Oxford University, May 1991.
- [4] G. Hill. Category theory for the configuration of complex systems. In T. Rus M. Nivat, C Rattray and G. Scollo, editors, *Algebraic Methodology and Software Technology, Entschede, 1993*, pages 193–200. Proceedings of the Third International Conference on Algebraic Methodology and Software Technology, University of Twente, The Netherlands, 21–25 June 1993, Springer-Verlag, 1994. Workshops in Computing series.
- [5] G. Hill. The configuration of complex systems. In T. Ören, editor, *CAST '94 Lecture Notes*. Fourth International Workshop on Computer Aided Systems Technology, University of Ottawa, Ottawa, Ontario, Canada, 16–20 May, 1994. to be published in 1995 by Springer-Verlag.
- [6] G. Hill. *A Language for System Configuration*. PhD thesis, Department of Computing, Imperial College, University of London, 1994. draft.
- [7] G. Hill. A logical approach to systems construction. In Rudolph Albecht and Franz Pichler, editors, *Proceedings of EUROCAST '95*. Fifth International Workshop on Computer Aided Systems Technology, Conference Center, Innsbruck, Austria, 22–25 May, 1995. to be published by Springer-Verlag.
- [8] A. Kock. Monads for which structures are adjoint to units, (version 3). *Aarhus Preprint Series*, pages 1–24, June 1993.
- [9] S. MacLane. *Categories for the Working Mathematician*. Springer-Verlag, 1971.
- [10] T. Maibaum, J. Fiadeiro, and M. Sadler. Stepwise program development in II institutions. Technical report, Imperial College, November 1990.
- [11] J. Meseguer. Order completion monads. *Algebra Universalis*, 16:63–82, June 1983.
- [12] D. Sannella. *Semantics, Implementation and Pragmatics of Clear, A Program Specification Language*. PhD thesis, Department of Computer Science, University of Edinburgh, 1982. Thesis CST–17–82.

- [13] V. Sassone, J. Meseguer, and U. Montanari. ω -inductive completion of monoidal categories and infinite petri net completions. *to be published*, 1994.
- [14] W. M. Turski and T. S. E. Maibaum. *The Specification of Computer Programs*. International Computer Science Series. Addison Wesley, 1987.
- [15] L. Úry and T. Gergely. A constructive specification theory. In G. David, R. T. Boute, and B. D. Shriver, editors, *Declarative Systems*. Elsevier Science Publishers B. V. (North Holland) IFIP, 1990.
- [16] P. A. S. Veloso. Program development as theory manipulations. Technical report, PUC/RJ Departamento de Informatica, Rio de Janeiro, Brazil, May 1985. Series: Monografias em Ciencia da Computacao No4/85.
- [17] R. F. C. Walters. *Categories and Computer Science*. Cambridge Computer Science Texts, Cambridge University Press, 1991.