Styles in Heterogeneous Modelling With UML

Marius C. Bujorianu and Manuela L. Bujorianu
Formal Methods Group
Faculty of Computer Science - EWI
University of Twente,
Enschede, the Netherlands
email: L.M.Bujorianu@cs.utwente.nl

March 13, 2008

Abstract

Software development is becoming increasingly heterogeneous, and therefore the formal approaches to heterogeneity are geting very important but, unacceptable extremely complex. We propose a type theoretic approach based on the concept of style, as an attempt to simplify the complex interaction of different formal aspects of system specification. We use category theory to investigate three major semantic styles - algebraic, coalgebraic and relational - and specification methodologies like viewpoints and precise metamodeling.

Keywords: *relational specification, Z, UTP, UML, fibrations, algebra, coalgebra*

1 Introduction

In this work we propose a type theoretic approach to formal heterogeneity that characterizes the contemporary system development. Our approach is based on the *precise metamodeling* [40], which is also known in the field of formal specification as *viewpoint specification* [10].In the precise metamodeling, software models are built from different perspectives and their interconnections can be complex and are formally specified. The metamodeling became popular especially because model driven architecture approaches based on the Unified Modeling Language (UML) [37]. In the UML, software models are described using different diagrammatic languages, usually 'united ' using OCL, a formal first order language. The linking formulas are called, in the UML parlor, *constraints*. The semantics of the UML metamodeling is given using UML itself. Because of that, a strong objection against the UML is the lack of precise semantics and its methods are criticized of missing the necessary mathematical rigor. The precise metamodeling adds to UML metamodeling formal semantics. UML formalisation is an important source of heterogeneity. Almost every important specification notation has been applied to UML. But different diagrammatic views find specific, convenient notations, but different.

Traditionally, systems were decomposed according to functionality. In addition to this, modern approaches favor decompositions according to "aspects" or "viewpoints". Also, these may be interpreted as views of the system functionality from different participants. Viewpoints mean different perspectives on the same system, aspect oriented specifications written in different languages by teams having different backgrounds, etc. Of course, for an implementation we need an *integration* of all these formal descriptions, more specifically a minimal one, called *unification*. The only issue that distinguishes viewpoint specification from precise metamodeling is the construction of unification. The increasing importance of the viewpoint model in software engineering is exemplified by its use in the Open Distributed Processing - ODP standard [9], OO design methodology, requirement engineering, software architecture [27] and reverse engineering.Considering application domains of viewpoint specification one gets richer sources of heterogeneity. In this work we review approaches to heterogeneity and propose new methodologies to strength them. We use categorical type theory and a concept of style, formalised as frames (see section 3).

The paper is structured as follows. In the next section we treat relational specification from a categorical perspective. In section 3 we introduce a type theoretic formalisation of specification languages. Section 4 contains a treatment of heterogeneity based on categorical type theory. In section 5 we interpret precise metamodeling as viewpoint specification. The partial conclusions of this formal experiment are sketched in the last section.

This paper is addressed to readers with advanced knowledge of category theory. All definitions and notations we use hereafter can be found in [5]. The main notions of category theory we use are presented in the Appendix[1].

2 Relational specification

2.1 Relations as predicates

A fibration $Fib : \mathbf{E} \to \mathbf{B}$ (see the Appendix for full definitions) is the categorical way of qualitative reasoning on the elements of a category using the elements of another category. In a typed specification language, as Z or VDM [30], the base category $\mathbf{B}$ of a fibration is generated by a set of basic typed variables $V = \{x_i : T_i \mid i \in 1, n\}$. The objects of $\mathbf{B}$ are all finite products of V, including the empty product 1, and arrows are the smallest set of arrows containing all the projections and identity arrows, and closed under composition and pairing.

Hyperdoctrines[2] were defined by F.W. Lawvere [17] as a categorical model of predicate logic. At the hart of the concept lies the observation that universal and existential quantifications are adjoints to projections. Hyperdoctrines provide a notion of model for first order and higher order logics -HOL.

Every relational algebra admits a logic formulation. This affirmation is based on equivalence between bifibrations and hyperdoctrines, stated in the following result [39] .

Proposition 1 *A hyperdoctrine is a bifibration which is BC and F stable.*

This result is the categorical foundation of techniques used in Z, the unified theory of programming (UTP) [45, 33] and other relational specification logic of moving freely between relational algebras and formal logics.

2.2 Coalgebras

We recall [29] the correspondence between coalgebras and labelled transition system (lts) $(S, \to, L)$, $\to \subseteq S \times L \times S$ when $\mathbf{M}$ is $\mathbf{SET}$. Define $B[X] =$

[1]This is included for the reviewers' convenience and could be omitted for the final version.

[2]In the sixties, the term doctrine was used to denote every equational theory over category of small categories.

$\mathbb{P}[L \times X] = \mathbf{SPP}_{L,X}$ for any set X, that is an endofunctor on $\mathbf{SET}$. The $B-$coalgebra associated to lts is (S, α_S) with $\alpha_S : S \to B[S]$, $\alpha_S(s) = \{(a, s') \mid s \xrightarrow{a} s'\}$. To any $B-$coalgebra (S, α_S) we can attach an lts $(S, \to, A)$ by defining $s \xrightarrow{a} s' \Leftrightarrow (a, s') \in \alpha_S(s)$.

Every binary relation can be substituted equivalently by its functional representation. This technique can be applied to the ternary relation $\to \subseteq S \times L \times S$ in two different way: by giving a function $B : S \to \mathcal{P}[L \times S] = \mathbf{REL}_{L,S}$ or by giving a function $\rho : L \to \mathcal{P}[S \times S] = \mathbf{REL}_S$. The former function describes a coalgebra. Relations involved are heterogenous (between L and S). The last way is more interesting. It associates to every transition label a relation between states. The dynamics of the lts is described entirely relational. This is the way the dynamic systems are described in Z as abstract data types. Relations are homogenous (between the state space). This difference has more impact when considering a structure for the state space. In former case S and L are objects of the same category, which implies that the state structure will be lifted to the action space. The only way we can structure states is then by choosing S to be an object in a category $\mathbf{C}$. For example, considering $\mathbf{C}$ to be $\mathbf{Mon}$ the category of monoids, then the states and actions can be structured by the monoid operator (which could mean parallel or nondeterministic composition). This specification style has been used by Montanari et. al. [16] using the term *structured coalgebras*. In the last case, transitions form always a category different from those of states. If we consider the states as objects of a category $\mathbf{C}$ (which means that all states respect a prescribed structure, which can be formally specified) with pullbacks and mon/epi factorization, then actions will be relations over $\mathbf{C}$, i.e. members of $\mathbf{REL_C} = \mathbf{SP}^{\mathcal{E}}_{\mathbf{C}}$. The impact on software specification is significant: dynamics is not anymore specified as graph, specified locally by giving to each node all its successors, as in case of coalgebra, but generically. A state or a class of states are specified by a logic description defying a category of models. A transition or a finite set of transitions are specified by a finite set of relations. There is a transition if there is a relation whose precondition (i.e. its domain) is valid in that state. In the following, the last way of describing dynamics will be called *relational abstract state machines* (RASMs for short) - see next subsection. These are the categorical generalisations of *abstract data types* from Z (ADTs for short).

We want to construct a similar correspondence for categories $\mathbf{M}$ more general than **SET**. Following the same construction, we define $B[X] = \mathbb{P}[L \times X]$, which implies that $L \in |\mathbf{M}|$. This makes sense if $\mathbf{M}$ is the category of schemas (with inclusion providing morphism), as in Z operations and states are specified using the same schema notation. In order to use $\mathbf{M}$ as the category of models, we define $F[L] = \mathbf{REL}_{M,M'}$ and the transition $M \xrightarrow{L} M'$ iff $M, M' \in F[L]$. But, in this case, F fails to be an endofunctor. A closer look at definition of B on **SET** shows that what makes things work in this case is the fact **SET** has power objects, and thus the powerset construction $\mathbb{P}$ generates an endofunctor. This suggests a more general technique, namely to replace **SET** with a category $\mathbf{M}$ with power objects, in which all relations can be represented as set valued functions. Such categories are the toposes.

Proposition 2 *A category with pullbacks and power objects, in which all relations are representable, is a topos.*

In the rest of this paper, the carrier of any coalgebra is supposed to be an objects of the topos of models **MOD**.

A $\wp-$*coalgebra* is a function from a state set ST to the powerset of $L \times ST$. Using the characterisation (in a topos only) of relations as set valued functions, we can formulate a relational version of coalgebras. A *relational coalgebra* over a set of states ST with actions from a set L is defined as a relation from an object ST in a topos to an object $L \times ST$. Thus we can represent a coalgebra as predicate in the logic of a hyperdoctrine. Using the coalgebraic semantics of CCS (i.e. of lts's) [29], we get again hyperdoctrine representation of operation type.

Proposition 3 *If the state type of Z admits an initial algebra, then the state and operation type of Z admit a final coalgebra.*

2.3 Relational refinement

We define the *operation name space* as a discrete category category $\mathbf{L}$ and the *RASM* as a tuple $A = (\mathbf{S}, \mathbf{L}, \mathcal{D}_{\mathbf{S}})$, where $\mathbf{S}$ is a category (the *state space*) and $\mathcal{D}_{\mathbf{S}}$ is a functor (the *dynamics*) from $\mathbf{L}$ to $\mathbf{REL}_{\mathbf{S}}$.

In Z ADTs, $\mathbf{L} = \mathbf{IN} \times \mathbf{OP}^* \times \mathbf{FIN}$ and $\mathbf{OP}^*$ is the free monoid on a set OP construed as a one-object category. The notation $s \xrightarrow{Op} s'$ stands for $(s, s') \in \mathcal{D}[Op]$, $Op \in |\mathbf{S}|$, that is, the states s, s' are related by the operation Op. Giving a relation $\rho \in |\mathbf{REL}_{\mathbf{C}}|$ we define its *precondition* as *pre*_$\rho = id \cap \rho\rho^{-1}$ and its *postcondition* by *post*_$\rho = id \cap \rho^{-1}\rho$. Obviously, *pre*_$\rho$ and *post*_ρ can be seen as subobjects in $\mathbf{C}$ (for example *pre*_$\rho \subseteq id$ and can be identified with the set $\{c \mid \{c\} \in |\mathbf{C}| \wedge (c,c) \in id \cap \rho\rho^{-1}\}$). There is an issue for the situation when a state $S \in |\mathbf{C}|$ has an empty intersection with *pre*_ρ. In the *contract interpretation*, there is no transition associated with the label defined by ρ. In the *behavioral interpretation*, the relation ρ can define any possible transition.

Given two RASMs $(\mathbf{S}, \mathbf{L}, \mathcal{D}_\mathbf{S})$ and $(\mathbf{T}, \mathbf{L}, \mathcal{D}_\mathbf{T})$, a *backward simulation* between them is defined by a relation $\rho : T \nrightarrow S$ such that

$$\forall Op \in \mathbf{L} \bullet (t,s) \in \rho \wedge t \overset{Op}{\to} t' \Longrightarrow \exists s' \in S \bullet s \overset{Op}{\to} s' \wedge (t', s') \in \rho.$$

Therefore, the following formula also takes place

$$\forall Op \in \mathbf{L} \bullet (s,t) \in \rho \wedge t \overset{Op}{\to} t' \Longrightarrow \exists s' \in S \bullet s \overset{Op}{\to} s' \wedge (s', t') \in \rho.$$

This latter expression admits the following diagrammatic interpretation in $\mathbf{REL}$:

$$\begin{tikzcd} S \arrow[r, "\rho"] \arrow[d, "Op_\mathbf{S}"'] \arrow[dr, phantom, "\Downarrow"] & T \arrow[d, "Op_\mathbf{T}"] \\ S \arrow[r, "\rho"'] & T \end{tikzcd}$$

for every operation name $Op \in \mathbf{L}$. This amounts to a lax-transformation $\rho : \mathcal{D}_\mathbf{S} \Rightarrow \mathcal{D}_\mathbf{S} : \mathbf{PROG} \to \mathbf{REL}$. An immediate consequence is that forward simulations compose via lax transformations (by pasting the lax-squares), hence they are closed under relational composition.

The lifting of fibration of subobjects $\iota : Sub[\mathbf{SET}] \to \mathbf{SET}$ to relations gives a functor $F_\iota : \mathbf{REL} \to \mathbf{FRM}$, the extension of the ordinary logic of sets. The composite $F_\iota \circ \mathcal{D}_\mathbf{S} : \mathbf{PROG} \to \mathbf{FRM}$ yields a new RASM, whose state space is a frame (of subsets or $Sub-$predicates). We denote this RASM $\mathcal{PA} = (\wp[\mathbf{S}], \mathbf{L}, \langle \mathcal{D}^*_\mathbf{S} \rangle))$ on the frame of subsets of S (identifying predicates on a set with their extents), with

$$< Op >^* = \{(\phi, \psi) | \phi \equiv < Op > \psi\}$$

which restricts to a RASM on the set of formal specifications Φ. We have endowed Φ with the structure of a RASM. We now seek to characterise the satisfaction relation as a forward simulation:

Proposition 4 *For a RASM $A = (\mathbf{S}, \mathbf{L}, \mathcal{D}_{\mathbf{S}})$, the satisfaction relation $\models_S$: $S \nrightarrow \Phi$ satisfies: $\forall Op \in \mathbf{L}$*

$$\begin{tikzcd} S \arrow[r, "\models"] \arrow[d, "Op_{\mathbf{S}}"'] & \Phi \arrow[d, "<Op>^*"] \\ S \arrow[r, "\models"'] & \Phi \end{tikzcd}$$

Thus, $\models$: $S \nrightarrow \Phi$ is a forward simulation.

Given a set S and a sup-lattice U, a relation $\rho : S \nrightarrow U$ is called a σ-relation if its transpose $\widehat{\rho} : F \longrightarrow \wp S$ is a sup-lattice homomorphism (where $\widehat{\rho}[f] = \{s \mid (s, f) \in \rho\}$). Given RASMs $(\mathbf{S}, \mathbf{L}, \mathcal{D}_{\mathbf{S}})$ and $(\mathbf{U}, \mathbf{L}, \mathcal{D}_{\mathbf{U}})$, with U a sup-lattice, a relation $\rho \subseteq S \times U$ is called a *σ-forward simulation* if it is both a σ-relation and a forward simulation.

Theorem 5 *Given a RASM $A = (\mathbf{S}, \mathbf{L}, \mathcal{D}_{\mathbf{S}})$, the satisfaction relation $\models_S$: $S \nrightarrow \Phi$ is the largest σ-forward simulation between A and $(\Phi, \mathbf{L}, \langle \mathcal{D}^*_{\mathbf{S}} \rangle))$.*

Corollary 6 *Consider RASMs A and B and a forward simulation $\rho : S \nrightarrow T$ between them. If $(s, t) \in \rho$ and $t \models_T \phi$ (for a formula ϕ), then $s \models_S \phi$.*

This means that two states s and t related by a forward simulation are observationally foward-similar, i.e. any formal specification satisfied by t is also satisfied by s.

Lemma 7 *Given a σ-relation $\rho : S \nrightarrow F$ and a relation $R : T \nrightarrow S$, the composite $R \bullet \rho : T \nrightarrow F$ is a σ-relation.*

Given RASMs A and B and states $s \in S$ and $t \in T$, which are opsimilar:

$$\forall \phi \in \Phi \bullet (t \models_T \phi) \Longrightarrow (s \models_S \phi)$$

implies that s opsimulates t (i.e. that there is a forward simulation relating them), the usual approaches impose restrictions on the systems so that the above condition gives a backward simulation between A and B.

3 Categorical Foundation of Specification Languages

Category theory was proposed as a natural candidate to deal with heterogeneous specifications. The most developed categorical approaches to specifications are based on the theory of institutions. Conceptually, the institution notion can be identified exactly with the concept of (model theoretic) logic framework described in [22]. In this way, the concept of formal specification language is identified with a model theoretic logic.

In principle, an institution is a model-oriented logic described in the categorical language. Most of categorical machinery recasts results from (set-theoretic general logics) and most of newest results are related to the issue of translating logics (transporting the logical structure). The theory has been equally adored by some scientific communities and rejected by others. It is not the purpose of this paper to investigate the promises and the real benefits of the theory of institutions, but to review an existing, largely developed, approach. Many specification languages have got an institutional formalisation. Examples include algebraic specification languages, temporal logics, coalgebraic logics. Benefits of institutions are collected when considering institution morphisms, i.e. translations between logics. This theory is very rich [14], but this might be also an disadvantage from the practitioner point of view. Heterogeneity issue has generated an exploding amount of research in the institutional world. The pioneering work of Diaconescu, within the Café-OBJ project has introduced an abstract rigorous framework for multi-logical software specifications, based on Grothendieck constructions [5]. The practical benefits of this approach were clearly evidenced and the whole abstract machinery was implemented in a concrete rewriting system. Diaconescu's approach was continued by an impressive amount of work by Mossakowski et. al. [35]. The recent developments include views, a categorical concept for heterogeneous software components.

Now we define a type theoretic formalisation of specification language. Cerioli and Zucca [13] firstly voiced the need of generalising the institution concept to accommodate more sophisticated concepts of types.

A *spec space* $(S, \sqsubseteq)$ is a complete join semilattice. Elements of S are specifications and $\sqsubseteq$ models the refinement relation. A *spec space morphism* is a map $m : S_1 \to S_2$ such that $m(\sqcup F) = \sqcup m(F)$ for each nonempty subset F of S. Spec spaces and their morphisms form a category $\mathbf{S}$.

In practice, a specification is constructed in terms of some finitary operations. We model this by asking spec spaces to be co-algebraic.

A cpo $(S, \sqsubseteq)$ is called *co-algebraic* iff for every $s \in S$, $s^{\sqsubseteq} = \{t \sqsupseteq s \mid t \text{ is compact}\}$ is directed and $s = s^{\sqsubseteq}$. Colgebraic spec spaces form a full subcategory $\mathbf{aS}$ of $\mathbf{S}$.

A *constraint oriented development frame* (*COD frame* for short) $(\mathbf{SS}, \mathbf{B}, \mho, \wp, Sem)$ consists of

- A subcategory $\mathbf{SS}$ of $\mathbf{aS}$
- A fibration [15] $\mho : \mathbf{SS} \to \mathbf{B}$
- a posetal hyperfibration $\wp : \mathbf{B}^{op} \to \mathrm{Pr}\,\mathbf{eOrd}_{\vee}$
- a functor $Sem : \mathbf{SS} \to \mathrm{Pr}\,\mathbf{eOrd}_{\vee}$, such that $Sem[Sp] \subseteq \wp[\mho[Sp]]$.

Example 8 (Z) *Let* $\mathbf{SS}$ *be the category of Z schemas, ordered by inclusion, and* $\mathbf{B}$ *the category of records of typed variables and constants. Then* $\Sigma : \mathbf{SS} \to \mathbf{B}$ *defined by* $\Sigma S = S \vee not S$ *(the signature of a schema) is a fibration.*

Example 9 (Unified Theories of Programming) *The alphabetised relational calculus is similar to Z schema calculus, except that it is untyped and rather simpler. An alphabetised predicate* $(P, Q, ..., true)$ *is an alphabet-predicate pair, where the predicate's free variables are all members of the alphabet. The alphabet is composed of undecorated variables* $(x, y, z, ...)$ *and dashed variables* $(a', b', ...)$*; the former represent initial observations, and the latter, observations made at a later intermediate or final point. The alphabet of an alphabetised predicate* P *is denoted* αP *and it defines a fibration. It may be divided into its before-variables* $(in\alpha P)$ *and its after-variables* $(out\alpha P)$*. A homogeneous relation has* $out\alpha P = in\alpha P'$*, where* $in\alpha P'$ *is the set of variables obtained by dashing all variable in the alphabet* $in\alpha P$*. A condition* $(b, c, d, ..., true)$ *has an empty output alphabet. Two distinguished variables are introduced into the alphabet of relations: okay records the observation that a program has started; okay′ records the observation that the program has terminated. A design is a pair of predicates* $P \vdash Q$*, where neither predicate mentions okay or okay′. It is similar to Morgan's specification statement* $w : [P, Q]$*, for an alphabet that contains* w *and* w'*. A design has the following meaning:*

$$(P \vdash Q) \widehat{=} (okay \wedge P \Rightarrow okay' \wedge Q)$$

where P is the precondition and Q is the postcondition. The categorical semantics of a design is a predicate transformer [36]. The separation of precondition from postcondition allows a specification style where one might use a more generous precondition than the domain of the specification semantics (which is a relation as in Z). The logic of designs is given by tripos (and thus it is higher order).

The intuition behind the definition of COD frames is that the objects of **SS** form a class of axiomatic specifications, closed under consequence relation. The fibration $\mho$ describes the vocabulary used to build a specification, and the hyperfibration $\wp$ describes all possible structures implementing a given vocabulary. From these structures, the functor *Sem* selects those that are precisely the models of a specification.

The most obvious example of posetal hyperfibration is the powerset functor, essential in modelling the Z type system in [11]. Many important examples from type theory can be found in [17].

Most algebraic specification and model oriented languages, as Z and VDM, are instances of COD frames.

The contravariance of the hyperfibration $\wp$ means that in the refinement process one gets less models by adding more formulas to specification. Adding one more formula to a specification means a new constraint on the system to be developed.

OCL constitutes an important motivation for generalizing institutions to COD frames. OCL has a rich collection of types, and some of them are very complex, as model types. The full categorical formalisation of OCL as a COD frame will be given in a forthcoming paper. The first steps were started in [7, 11].

A *meta-model development frame* (*MMD frame* for short) $(\mathbf{SS}, \mathbf{B}, \mho, \wp, Sem, \mathbf{MOD})$ has a similar definition as COD frame. The difference is that the hyperdoctrine component is nonposetal, pseudofunctorial and defined on the dual category of vocabularies: $\wp : \mathbf{B} \to \mathbf{MOD}$, where **MOD** is some subcategory of **CAT**, usually the category of Cartesian closed categories.

An important example of MMD frames is the (higher order) categorical logics of lts and their many variants. An lts P is a diagram

$$\Sigma_P \underset{\lambda}{\leftarrow} T_P \underset{\varrho}{\overset{\delta}{\rightrightarrows}} S_P \underset{\iota}{\leftarrow} 1 \qquad (1)$$

in **B**. The operations λ, δ and ϱ assign to each transitions respectively a label, a source and a target; the constant ι is the initial state. A morphism

$\varphi : P \to Q$ of labelled transition systems is a natural transformation between diagrams as (1)

$$\begin{tikzcd}
\Sigma_P \arrow[r, "\lambda"'] \arrow[dd, "\Sigma_\varphi"'] & T_P \arrow[r, shift left, "\delta"] \arrow[r, shift right, "\varrho"'] \arrow[dd, "T_\varphi"'] & S_P \arrow[dd, "S_\varphi"'] & \\
 & & & 1 \arrow[ul, "\iota"'] \arrow[dl, "\iota"] \\
\Sigma_S \arrow[r, "\lambda"] & T_Q \arrow[r, shift left, "\delta"] \arrow[r, shift right, "\varrho"'] & S_Q &
\end{tikzcd}$$

So φ is in fact a triple of functions $({}^{\Sigma}\varphi, {}^{T}\varphi, {}^{S}\varphi)$. The latter two form a graph morphism, preserving the initial state and the labelling - in the sense that it takes an a-labelled transition to a ${}^{\Sigma}\varphi(a)$-labelled one. With morphisms like this, the labelled transition systems form a category **LTS**. It is fibred by the functor $\Sigma_{\mathbf{LTS}} : \mathbf{LTS} \to \mathbf{B}$, which projects each labelled transition system to the corresponding alphabet. $\Sigma_{\mathbf{LTS}}$ is a regular fibration [39]. There is a regular fibration $\mathbf{LTS}^{\tau} \to \mathbf{B}$, spanned by reachable labelled transition systems, where each state can be reached from the initial state. The inclusion $\mathbf{LTS}^{\tau} \hookrightarrow \mathbf{LTS}$ has a right adjoint, i.e. reachable labelled transition systems span a coreflexive subcategory of labelled transition systems. The fibred subcategory $\mathcal{T}$ of synchronization trees is coreflective in all of them and supports the full higher order predicate logic. A fibration of asynchronous transition systems is described in [44].

The covariance of the hyperfibration $\wp$ means that in the refinement process one gets less models by eliminating formulas from specification. Adding one more formula to a specification means a new building rule for the models of the system to be developed. In an MMD frame the models of specifications are not specified directly, but rules to build them are described instead. This style is specific to meta-modeling. The most prominent examples of this style are UML, the structured operational semantics, grammars and graph transformation systems.

4 Herogeneity via structuring categories

A *structuring category* $St[Sp]$ for a specification Sp can be thought of as a category (with some specific properties) which, in some sense, is the smallest such category in which Sp can be modelled soundly.

A COD frame is called **C**-*structured* if there is a category **C** (called the *structuring category*) with pushouts and with a faithful functor $\mathbf{C} \to \mathbf{CAT}$ such that :

- There are functors $U : \mathbf{C} \to \mathbf{SS}$ and $F : \mathbf{SS} \to \mathbf{C}$ with F left adjoint to U.
- The functor $Sem : \mathbf{SS}^{op} \to \Pr\mathbf{eOrd}_{\vee}$ is naturally isomorphic to the functor

$$\mathbf{SS}^{op} \overset{F^{op}}{\to} \mathbf{C}^{op} \overset{\_/\mathbf{C}}{\to} \Pr\mathbf{eOrd}_{\vee}$$

where the functor $\_/\mathbf{C}$ sends an object $c \in |\mathbf{C}|$ to the slice category $c/\mathbf{C}$.

Examples

- the equational logic, where **C** is the *category of categories with product*;
- the *ADT logic*, where **C** is the *category of elementary toposes* [11*]*;
- the logic of VDM, where **C** is the *category of partial map categories* [40];
- the logic of the *typed lambda calculus*, where **C** is the *category of Cartesian closed categories*;
- the logic of the *polymorphic lambda calculus*, where **C** is the *category of relatively Cartesian closed categories;*
- the *Girard's linear logic*, where **C** is Seely's *category of linear categories.*
- the *logic of Martin-Lӧf type theory with equality types*, where **C** is the *category of locally Cartesian closed categories*;
- the logic of *Martin-Lӧf type theory without equality types*, where **C** is either Cartmell's *category of contextual categories.*

We say that two specifications are equivalent (relation denoted by using $\cong$ symbol) if their structuring categories are equivalent.

Every structuring category gives rise to a specification in that logic, and this process is mutually inverse to that of constructing a structuring category for a given specification.

In this paper, we make this correspondence more precise for equational logic, which is at the heart of the most specification languages.

Proposition 10 *For any category* $\mathbf{C}$ *with finite products, we can associate a particular equational specification* $Sp[\mathbf{C}] = (Sig, Eqns)$*. Moreover, there is a canonical model of* $Sp[\mathbf{C}]$ *in* $\mathbf{C}$*.*

The equational specification $Sp[\mathbf{C}]$ allows one to reason about the category $\mathbf{C}$ thought of as the category of sets and functions.

Proposition 11 *For every equational specification* Sp *we have* $Sp \cong Sp[St[Sp]]$

This statement is extremely useful because it establishes that the categories with finite products provide a representation of the notion of equational specification, which is syntax independent.

The benefits of the type theoretic approach can be used to define specification language translation. For example, the translation $Tra : Sp_1 \rightarrow Sp_2$ of equational specifications can be given as a finite product preserving functor $Tr : St[Sp_1] \rightarrow St[Sp_2]$. Using functor equivalence, this amount gives a functorial model $T : Sp_1 \rightarrow St[Sp_2]$. In order to extend this idea to full specification languages, categories of COD frames need to be defined. Using functors that relate categories of partial maps with triposes [40], one can get a categorical syntax free interpretation of the practical translation methods between Z and VDM proposed in [30].

5 Precise Metamodeling using Category Theory. Viewpoints

The formal metamodeling is a precise technique for the definition of multiple-view languages like UML [37].

In [25] the metamodeling is defined as the use of different models to clarify different important aspects of the system, but it has to be taken into consideration that these models are dependent on each other and are semantically overlapping. Therefore it is necessary to state how these models are related. The different views on a system have to be semantically compatible and there are several constraints between them. In [10], viewpoints are defined in similar terms.

In UML parlor the concept of model corresponds to our term of viewpoint. Viewpoint specification offers specific challenges to computer science, like the consideration of multiple development techniques, expressing formally *correspondences* (i.e. interconnections, overlapping) between specifications and the construction of unifications (all concepts defined in this section).

Reflection [32] means the capability of a formal language to represent its own semantics. Reflection is a key characteristics of MOF [37]. UML state machines can be formalized in an institutionalized logic language, and constraints on their behavior can be imposed. This approach is closely related to what in formal methods is called s*hallow embedding* of operational semantics: both, the language and its semantics, are formalized into logic (denotationally defined language). This situation is opposite to the deep embedding, where the semantics is embedded in the semantics of the formalizing language.

There are several well-established viewpoint specification techniques. We use the methodology presented in [10,9]. Correspondences are defined as relations, and viewpoint integration is defined in Z, based on a relational semantics.

By using category theory, we can treat abstract data type viewpoints in a proper manner. We can characterize unifications using limits and colimits (pushouts). Correspondences can be expressed as spans. Consistency conditions can be formulated using categorical logic [17].

Viewpoints are just formula in categorical logic. Typical examples are algebraic specifications and data definitions in Z. The unification is constructed by pushout [11].
A (partial) specification *Ref refines* (by *model containment* MC) a viewpoint Psp and we note this by $Ref \sqsupseteq Psp$ if $Sem[Ref] \subseteq Sem[Psp]$
We adopt the view on viewpoints consistency expressed in [10] by "A collection of viewpoints is consistent if and only if it is possible for at least one example of an implementation to exist that can conform to all viewpoints."
The *unification* $Unif[Psp, Psp']$ of two viewpoints Psp and Psp' is defined as the smallest common refinement [10].
The unification of viewpoint based on MC refinement is constructed by taking the pushout of the correspondence diagram. This construction is specific to COD frames [12].

The unification of viewpoint RASMs is constructed by adapting unification procedure from Z [10, 11, 7]. The unification $StU\rho$ of two state spaces StA and StB, connected by a relation $\rho \subseteq StA \times StB$ is given by

$$StU\rho = \rho \cup (StA \setminus\ dom\_\rho) \times \{\bot_A\} \cup (StB \setminus\ ran\_\rho) \times \{\bot_B\}$$

The semantics of every state, let say StA, is enriched with a new symbol, let say $\perp_A$, that is put in correspondence with every element outside the domain (or range) of the correspondence relation. State unification is characterised categorically in [11] by an extension of the pushout construction, called relational pushout.

The unification $U_{A,B}$ of transitions A (acting on StA) and B (acting on StB) is

$$\begin{aligned} pre_{U_{A,B}} &= (pre_A \vee pre_B) \text{ and} \\ post_{U_{A,B}} &= (pre_A \Rightarrow post_A) \wedge (pre_B \Rightarrow post_B) \end{aligned}$$

where pre_A (resp. $post_A$) is the precondition (resp. postcondition) of A etc.

Theorem 12 *Unification ADT is the least developed refinement of the viewpoint ADTs.*

Corollary 13 *Unification of operations can be constructed as a limit of coalgebras.*

This corollary and operation unification for RASMs, show that, for MMD frames, unification is a limit. The frame duality is rediscovered in duality of unification procedures.

We describe now a quick application of the last theorem to process algebra. Consider CCS coalgebraic semantics as described in [29] together with the trace refinement..

Proposition 14 *The unification of two CCS processes is given by parallel composition.*

Viewpoints are a fundamental vehicle in UML. Every diagram express a particular view on the system, and viewpoint integration is usually done by formalizing different UML diagrams in a single, monolithic formalism, like Object Z. UML was extended with model viewpoints in [7], and package semantics was formalized in a generic manner using relational data types.

The UML relational formalisation can turn our approach into metamodeling framework. For example, the hyperfibration of binary relations can be represented in UML as in Figure1.

Relational metamodels are expressed using a subset of UML comprising class diagrams and OCL for writing constraints, which is very similar to the

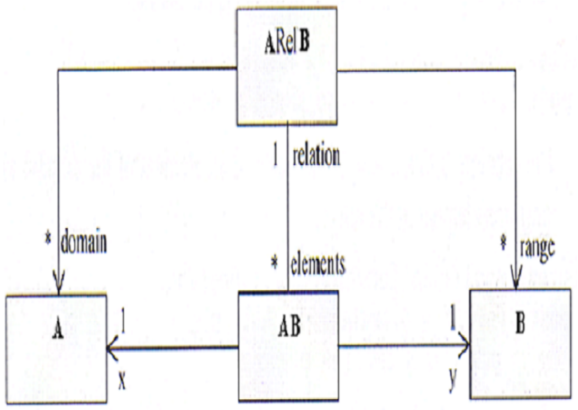


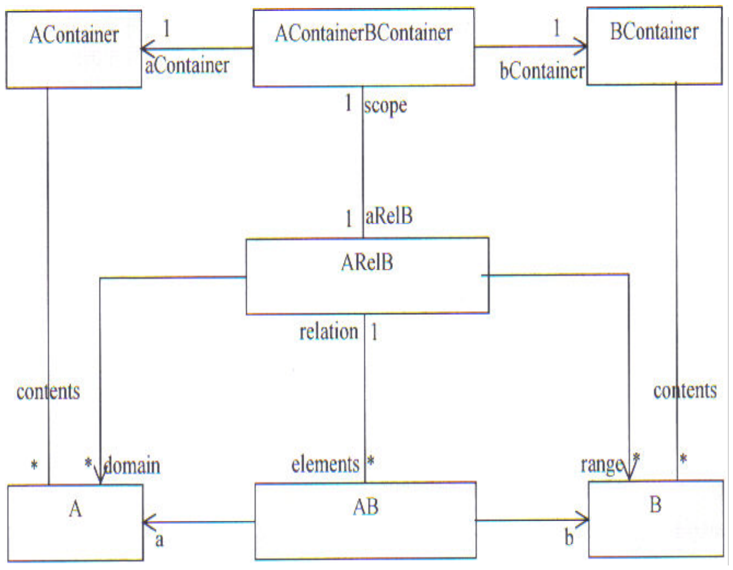
Figure 1: Relations class diagram

language used by MOF [42]. For example, the Figure 1 must be accompanied by a least an OCL invariant which ensures that, looking up an element from the domain returns the pairs in the relation, which mentions that element:

context XrelY : : rangeLookup(Y:y) : Set (XY)

post: result = elements->select(p | p.y = y)

A metamodeling illustration is the typical example of containers (see a more elaborate discussion in [1]).



6 Conclusions and Future Work

In this paper we have treated the difficult, but important problem of heterogeneous use of specification languages. Our approach is founded on categorical type theory. We are a bit unhappy because we have had to use heavily category theory at an advanced level. The limited space of this paper does not allow us to present a proper category theory background. But the interested reader could find excellent introductions in the literature [39, 36, 15].

Inspired by algebra/coalgebra duality we have defined two dual categorical concepts of specification languages. Instead of using algebraic or coalgebraic semantics, we have considered refinement as a primary object. Specifications are sets closed under logical consequence relations and they are ordered by a refinement relation (the reader familiar with the theory of institutions could consider inclusion morphisms as a quick example). Semantics is given by a functor from the category of specification to an ordered category of models. This functor can have covariant or contravariant behaviour. Contravariance is specific to a constraint-oriented style. A formula component of a specification expresses a constraint on the class of models of the specification. Adding a formula to a specification (i.e. refining it) yields less models. For example, this is the case refinement is defined for institutions. The covariance behaviour of the semantics functor means that restriction of the class of models (i.e. refinement) is achieved by eliminating syntactic components from the specification. This is the case for specification style where models are not described directly, but instead the semantics consists of rules for constructing the intended models. This is specific to metamodeling where metadata are specified instead of the intended data. Other instances of this style include structured operational semantics, grammars and graph transformation systems. We have called this specification style metamodeling oriented.

The duality constraint-oriented / metamodeling oriented styles mirrors the algebra / coalgebra duality at a foundation level. The languages with coalgebraic semantics use a constraint-oriented style. Examples include modal and temporal logics or varieties of colgebraic specification. The "Coalgebraic Methods in Computer Science" workshop abounds in papers presenting institutions for coalgebraic logics. But the categorical duality has been never expressed for specification styles.

The precise metamodeling concept we have used in this paper is that presented in [42]. This is an early definition of the concept, as it has got richer interpretations.

Related work. In [6] Baumeister has defined relations as predicates in logics (formalised as institutions). He defines categories of relations between categories of models of algebraic specifications. This construction is similar to our fibrational construction, but comparisons can not be formulated very succinct. Related categorical approach to coalgebraic refinement are presented in [38, 34]. How these approaches relate to relational refinement will be subject of further investigations.

References

[1] D. H. Akehurst, O. Patrascoiu: *Tooling Metamodels with Patterns and OCL*. In Proceedings of the Metamodelling for MDA Workshop, York, November 2003.

[2] M. Anlauf and D. Pavlovic: *On Specification Carrying Software, its Refinement and Composition*, in: H. Ehrig, B.J. Kramer and A.Ertas, eds., *Proceedings of IDPT 2002*, Society for Design and Process Science, 2002.

[3] M. Anlauf and D. Pavlovic: *EPOXI: Evolutionary Programming Over Explicit Interfaces* www.kestrel.edu/home/projects/ dasada/demo-days-020701.ppt

[4] E. Astesiano, G. Reggio: *An Attempt at Analysing the Consistency Problems in the UML from a Classical Algebraic Viewpoint* WADT'02, Springer LNCS, 2003.

[5] M. Barr, C. Wells, **Category Theory for Computing Science** Prentice Hall, 1990.

[6] H. Baumeister *Relations between Abstract Datatypes modeled as Abstract Datatypes* PhD Thesis, University Saarbrucker, 1998.

[7] E.A. Boiten, M.C. Bujorianu: *Exploring UML Refinement through Unification* Workshop on Critical Systems Development with UML, <<U M L>> 2003, San Francisco, California, USA, October 20 - 24, 2003.

[8] E.A. Boiten, J. Derrick (eds.): **IFM 2004: Integrated Formal Methods** Canterbury, Kent, UK, Springer Verlag, LNCS 2999, 2004.

[9] E.A. Boiten, H. Bowman, J. Derrick, P.F. Linington, M.W.A. Steen *Viewpoint consistency in ODP*, Computer Networks **34(3)**, pp. 503–537, 2000.

[10] H. Bowman, E. A. Boiten, J. Derrick, M. W. A. Steen *Strategies for Consistency Checking Based on Unification* Science of Computer Programming, **33**, pp. 261-298, 1999.

[11] M.C. Bujorianu, E.A. Boiten *Towards Correspondence Carrying Specifications*, In [41].

[12] M.C. Bujorianu, S. Maharaj, M.L. Bujorianu *Towards a Formalization of Viewpoints Testing.* In Rob Hierons and Thierry Jeron eds., Proceedings of Formal Approaches to Testing of Software, pp. 137-151, 2002.

[13] M. Cerioli, E. Zucca *Implementation of Derived Programs (Almost) for Free* Recent Trends in Data Type Specification, Springer LNCS 1376, pp. 141–155, 1998.

[14] M. Cerioli. *Relationships between Logical Frameworks*, PhD thesis, University of Genova, 1993.

[15] R. Colomb, C.N.G. Dampney, M. Johnson. *The use of category-theoretic fibration as an abstraction mechanism in information systems.* Acta Informatica, **38** (**1**), pp. 1–44, 2001.

[16] A. Corradini, R. Heckel, U. Montanari, *From SOS Specifications to Structured Coalgebras: How to Make Bisimulation a Congruence*, CMCS '99, ENTCS Vol.19, 1999.

[17] R. Crole **Categories for Types** Cambridge University Press, 1993.

[18] J. R. B. Cockett "Charitable thoughts" 1996. Draft available from http://www..cpsc.ucalgary.ca/projects/charity/home.html

[19] J. Davies, C. Crichton: *Concurrency and Refinement in the Unified Modelling Language* Formal Aspects of Computing 2003.

[20] J. Derrick, D. Akehurst, and E. Boiten. *A framework for UML consistency.* In L. Kuzniarz, G. Reggio ea., eds., <<UML>> 2002 Workshop on Consistency Problems in UML-based Software Development, pages 30-45, 2002.

[21] J. Derrick, E.A. Boiten **Refinement in Z and Object-Z: Foundations and Advanced Applications** Formal Approaches to Computing and Information Technology. Springer, May 2001.

[22] H.D. Ebbinghaus *Extended Logics: The General framework* In J. Barwise, S. Feferman (eds.) **Model Theoretic Logics**, Springer Perspectives in Mathematical Logic Series, 1985.

[23] R.B. France, J.-M. Bruel, M.M. Larrondo-Petrie: *An Integrated Object-Oriented and Formal Modeling Environment* Journal of Object Oriented Programming, **10**(7), pp. 25–34, 1997.

[24] F. D'Souza, A.C. Wills: "Objects, Components and Frameworks with UML: The Catalysis Approach" Addison-Wesley 1998.

[25] R. Geisler, M. Klar, C. Pons. *Dimensions and dichotomy in metamodeling.* In Proceedings of the Third BCS-FACS Northern Formal Methods Workshop. Springer-Verlag, 1998.

[26] J. Goguen and R. Burstall: *Institutions: Abstract Model Theory for Specification and Programming.* Journal of the ACM, 39(1):95-146, 1992.

[27] IEEE Architecture Working Group *IEEE P1471/D5.0 Information Technology -Draft Recommended Practice for Architectural Description*, 1999.

[28] ITU Recommendation X.901-904 | ISO/IEC 10746 1-4. *Open Distributed Processing - Reference Model* - Parts 1-4, July 1995.

[29] P. Johnstone, J. Power, T. Tsujishita, H. Watanabe, J. Worrell *An Axiomatics for Categories of Transition Systems as Coalgebras* LICS 98, IEEE Computer Society Press, 1998.

[30] P.A. Lindsay *On transferring VDM verification techniques to Z*, In Proceedings of Formal Methods Europe (FME'94), Springer LNCS, 1994.

[31] J. Lilius, I.P. Paltor *vUM: a tool for verifying UML models* Proceedings of ASE'99, IEEE Computer Society, pp. 255–258, 1999.

[32] A. Lopes, J.L. Fiadeiro *Preservation and Reflection in Specification.* AMAST'97, Springer LNCS, pp. 380-394, 1997.

[33] Z. Liu, J. He, X. Li, Y. Chen *A relational model for object-oriented requirement analysis in UML* Proc. ICFEM 2003, 2003.

[34] S. Meng, L. Barbosa *Refinement of Generic Software Components* In [41]

[35] T. Mossakowski *Heterogeneous development graphs and heterogeneous borrowing.* Fossacs'02, Springer LNCS 2303, pp. 326-341, 2002.

[36] D. Naumann *Two Categories and Program Structure* PhD Thesis, 1992.

[37] Object Management Group (OMG) *Unified Modeling Language (UML®) and Meta-Object Facility (MOF) Specification Documents.* Available at http://www.omg.org/technology/documents/

[38] D. Pavlovic, D.R. Smith: *Composition and Refinement of Behavioral Specifications*, Proc. of the 16th International Conference on Automated Software Engineering, IEEE Computer Society Press, pp. 157-165, 2001.

[39] D. Pavlovic: *Maps II: Chasing diagrams in categorical proof theory*, Journal of the IGPL 4(2): 1–36, 1996.

[40] M. Proietti *Connections between partial maps categories and tripos theory* In Category Theory and Computer Science, Springer LNCS 283,1987.

[41] C. Ratray, S. Maharaj, C. Shankland (eds.) **Algebraic Methodology and Software Technology. 10th International Conference AMAST 2004**, Springer Verlag LNCS series, 2004.

[42] G. Reggio. *Metamodeling Behavioural Aspects: the Case of the UML State Machines.* In Proc. IDPT 2002. Society for Design and Process Science, USA, 2002.

[43] M. Sun, Aichernig, B., L. S. Barbosa and Z. Naixiao. *A coalgebraic semantic framework for component based development in UML* In Proc. CTCS'04, Elsevier Elect. Notes in Theor. Comp. Sci., 122, pp 229–245, 2004.

[44] G. Winskel, M. Nielsen: Models for Concurrency *Handbook of Logic and the Foundations of Computer Science*, vol. 4, pp. 1-148, Oxford University Press, 1995.

[45] J. Woodcock, A. Cavalcanti *A Tutorial Introduction to Designs in Unified Theories of Programming* In [8], pp 40-66, 2004.

7 Appendix

7.1 Elements of Category Theory

Let **REL** denote the category with sets $A, B, C, ...$ as objects; with relations $\rho : A \leftrightarrow B$ as morphisms, i.e., triples $< A, \rho, B >$ where $\rho \subseteq A \times B$. Let $\rho^{-1} : B \leftrightarrow A$ denote the converse of a relation ρ. Composition is written in diagrammatic order, and is denoted by a semicolon. Application of an functor F to an argument X is denoted by $F[X]$ or by $F.X$. The powerset of A is denoted by $\wp[A]$.

Let **pHa** denotes the category of pre-Heyting lattices [5] and **FRM** denotes the category of frames and sup-preserving morphisms between them (which, being locally ordered, it is a 2-category [36]).

Given a category **C** with pullbacks, a span $< f, g >: B \leftarrow A \rightarrow C$ over **C** is a formed by three objects $A, B, C \in |\mathbf{C}|$ and two arrows (the"legs") $A \xrightarrow{f} B$ and $A \xrightarrow{g} C$.

Many categorical formalisation of relations identify them with spans. This is very tempting because of the very simple span algebra. Pullbacks provide the relational composition, swapping around the span's legs provides the conversion and so on [5]. We note by $\mathbf{SP_C}$, the category of spans over **C** . Moreover, $\mathbf{SP_{SET}}$ is isomorphic to the category of multirelations [5]. In order to construct a category of relations given as spans, we have to restrict the spans to a quotient structure, which 'eliminates' multiplicities.

When one is looking to construct a relational semantics for specification languages, the category of relations $\mathbf{REL_C}$ over a relational category **C** provides a concept that often can be seen as too strong. We recall that $\mathbf{REL_C}$

is obtained from $\mathbf{SP_C}$, the category of spans over a category with pullbacks $\mathbf{C}$, by considering the quotients of spans with respect of equivalence relations generated by regular epimorphisms. A problem is that $\mathbf{REL_{REL}}$ does not exist. This means that, if the plant behaviour has a relational description, we can not distinguish between the plant's and controller's actions.

A natural way to generalize $\mathbf{REL_C}$ is to replace equivalence relations. The natural question is then "What properties a collection $\mathcal{A}$ of arrows must satisfy such that there is an equivalence relation $\equiv_{\mathcal{A}}$on objects characterized as $a \equiv_{\mathcal{A}} b$. iff a and b are related by a span of $\mathcal{A}$ arrows?". As $\equiv_{\mathcal{A}}$ is idempotent, $\mathcal{A}$ must contain all isomorphisms. As $\equiv_{\mathcal{A}}$ is transitive, $\mathcal{A}$ must be closed to composition. The symmetry of $\equiv_{\mathcal{A}}$implies that in the following pullback square

$$\begin{tikzcd}
\bullet \arrow[r] \arrow[d, "f"'] & \bullet \arrow[d, "g"] \\
\bullet \arrow[r] & \bullet
\end{tikzcd}$$

$g \in \mathcal{A}$ implies $f \in \mathcal{A}$.

Let $\mathbf{C}$ be a category with pullbacks. A *cover system* is a collection $\mathcal{A}$ of arrows which contains all isomorphisms and it is closed to composition and to pullbacks along arbitrary arrows.

Examples (other than regular epimorphisms $\mathcal{E}$) include the isomorphisms, the retractions and the monics (denoted $\mathcal{M}$).

We denote by $\mathbf{SP}^{\mathcal{A}}_{\mathbf{C}}$, the category of spans over a category with pullbacks $\mathbf{C}$, factorized by a cover system $\mathcal{A}$.

The category $\mathbf{SP}^{\mathcal{M}}_{\mathbf{REL}}$ is isomorphic with the category of monotonic predicate transformers [36].

A category $\mathbf{B}$ is called *Cartesian* if it has all finite products.

A *fibration* from the *total* category $\mathbf{E}$ to the *base* category $\mathbf{B}$ is a functor $Fib : \mathbf{E} \to \mathbf{B}$ for which, for every $e \in |\mathbf{E}|$ and $u : i \to Fib[e] \in \mathbf{B}$, there is a Cartesian lifting [5] of u.

The fibre category $\mathbf{REL_E}(A_1, A_2)$ is the fibre category $\mathbf{E}_{A_1 \times A_2}$ while $\mathbf{SREL_E}(A)$ corresponds to $\mathbf{E}_{A \times A}$. More concisely, the objects R of $\mathbf{REL_E}$ over context (A_1, A_2) are object of $\mathbf{E}$ over $A_1 \times A_2$ in $\mathbf{B}$; a morphism $f : R \to S$ over $(u, v) : (A_1, A_2) \to (B_1, B_2)$ is a morphism in $\mathbf{E}$ over $u \times v$. Thus, the substitution functor $(u, v)^{\#}$ in $\mathbf{REL_E}$ corresponds to substitution functors $(u \times v)^{\#}$ in $\mathbf{E}$. Moreover, $\mathbf{SREL_E}$ is a subcategory of $\mathbf{REL_E}$ via pullback $\overline{\Delta}$ of the mono Δ. In fact the functor $\overline{\Delta}$ simply expands the context in the base: it sends the relation $R \in \mathbf{SREL_E}(A)$ to the same relation

$R \in \mathbf{SREL}_{\mathbf{E}}(A, A)$.

The functor $OFib : \mathbf{E} \to \mathbf{B}$ is an *opfibration* if $OFib^{op} : \mathbf{E}^{op} \to \mathbf{B}^{op}$ is a fibration. The Cartesian arrows with respect to $OFib^{op}$ are called *opCartesian* with respect to $OFib$. The opcartesian lifting of $f : A \to B$ at P over A is written $\sigma^f_P : P \to f_! P$. A functor is a *bifibration* if it is both a fibration and an opfibration. A morphism of bifibrations must preserve both Cartesian and opcartesian arrows.

A *hyperfibration* is a fibrewise Cartesian closed fibration $HFib : \mathbf{E} \to \mathbf{B}$, such that both $HFib$ and $HFib^{op}$ are bifibrations.

A *posetal hyperdoctrine* is a contravariant functor $\wp : \mathbf{B}^{op} \to \mathrm{Pr}\,\mathbf{eOrd}_{\vee}$ where $\mathbf{B}$ is Cartesian.. We assume that, for each arrow $f \in \mathbf{B}$, the monotone function $\wp[f]$ (often denoted $f^{\#}$) preserves sups and has a left adjoint, written $\exists_f$. We also require the following two conditions:

1. *Beck–Chevalley condition* If the following diagram is a pullback

$$\begin{tikzcd} A \arrow[r] \arrow[d, "g"'] & B' \arrow[d, "h"] \\ A' \arrow[r] & B' \end{tikzcd}$$

and $\phi \in \wp[B]$ then $\exists_g[\ f^{\#}[\phi]] \approx k^{\#}[\exists_h[\phi]]$, where $\approx$ denotes two way refinement.

2. *Frobenius reciprocity* For each $f : A \to B \in \mathbf{B}$ and $\phi \in \wp[A]$ and $\varphi \in \wp[B]$ we have $\exists_f[\ f^{\#}[\phi] \wedge \varphi] \approx \varphi \wedge \exists_h[\phi]$

A *nonposetal hyperdoctrine* is a contravariant functor from a category $\mathbf{B}$ (of 'sets') to $\mathbf{CAT}$, the category of categories. The category $\wp[A]$ is meant to be a category of 'predicates' over the 'set' $A \in \mathbf{B}$.

7.2 Proofs

Proof of Proposition 3.

Consider a signature endofunctor Ω on $\mathbf{SET}$, required to be be a relator with initial algebra U. We denote by $\langle \_ \rangle$ the catamorphism of U. Every monotonic endofunctor Λ on $\mathbf{REL}$ (whose initial algebra models the semantics of state specifications in Z) is the extension of a relator Ω with that it agrees on functions. We call the extension Λ of the signature functor Ω the *state functor*.

We define the extension of the state functor Λ to an *operations functor* Ξ by $\Xi[\lceil\sigma\rfloor] = \lceil{}^{\forall}[\Lambda][\ni_A];{}^{\exists}[\Lambda].\wp^{-1}[\sigma]\rfloor$ for all $\lceil\sigma\rfloor : A \to \Omega[A]$ in $S_{\mathbf{REL}}$.

We prove that ${}^{\forall}\lceil U \rfloor$ is a final coalgebra of $\Xi : S_{\mathbf{REL}} \rightarrow S_{\mathbf{REL}}$, and its anamorphism (denoted, by an abuse of notation $\langle \_ \rangle$ as well) is given by $\langle \lceil \sigma \rfloor \rangle = \left\lceil \wp[\langle \wp^{-1}[\sigma]/(\Lambda. \ni_A)^{-1}\rangle^{-1}] \right\rfloor$
for all $\lceil \sigma \rfloor : A \rightarrow \Omega[A]$ in $S_{\mathbf{REL}}$.

Let C be the target of the initial algebra $U : \Omega[C] \rightarrow C$, and let $\lceil \varsigma \rfloor : A \rightarrow C$ in $S_{\mathbf{REL}}$. Define $g : \mathbb{P}.\Omega[A] \rightarrow \mathbb{P}.\Omega[C]$ in **SET** by $\lceil g \rfloor = \Xi \lceil \varsigma \rfloor$.
First, observe that

$$\begin{aligned}
\wp^{-1}[\sigma;\tau] &= \sigma;\wp^{-1}[\tau] \\
&= \sigma;\wp^{-1}[{}^{\forall}[\Lambda. \ni_A];{}^{\exists}[\Lambda.\wp^{-1}.\varsigma]] = \\
&= \sigma;\ni_{\Omega[A]} /(\Lambda. \ni_A);\Lambda.\wp^{-1}.\varsigma = \\
&= (\wp^{-1}.\sigma)/(\Lambda. \ni_A);\Lambda.\wp^{-1}.\varsigma
\end{aligned}$$

Furthermore,

$$\wp^{-1}[\varsigma;{}^{\forall}[U]] = \wp^{-1}[\varsigma;{}^{\exists}[U^{-1}]] = \wp^{-1}[\varsigma];U^{-1}$$

Using these results, we can obtain the desired equivalence as follows:

$$\begin{aligned}
\lceil \varsigma \rfloor ;{}^{\forall}\lceil U \rfloor &= \lceil \sigma \rfloor ;\Xi[\lceil \varsigma \rfloor] \Leftrightarrow \\
\varsigma;{}^{\forall}[U] &= \sigma;\tau \Leftrightarrow \\
\wp^{-1}[\varsigma;{}^{\forall}[U]] &= \wp^{-1}[\sigma;\tau] \Leftrightarrow \\
\wp^{-1}[\varsigma];{}^{\exists}[U^{-1}] &= \wp^{-1}[\sigma]/(\Lambda. \ni_A);\Lambda.\wp^{-1}.\varsigma \Leftrightarrow \\
\wp^{-1}.\varsigma &= (\wp^{-1}.\sigma)/(\Lambda. \ni_A);\Lambda.\wp^{-1}.\varsigma;U \Leftrightarrow \\
\wp^{-1}.\varsigma &= \left\langle ((\wp^{-1}.\sigma)/(\Lambda. \ni_A))^{-1} \right\rangle^{-1} \Leftrightarrow \\
\varsigma &= \wp[\left\langle ((\wp^{-1}.\sigma)/(\Lambda. \ni_A))^{-1} \right\rangle^{-1}]
\end{aligned}$$

It remains to show that $\wp[\langle ((\wp^{-1}.\sigma)/(\Lambda. \ni_A))^{-1} \rangle^{-1}]$ is a monotonic function on the subset ordering. Since $\lceil A \rfloor .U$ is an isomorphism (because U is an isomorphism and functors preserve isomorphisms), we have

$$\wp[\left\langle ((\wp^{-1}.\sigma)/(\Lambda. \ni_A))^{-1} \right\rangle^{-1}] = \sigma;\tau;{}^{\forall}[U^{-1}]$$

The right-hand side of this equation is the composite of three monotonic functions.

Proof of Proposition 14.

The proof is an immediate consequence of Example 3.11 from [29].