

Monadic Maps and Folds for Multirelations in an Allegory

C. E. Martin and S. A. Curtis

Oxford Brookes University, UK

Abstract. This paper contributes to the unification of semantic models and program development techniques by making a link from multirelations and predicate transformer semantics to algebraic semantics and the derivation of programs by calculation, as used in functional programming and relational program development. Two common ways to characterise iteration, namely the functional programming operators *map* and *fold*, are extended to multirelations, using concepts from category theory, power allegories and monads.

1 Introduction

Multirelations were introduced by Rewitzky [1] in the context of complete atomic boolean algebras, and they generalise relations in the same way that relations generalise functions: they are relations whose target type is a powerset type. Whilst ordinary relations model specifications by relating inputs to output values, multirelations relate inputs to guarantees (predicates) on the outputs, and are thus able to model two kinds of nondeterminism: angelic choices made in “our” interest, and demonic choices made by another party. The ability to express dual nondeterminism is extremely useful for modelling many different kinds of two-party transactions, and indeed multirelations have been used for a variety of such purposes. For example, Dunne [2] uses them to develop an extended substitution language, and in Martin and Curtis [3, 4] they are used for specification of and calculation with resource-sharing and voting protocols.

In the Unifying Theories of Programming (UTP) [5], however, programs are modelled as alphabetised relations expressed as predicates, and only demonic nondeterminism is modelled in Hoare and He’s work in [5]. Cavalcanti et al [6, 7] extended the UTP model to include angelic nondeterminism, providing an explicit embedding of UTP predicates into the predicate transformer model, which is known to be isomorphic to the multirelational semantic model (see [8], or Section 3) and also to the choice semantics, as presented by Back and von Wright [9].

So why investigate properties of multirelations if they are isomorphic to other semantic models? And what contribution does this make to UTP?

The answer can be seen when considering the differences between different semantic models: the style of program development and proof in the UTP model

or in that of predicate transformer semantics [10] is very different from that in algebraic semantics, where programs are directly manipulated within the semantic model. To achieve greater unification between different semantic models it would be greatly beneficial to be able to calculate with and reason about algebras that incorporate more kinds of nondeterminism than functional and relational algebras, for example the dual angelic and demonic nondeterminism seen in predicate transformers, choice semantics and multirelations.

It remains to be seen which semantic model of dual nondeterminism best suits the calculational style, but multirelations would seem to be a good candidate. As will be seen later in this paper, they have clean algebraic definitions, facilitating point-free calculation, and unlike weakest precondition semantics that transform postcondition predicates into precondition predicates, multirelations express programs in a forward manner, relating inputs to predicates on outputs, which is more convenient for humans to think about and manipulate.

Thus, this paper contributes to the unification of semantic models and program development techniques by expanding the multirelational calculus with operators and algebraic laws found within the algebraic style of calculation used for functional and relational program development by Bird and de Moor [11, 12]. The resulting laws could then be mapped back into the UTP framework.

To that end, we (i) use point-free definitions of multirelations and their operators, engendering elegant and streamlined proofs, and (ii) extend common functional programming operators to the multirelational model, hopefully paving the way for similar treatment for more such operators. In particular, this paper addresses the *map* and *fold* operators, which have a more familiar form in multirelations than in predicate transformers.

Maps and folds are useful standard computations on datatypes: maps systematically alter data whilst leaving the overall data structure unchanged, and folds perform a computation over a whole data structure yielding a result value. This paper includes a monadic construction of multirelational maps and folds. Multirelational maps and folds are not new, in fact a different construction for them, using span categories, was presented in [4]. So why are we interested in a second construction? The answer is that ultimately, we hope that this monadic approach will help with the construction of a multirelational version of a further operator from functional programming: the *unfold* operator.

An unfold is a dual kind of computation to a fold [13]: in contrast to a fold, an unfold takes an input value and produces a data structure result. The unfold operator is particularly desirable for a calculus of multirelations because several application areas of angelic and demonic nondeterminism naturally involve an unfold type of computation. For example, many two-player games, with the angel and demon as the two players, are naturally expressed as an unfold: the input value is the starting position of the game, and as the game unfolds, the data structure that emerges is the history of the moves played in the game.

This is still work in progress; we do not currently have a multirelational unfold. The construction method used to define maps and folds in [4] does not obviously dualise, and so in this paper we consider a different technique that

is sometimes known as the monadic extension [14], which is a method that has been successfully dualised in some different domains [15, 16].

For this work we choose the setting of *power allegories*. Allegories [17] serve the algebra of sets and relations in the same way as categories serve the algebra of sets and total functions. Power allegories have some additional properties, including the existence of a power object, but excluding negation. The attraction of this setting (covered in more depth in [18]) is that it lends itself well to point-free reasoning, which engenders elegant and streamlined proofs.

Thus the contribution of this paper is two-fold: to give a clear and readable account of how to extend maps and folds to multirelations using a lax variant of the monadic method in the power allegory setting, and to also illustrate the elegance of point-free algebraic reasoning in the multirelational model. This paves the way for future work on the definition of unfolds.

For any readers not so familiar with the category theory that this work is based on, it is hoped that this paper will still provide a glimpse into the links between the algebraic semantics traditionally used in calculational program development and the semantic models of the UTP framework. In addition, such readers can feel reassured of the existence of solid mathematical foundations behind the development of the calculus of multirelations, including the familiar *map* and *fold* operators.

The paper is structured as follows. Some preliminary definitions are given in Section 2, and the definition of a multirelation in a power allegory is introduced in Section 3. This is followed in Section 4 by the introduction of an equivalent model in which multirelations are viewed as set-valued functions. The resulting Kleisli category is derived in Section 5, and maps and folds for this model are defined in Section 6. For all proofs of laws and lemmas contributed by this paper, please refer to the appendix.

2 Preliminaries

Binary multirelations have previously been defined in the context of sets and relations [1, 3], but in this paper we choose the more general setting of power allegories, because it lends itself well to point-free reasoning in the style of Bird and de Moor [12]. Therefore we begin by covering the basic definition of an allegory as seen in [17], as well as looking at two particular varieties: division allegories and power allegories.

Familiarity with basic category theory is assumed; for more details see [19] for example. The notation $p; q$ will be used for the composition of each pair of arrows $p : A \rightarrow B$ and $q : B \rightarrow C$, and id_A is the identity arrow on object A . Function application will be denoted by juxtaposition.

2.1 Allegories

Recall that a category can be thought of as an abstraction of sets and functions, with objects of the category being analogous to sets, and arrows being analogous to functions. Similarly, an *allegory* is an abstraction of sets and relations:

Definition 2.1. *An allegory \mathbf{A} is a category endowed with three additional operators \subseteq , \cap and \circ , as follows:*

- *Any two arrows with the same source and target objects can be compared with a partial order \subseteq , with respect to which composition is monotonic.*
- *For every pair of arrows $r, s : X \rightarrow Y$ there is an arrow $r \cap s : X \rightarrow Y$, called the meet of r and s , and this is characterised by the following universal property, for all arrows $q : X \rightarrow Y$:*

$$q \subseteq r \cap s \equiv (q \subseteq r) \wedge (q \subseteq s)$$

- *For each arrow $r : X \rightarrow Y$ there is a converse arrow $r^\circ : Y \rightarrow X$ such that for all $s : X \rightarrow Y$, $t : Y \rightarrow Z$ and $u : X \rightarrow Z$*

$$\begin{aligned} (r^\circ)^\circ &= r \\ r \subseteq s &\equiv r^\circ \subseteq s^\circ \\ (r;t)^\circ &= t^\circ;r^\circ \\ (r;t) \cap u &\subseteq (r \cap (u;t^\circ));t \quad (\text{modular law}) \end{aligned}$$

The first of the above axioms, concerning the partial order \subseteq , says that \mathbf{A} is an *order-enriched category*. Such an ordering can be useful for modelling program semantics: it is typically used to define a refinement ordering between arrows.

The category **Rel** of sets and relations is thus the archetypal example of an allegory: the ordering \subseteq is the familiar subset inclusion, the meet of two relations is their intersection and the converse is the relational converse. One important subcategory of **Rel** is the category of sets and total functions (**Set**). The corresponding subcategory of an allegory is characterised via the following definitions:

Definition 2.2. *An arrow $m : X \rightarrow Y$ in an allegory \mathbf{A} is a function arrow iff*

$$id_X \subseteq m;m^\circ \text{ and } m^\circ;m \subseteq id_Y.$$

In particular, note that for any allegory \mathbf{A} , its function arrows will include its identity arrows.

Definition 2.3. *Given an allegory \mathbf{A} , $\mathbf{Fun}(\mathbf{A})$ is defined to be the subcategory of \mathbf{A} consisting of the objects and function arrows of \mathbf{A} .*

For example, the function arrows of the allegory **Rel** are simply those relations which are also total functions. Thus we have $\mathbf{Fun}(\mathbf{Rel}) = \mathbf{Set}$.

2.2 Division Allegories

There are several varieties of allegory given in [17]. A *distributive allegory* has the property that the collection of all arrows with the same source and target type form a distributive lattice [20]:

Definition 2.4. A *distributive allegory* is an allegory together with two operators: a nullary operator \emptyset and a binary operator \cup (called join). For every pair of objects, there is an arrow $\emptyset_{X,Y} : X \rightarrow Y$ such that for every arrow $r : X \rightarrow Y$, $\emptyset_{X,Y} \subseteq r$ and $r ; \emptyset_{Y,Z} = \emptyset_{X,Z}$. Also, for every pair of arrows $r, s : X \rightarrow Y$ there is an arrow $r \cup s : X \rightarrow Y$, which is characterised by the following universal property for all arrows $q : X \rightarrow Y$:

$$r \cup s \subseteq q \equiv (r \subseteq q) \wedge (s \subseteq q)$$

and which satisfies the following laws for all $p : W \rightarrow X$ and $q, r, s : X \rightarrow Y$:

$$\begin{aligned} p ; (r \cup s) &= p ; r \cup p ; s \\ q \cap (r \cup s) &= (q \cap r) \cup (q \cap s) \end{aligned}$$

In **Rel**, \emptyset returns the empty relation, and join is relational union.

Definition 2.5. A *division allegory* is a distributive allegory with an additional binary division operator \setminus , such that for each pair of arrows with common source $r : Z \rightarrow Y$ and $s : Z \rightarrow X$, $s \setminus r : X \rightarrow Y$ is defined by the universal property

$$t \subseteq s \setminus r \equiv s ; t \subseteq r$$

The interpretation of $s \setminus r$ in **Rel** is $x (s \setminus r) y \equiv \forall z : z s x \Rightarrow z r y$.

This division operator is especially useful for specifications that involve universal quantification. This operator is sometimes known as *left division* [12], or *weakest postspecification* [5]. There is a dual right division operator; we use left division because we will use it to define multirelations in Section 3.

2.3 Power Allegories

A power allegory [17] is a division allegory that contains an analogue of the familiar notion of a powerset. Specifically,

Definition 2.6. A *division allegory* **A** is a power allegory if there is

- for each object X , an object $\mathbf{P}X$ called the power-object of X
- a power transpose Λ that for each arrow $r : X \rightarrow Y$ returns a function arrow $\Lambda r : X \rightarrow \mathbf{P}Y$
- a membership arrow $\ni_x : \mathbf{P}X \rightarrow X$,

These three things are defined up to isomorphism by the universal property:

$$(f = \Lambda r) \equiv (f ; \ni = r) \tag{2.1}$$

The converse of \ni (the subscript X will usually be omitted for brevity) is denoted by \in and in **Rel** this represents the familiar set membership relation, with the power-object being the powerset. The power transpose in **Rel** is the function that defines the isomorphism between relations and set-valued functions via the universal property (2.1). The following functor is also useful:

Definition 2.7. *The existential image functor $E : \mathbf{A} \rightarrow \mathbf{Fun}(\mathbf{A})$ is defined as:*

$$Er \hat{=} \Lambda(\ni; r)$$

3 Multirelations

Set Theoretic Definition

In **Rel**, a multirelation is a relation with a powerset target, and a multirelation $r : X \rightarrow \mathbb{P}Y$ is said to be *up-closed* if for all $x \in X$ and $U, V \subseteq Y$

$$x r U \wedge U \subseteq V \Rightarrow x r V$$

Specifications can be modelled by up-closed multirelations [1], as follows.

Interpretation

A multirelation r relates a value x to a set U if and only if, given input x , the angel can ensure that the program r will terminate with an output value that satisfies postcondition U , where the set U is interpreted as a predicate. Of course, if an output value satisfies U and $U \subseteq V$, then it must also satisfy V , hence the up-closure property. This interpretation of multirelations thus relates input values to guarantees (predicates) on the output, one of which is chosen by the angel. The actual output value is chosen by the demon amongst values in the angel's chosen guarantee set.

Predicate Transformer Correspondence

The weakest precondition predicate transformer $wp.r$ that corresponds to the multirelation $r : X \rightarrow \mathbb{P}Y$ is given by

$$x \in wp.r.U \equiv (x, U) \in r$$

When multirelations are used to model programs and specifications, the up-closure property corresponds to the monotonicity requirement of predicate transformer models.

3.1 Multirelations in an Allegory

The above set-theoretic definition of a multirelation is useful from an intuitive point of view, but it is difficult to reason about equationally, as was seen in [3], because the associated definition of multirelation composition is unwieldy to calculate with. Therefore we now give a new alternative pointfree definition in the setting of a power allegory, which is easier to reason about.

In allegories, multirelations are modelled as arrows whose targets are power objects, and we have the corresponding definition:

Definition 3.1 (multirelation). An up-closed multirelation with source X and target Y is an arrow $r : X \rightarrow \mathbb{P}Y$ in a power allegory such that

$$r ; \sqsubseteq_Y = r$$

where $\sqsubseteq_Y \hat{=} \in_Y \setminus \in_Y$; the subscript will usually be omitted.

Note that the family of arrows \sqsubseteq coincides with the family of arrows \subseteq in **Rel**. We will abbreviate ‘up-closed multirelation’ by ‘multirelation’, and denote the type of all multirelations with source X and target Y in a power allegory by $X \rightrightarrows Y$. Multirelations cannot be composed using ordinary composition for obvious type reasons. Instead, we have:

Definition 3.2 (multirelation composition). The composition of two multirelations $r : X \rightrightarrows Y$, $s : Y \rightrightarrows Z$, denoted by $r \circledast s : X \rightrightarrows Z$, is defined by

$$r \circledast s \hat{=} r ; (\in \setminus s)$$

Interpreting composition in set-theoretic terms, for all $x : X$ and $V : \mathbb{P}Z$, we have $x (r \circledast s) V \Leftrightarrow (\exists U : x r U : (\forall y : y \in U : y s V))$. So, given input x , the angel can guarantee that $r \circledast s$ will output a value satisfying V when he can ensure that r will establish an intermediate postcondition U such that, given any input value from U , he can guarantee that s will establish V . Whilst this pointwise definition provides useful intuition, laws like the following are more easily deduced from Definition 3.2 in the point-free style:

$$(r ; s) \circledast t = r ; (s \circledast t)$$

Definition 3.3 (multirelation identity). The identity multirelation arrow is \in_X for each object X .

It is the case that multirelations form a category:

Lemma 3.4 (multirelations category). The multirelations in a power allegory \mathbb{A} form an order-enriched category with identity and composition as given in Definitions 3.2 and 3.3, and ordering \subseteq . This category will be referred to as $\mathbf{Mul}(\mathbb{A})$.

However, multirelations do not form a sub-category of the power allegory, because the category composition operator is different to that of the allegory.

4 Multifunctions in an Allegory

Relations can be interpreted as set-valued functions (multifunctions), and the arrows in a power allegory can be represented as function arrows in a similar way. So every multirelation can also be represented as a function, as shown below.

Definition 4.1 (up-closed multifunction). An up-closed multifunction with source X and target Y is a function arrow $p : X \rightarrow \mathbb{P}^2Y$ in a power allegory such that

$$p; \uparrow_Y = p$$

where the up-closure operator $\uparrow_Y \hat{=} \mathbf{E} \sqsubseteq_Y$; the subscript will usually be omitted.

Note that we may abbreviate ‘up-closed multifunction’ by ‘multifunction’.

In **Rel**, \uparrow is the standard up-closure function (e.g. see [20]). That is, for all $W \in \mathbb{P}^2Y$, $\uparrow W = \{V \mid (\exists U : U \in W : U \subseteq V)\}$. So a function $p : X \rightarrow \mathbb{P}^2Y$ is an up-closed multifunction if and only if for all $x \in X$ the set px is up-closed: if $U \in px$ and $U \subseteq V$ then $V \in px$. This is precisely the kind of function used in the *choice* semantics [9]; px produces a set of predicates (guarantees) that the angel can choose from.

Identity and composition for multifunctions are defined as follows:

Definition 4.2 (multifunction identity). The identity multifunction arrow for each type X is denoted by ι_x , where $\iota_x \hat{=} \Lambda \in_x$.

In set theory, the identity multifunction for all $x \in X$ is given by $\iota_x x = \uparrow \{\{x\}\}$.

Definition 4.3 (multifunction composition). The composition of each pair of up-closed multifunctions $p : X \rightarrow \mathbb{P}^2Y$ and $q : Y \rightarrow \mathbb{P}^2Z$ is defined by

$$p \star q \hat{=} p; \mathbf{E}^2q; \mathbf{E} \cap; \cup$$

where $\cap \hat{=} \Lambda(\in \setminus \ni)$ and $\cup \hat{=} \Lambda(\ni \ni)$.

In set theory the arrows \cap and \cup represent generalised intersection and union, so for all $x : X$, the value of $(p \star q)x$ is $\bigcup \{\bigcap \{qw \mid w \in W\} \mid W \in px\}$. Hence given input x , the angel can only ensure that $p \star q$ will establish a postcondition U if he can choose a postcondition W for p (given input x) such that he can ensure that q establishes U for every input value in W . The following two laws would be fiddly to prove using set-theoretic notation, but in this allegorical setting, are immediate from Definition 4.3:

$$(p; q) \star r = p; (q \star r) \tag{4.1}$$

$$(p; \mathbf{E}^2q) \star r = p \star (q; r) \tag{4.2}$$

Multifunctions also form a category within a power allegory:

Lemma 4.4 (multifunctions category). The up-closed multifunctions in a power allegory \mathbf{A} form an order-enriched category $\mathbf{MFun}(\mathbf{A})$, with identity and composition as in Definitions 4.2 & 4.3, where the ordering \leq on the arrows $p, r : X \rightarrow \mathbb{P}^2Y$ is defined by

$$p \leq r \hat{=} p; \ni \subseteq r; \ni$$

The isomorphism between multirelations and multifunctions is stated below:

Lemma 4.5. *There is an order-isomorphism of categories $\mathbf{MFun}(\mathbf{A}) \cong \mathbf{Mul}(\mathbf{A})$ as given by the universal property (2.1) for Δ and \exists .*

There are a number of useful operators for converting between functions, relations, multirelations and multifunctions, but the only one we require here is a lifting operator to convert a function to its multifunction equivalent:

Definition 4.6 (lifting). *Let $f : X \rightarrow Y$ in $\mathbf{Fun}(\mathbf{A})$, for power allegory \mathbf{A} . Then the lifting $\widehat{f} : X \rightarrow \mathbf{P}^2Y$ in $\mathbf{MFun}(\mathbf{A})$ is defined by*

$$\widehat{f} \hat{=} f; \iota$$

The lifting operator is a functor and satisfies many elegant algebraic properties, some of which are listed below for subsequent use:

$$\widehat{f} \star m = f; m \tag{4.3}$$

$$\widehat{f} \star \widehat{g} = \widehat{f; g} \tag{4.4}$$

$$m \star \widehat{f} = m; \mathbf{E}^2 f; \uparrow \tag{4.5}$$

$$\widehat{f}; \exists = f; \in \tag{4.6}$$

To recap, we have now arrived at the point where, given any power allegory \mathbf{A} , we can characterise its multirelation arrows by $\mathbf{Mul}(\mathbf{A})$, and its multifunction arrows by $\mathbf{MFun}(\mathbf{A})$. Furthermore, these two categories arising within \mathbf{A} are order-isomorphic to each other. The semantic models of multirelations and multifunctions are not new, but their presentation in the context of allegories is new. We have defined $\mathbf{MFun}(\mathbf{A})$ because it provides a useful stepping stone to cast $\mathbf{Mul}(\mathbf{A})$ as a Kleisli category of multirelations, in order to construct the monadic maps and folds, as will be seen in the following section.

5 Kleisli categories

This section constructs a Kleisli category order-isomorphic to that of multifunctions, and thus also to that of multirelations. In order to describe what Kleisli categories are, we will need the concept of a monad (e.g. see [19]):

Definition 5.1 (monad). *A monad $\langle \mathbf{M}, \eta, \mu \rangle$ in a category \mathbf{C} consists of a functor $\mathbf{M} : \mathbf{C} \rightarrow \mathbf{C}$ and two natural transformations, the unit $\eta : I_{\mathbf{C}} \rightarrow \mathbf{M}$ and multiplication $\mu : \mathbf{M}^2 \rightarrow \mathbf{M}$ such that*

$$\mathbf{M}\eta; \mu = id = \eta\mathbf{M}; \mu \text{ and } \mu\mathbf{M}; \mu = \mathbf{M}\mu; \mu$$

Every monad can be used to form a Kleisli category in the following way:

Definition 5.2 (Kleisli category). *Given a monad $\langle \mathbf{M}, \eta, \mu \rangle$ in a category \mathbf{C} , the Kleisli category $\mathbf{C}^{\mathbf{M}}$ has the same objects as \mathbf{C} and the arrows from X to Y in $\mathbf{C}^{\mathbf{M}}$ are the arrows of type $X \rightarrow \mathbf{M}Y$ in \mathbf{C} . Each object X has identity arrow η_X , and for each $f : X \rightarrow \mathbf{M}Y$ and $g : Y \rightarrow \mathbf{M}Z$, their Kleisli composition $f;_{\mathbf{M}} g : X \rightarrow \mathbf{M}Z$ is defined by*

$$f;_{\mathbf{M}} g \hat{=} f; \mathbf{M}g; \mu$$

Some laws derivable from the definition of $;\mathbf{m}$ can be found in [21].

Example 5.3. If \mathbf{A} is a power allegory then the triple $\langle \mathbf{P}, \eta, \cup \rangle$ is a monad in $\mathbf{Fun}(\mathbf{A})$, where $\eta = \mathit{Aid}$, $\cup = \mathit{A}(\ni; \ni)$ and $\mathbf{P} = \mathbf{J}; \mathbf{E}$, where \mathbf{J} is the embedding functor $\mathbf{J} : \mathbf{Fun}(\mathbf{A}) \rightarrow \mathbf{A}$.

Translating this to set theory, taking \mathbf{A} to be \mathbf{Rel} , η is the function that maps each element to its singleton set, \cup returns the generalised union of a set of sets, and \mathbf{P} is the functor that maps each object to its powerset and applies each function to all the elements of a set: $\mathbf{P} f X = \{f x \mid x \in X\}$. Thus $\mathbf{Fun}(\mathbf{Rel})^{\mathbf{P}}$, also known as $\mathbf{Set}^{\mathbf{P}}$, is the category of sets and set-valued functions.

Note that the universal property (2.1) defines an isomorphism between the Kleisli category of this monad and the original allegory \mathbf{A} . \square

The example above was a relatively simple illustration of a monad and its corresponding Kleisli category. The monad required to build the Kleisli category corresponding to multifunctions (and hence multirelations) is not so simple.

5.1 Multifunctions as a Kleisli Category

We will approach the construction of multifunctions as a Kleisli category in small steps. Firstly, in $\mathbf{Fun}(\mathbf{A})$ we will need the existence of equalisers:

Definition 5.4 (equaliser). *The equaliser of a pair of arrows $f, g : Y \rightarrow Z$ in a category is an arrow $eq : X \rightarrow Y$ such that for all $h : W \rightarrow Y$ with $h; f = h; g$ there is a unique $k : W \rightarrow X$ with $k; eq = h$. The source object of eq , namely X , is called the object of the equaliser.*

\mathbf{Rel} is an allegory such that $\mathbf{Fun}(\mathbf{Rel})$ has equalisers; there are others, for example every tabular allegory \mathbf{A} has equalisers in $\mathbf{Fun}(\mathbf{A})$ (see [17] for details).

We will need a suitable monad $\langle \mathbf{N}, \eta, \mu \rangle$ to form the Kleisli category for multifunctions in $\mathbf{Fun}(\mathbf{A})$. The definition below of $\mathbf{N} : \mathbf{Fun}(\mathbf{A}) \rightarrow \mathbf{Fun}(\mathbf{A})$ is presented separately on objects and arrows:

Definition 5.5 (action of \mathbf{N} on objects). *Let \mathbf{A} be a power allegory such that $\mathbf{Fun}(\mathbf{A})$ has equalisers. For each object Y in \mathbf{A} , $\mathbf{N}Y$ is defined to be the object of the equaliser of $id_{\mathbb{P}^2Y}$ and \uparrow_Y in $\mathbf{Fun}(\mathbf{A})$.*

Interpreting the above with $\mathbf{A} = \mathbf{Rel}$, $\mathbf{N}Y = \{Z \in \mathbb{P}^2Y \mid Z \text{ is up-closed}\}$.

From Definition 4.1 and the definitions above we can now define the following:

Definition 5.6 (embedding e , projection $'$). *Let \mathbf{A} be a power allegory such that $\mathbf{Fun}(\mathbf{A})$ has equalisers. Then there is an arrow $e_Y : \mathbf{N}Y \rightarrow \mathbb{P}^2Y$ for each object Y in \mathbf{A} , such that e_Y is an up-closed multifunction. Furthermore, for all $p : W \rightarrow \mathbb{P}^2Y$ in $\mathbf{MFun}(\mathbf{A})$ there is a unique $p' : W \rightarrow \mathbf{N}Y$ characterised by the universal property*

$$u = p' \equiv u; e_Y = p$$

We are now ready to complete the definition of \mathbf{N} :

Definition 5.7 (action of \mathbf{N} on arrows). *Let \mathbf{A} be a power allegory such that $\mathbf{Fun}(\mathbf{A})$ has equalisers. For each arrow $f : Y \rightarrow Z$ in $\mathbf{Fun}(\mathbf{A})$,*

$$\mathbf{N} f \hat{=} (e_Y \star \widehat{f})'$$

Interpreting the above definition with $\mathbf{A} = \mathbf{Rel}$ using property (4.5), applying $\mathbf{N}f$ to an up-closed set of sets Z results in $\mathbf{N} f Z = \uparrow \{\{E f z\} \mid z \in Z\}$. This applies f to all the values in the sets belonging to Z and then takes their upclosure.

Lemma 5.8. $\mathbf{N} : \mathbf{Fun}(\mathbf{A}) \rightarrow \mathbf{Fun}(\mathbf{A})$ *as defined above is a functor.*

The functor \mathbf{N} can now be used to form a monad:

Lemma 5.9. *Let \mathbf{A} be a power allegory such that $\mathbf{Fun}(\mathbf{A})$ has equalisers, and let \mathbf{N} be defined as above. For each object Y in \mathbf{A} , let $\eta : I_{\mathbf{Fun}(\mathbf{A})} \rightarrow \mathbf{N}$ and $\mu : \mathbf{N}^2 \rightarrow \mathbf{N}$ be defined by*

$$\begin{aligned} \eta_Y &\hat{=} \iota'_Y, \\ \mu_Y &\hat{=} (e_{\mathbf{N}Y} \star e_Y)'. \end{aligned}$$

Then $\langle \mathbf{N}, \eta, \mu \rangle$ is a monad.

The set-theoretic interpretations of η and μ are as follows: $\eta y = \uparrow \{\{y\}\}$ and $\mu Y = \bigcup \{\bigcap X \mid X \in Y\}$.

The universal property from Definition 5.6 shows that for each object X , there is a one-to-one correspondence between the up-closed multifunctions with target X and the functions with target $\mathbf{N}X$. This can be extended to an order-isomorphism of categories:

Theorem 5.10. *Suppose the monad $\langle \mathbf{N}, \eta, \mu \rangle$ is defined as in Lemma 5.9 and let the ordering on arrows in $\mathbf{Fun}(\mathbf{A})^{\mathbf{N}}$ be defined for all $u, v : W \rightarrow \mathbf{N}Y$ by*

$$u \preceq_{\mathbf{N}} v \quad \equiv \quad u; e_Y; \exists \subseteq v; e_Y; \exists$$

Then there is an order-isomorphism of categories $\mathbf{Fun}(\mathbf{A})^{\mathbf{N}} \cong \mathbf{MFun}(\mathbf{A})$ given by the universal property from Definition 5.6.

Corollary 5.11. *There is an order-isomorphism of categories $\mathbf{Fun}(\mathbf{A})^{\mathbf{N}} \cong \mathbf{Mul}(\mathbf{A})$.*

6 Monadic Extensions

It is known [14, 15] that if $\langle \mathbf{M}, \eta, \mu \rangle$ is a monad in some category \mathbf{C} , then every functor $F : \mathbf{C} \rightarrow \mathbf{C}$ can be extended to $F^{\mathbf{M}} : \mathbf{C}^{\mathbf{M}} \rightarrow \mathbf{C}^{\mathbf{M}}$, which is a functor under certain conditions. In this section we examine the properties of the extension $F^{\mathbf{M}}$ for the monad of Lemma 5.9 for a restricted class of functors. We start by recalling some standard definitions concerning functors and datatypes.

6.1 Functors and Initial Algebras

Definition 6.1 (polynomial functor). A polynomial functor in a category with products and coproducts is one of the following:

- The identity functor Id ,
- A constant functor K_X for some object X ,
- The composition FG , pointwise sum $\text{F} + \text{G}$ or pointwise product $\text{F} \times \text{G}$ of two polynomial functors F and G .

Here the juxtaposition FG denotes F after G , so $(\text{FG})f = \text{F}(\text{G}f)$.

Definition 6.2 (initial algebra). Let $\text{F} : \mathbf{C} \rightarrow \mathbf{C}$ be a functor. By definition, an F -algebra is an arrow of type $\text{FX} \rightarrow X$. An F -algebra $\alpha : \text{FT} \rightarrow T$ is initial if, for each F -algebra $p : \text{FX} \rightarrow X$, there exists a (unique) arrow $([p]) : T \rightarrow X$ that satisfies the equivalence

$$\alpha ; q = \text{F}q ; p \quad \equiv \quad q = ([p])$$

That is, α is an initial object in the category of F -algebras.

Arrows of the form $([p])$ are called *catamorphisms*, and are also known as *folds*.

Example 6.3. Consider the recursively defined datatype of cons-lists over an arbitrary type X :

$$\text{list } X ::= \text{nil} \mid \text{cons}(X, \text{list } X)$$

From this description, we can construct a functor L_X , where $\text{L}_X(Y) = 1 + (X \times Y)$ and $\text{L}_X(f) = \text{id} + (\text{id} \times f)$. If we are working within a category that supports the existence of initial algebras, such as **Set**, then the above definition of *list* X describes the datatype of cons-lists as an initial L_X -algebra that is the coproduct $[\text{nil}, \text{cons}]_X : \text{L}_X(\text{list } X) \rightarrow \text{list } X$. \square

Initial algebras are used to characterise datatypes, but they can also be used to construct functors. Suppose the datatype's functor is written as a bifunctor; for example, writing $\text{L}_X(Y)$ above as $\text{L}(X, Y)$. In general, if F is a bifunctor with initial algebra $\alpha_X : \text{F}(X, \text{T}X) \rightarrow \text{T}X$, then a *type functor* T can be defined, such that $\text{T}Y$ is the target object of α_Y , and for all $f : X \rightarrow Y$

$$\text{T}f \hat{=} ([\text{F}(f, \text{id}_{\text{T}Y}); \alpha_Y]) \tag{6.1}$$

Example 6.4. Given the datatype definition of *list* X above in the category **Set**, equation (6.1) translates to:

$$\begin{aligned} \text{list } f \ [] &= [] \\ \text{list } f \ (x : xs) &= f a : \text{list } f \ xs, \end{aligned}$$

where $[]$ is the empty list constructed by *nil*, and $x : xs$ is an abbreviation for *cons* (x, xs) . Thus the type functor *list* is the well-known *map* operator of functional programming, where *list* f applies f to every element of the list. \square

The class of functors we will consider in the remainder of this section are polynomial functors, type functors, or those obtained via the closure of such operations.

6.2 The Monadic Extension of a Functor

The monadic extension of a functor is defined [14, 15] as follows:

Definition 6.5 (monadic extension). *Let $F : C \rightarrow C$ be a functor, let $\langle M, \eta, \mu \rangle$ be a monad in C , and let $\delta^F : FM \rightarrow MF$. The extension $F^M : C^M \rightarrow C^M$ is defined for any object X and arrow $f : Y \rightarrow Z$ as*

$$\begin{aligned} F^M X &\cong FX \\ F^M f &\cong Ff ; \delta_Z^F \end{aligned}$$

To explain, note that the arrow $f : Y \rightarrow Z$ in C^M is an arrow $f : Y \rightarrow MZ$ in the category C . Applying F to f results in an arrow $Ff : FY \rightarrow FMZ$, but the extension $F^M : C^M \rightarrow C^M$ requires the type of $F^M f$ to be $FY \rightarrow MFZ$. Thus Ff is composed with a suitable $\delta_Z^F : FMZ \rightarrow MFZ$, which obtains an arrow of the correct type.

Naturally, we would like F^M to be a functor, and this requires conditions on δ^F :

Definition 6.6 (lifting). *F^M as defined above is called a lifting when it is a functor in C^M , which is the case if and only if it satisfies the following conditions:*

$$F\eta_x ; \delta_x^F = \eta_{Fx} \tag{6.2}$$

$$F\mu_x ; \delta_x^F = \delta_{Mx}^F ; \mu_x^F \tag{6.3}$$

Note that the above definitions generalise to functors of types $F : C^n \rightarrow C$ in an obvious way, where $C^n = C \times \dots \times C$ (n times), and $\delta^F : FM^n \rightarrow MF$.

The following definition [14, 15] gives an inductive way to define the natural transformation δ^F associated with the lifting of any functor defined using the polynomial and/or type functor constructors. The definition is valid for any monad; the only value that depends on the chosen monad is the choice of ϕ^M in the definition of $\delta^{F \times G}$:

Definition 6.7. *Let $\langle M, \eta, \mu \rangle$ be a monad, then δ is given by*

$$\begin{aligned} \delta_x^{Kx} &\cong \eta_x & \delta_{(x,y)}^{F+G} &\cong [\delta_x^F ; M \text{ inl} , \delta_y^G ; M \text{ inr}] \\ \delta_x^{\text{ld}} &\cong id_{Mx} & \delta_{(x,y)}^{F \times G} &\cong (\delta_x^F \times \delta_y^G) ; \phi^M \\ \delta_x^{FG} &\cong F\delta_x^G ; \delta_{Gx}^F & \delta_x^T &\cong ([\delta_{Tx}^{Hx} ; M\alpha_x]) \end{aligned}$$

Above, inl and inr are the injection arrows of the coproduct. The natural transformation $\phi^M : MX \times MY \rightarrow M(X \times Y)$ is associated with monad M . T is the type functor for a bifunctor H , with initial algebra $\alpha_x : H(X, TX) \rightarrow TX$ and associated functor $H_x(Y) = H(X, Y)$.

It is known [14, 15] that all of the components of the above definition, apart from ϕ^M , are natural transformations that satisfy appropriately typed forms of conditions (6.2) and (6.3). Moreover, if F is defined using any of the constructors used in the above definition, then δ^F is a natural transformation if ϕ^M is one, and properties (6.2) and (6.3) hold for δ^F if they are true for ϕ^M .

6.3 Lax Liftings of Functors on Multirelations

The monadic extension of functors described in the previous section cannot be applied directly to the monad associated with multirelations of Lemma 5.9. Instead, we introduce a weaker definition which we will call a *lax lifting*.

Definition 6.8 (lax lifting). *Let $\langle M, \eta, \mu \rangle$ be a monad in a category \mathcal{C} and suppose that \mathcal{C}^M is enriched with an ordering \preceq . Let $F : \mathcal{C} \rightarrow \mathcal{C}$ be a functor and let δ^F be a family of arrows such that for every object X there is an arrow $\delta_X^F : FMX \rightarrow MFX$, and for all $p : X \rightarrow Y$ in \mathcal{C} ,*

$$FMp; \delta_Y^F \preceq \delta_X^F; MFP \quad (6.4)$$

Suppose further that

$$F\eta_X; \delta_X^F = \eta_{FX} \quad (6.5)$$

$$F\mu_X; \delta_X^F \preceq \delta_{MX}^F; \mu_X^F \quad (6.6)$$

Then we define the lax lifting $F^M : \mathcal{C}^M \rightarrow \mathcal{C}^M$ on objects as $F^M X = FX$ and on arrows $f : Y \rightarrow Z$ in \mathcal{C}^M as $F^M f = Ff; \delta_Z^F$.

Lemma 6.9. *If F^M is a lax lifting then it preserves identities, and for all p, q it is the case that $F^M(p; \mu q) \preceq F^M p; \mu F^M q$.*

If F is defined using constructors from polynomial and/or type functors, then properties (6.4), (6.5) and (6.6) will hold for F provided that they hold for \times , since it is known that all the other constructors are natural transformations that satisfy the stronger analogues (6.2) and (6.3).

It remains to define the transformation ϕ in the Kleisli category associated with multirelations and examine its properties. The definition of ϕ given below requires the existence of *relational products* in the allegory \mathbf{A} , which are constructed using the left and right projections *outl* and *outr* of the product in $\mathbf{Fun}(\mathbf{A})$, provided that it has finite products. These exist if \mathbf{A} is a unitary tabular allegory, for example \mathbf{Rel} (see [12] for details).

$$r \times s \hat{=} (outl; r; outl^\circ) \cap (outr; s; outr^\circ)$$

The relational product is a monotonic bifunctor, but it is not a categorical product. We can now define ϕ^N for multirelations:

Definition 6.10. *Suppose that $\langle N, \eta, \mu \rangle$ is the monad of Lemma 5.9. Using the notation of (5.6), ϕ^N is defined by*

$$\phi_{(X,Y)}^N \hat{=} ((e_X \times e_Y); \Lambda((\exists \times \exists); ((\epsilon \times \epsilon) \setminus \epsilon)))'$$

To understand ϕ^N , it may help to consider the relation $(\ni \times \ni); ((\in \times \in) \setminus \in)$, in a set-theoretic setting:

$$(A, B) ((\ni \times \ni); ((\in \times \in) \setminus \in)) Z \equiv \exists p_A, p_B : p_A \in A \wedge p_B \in B \wedge (\forall a, b : a \in p_A \wedge b \in p_B : (a, b) \in Z)$$

Thus for two sets A, B of postconditions, Z contains (but is not limited to) all pairs of outcomes (a, b) where a satisfies p_A and b satisfies p_B . The rest of the definition of ϕ^N involves Λ , which returns all possible such Z s, and the embeddings and projection, which convert between the appropriate types of arrows and ensure upclosure of the output of ϕ^N .

Lemma 6.11. ϕ^N satisfies conditions (6.4), (6.5) and (6.6).

It follows that if \mathbf{A} is a power allegory then any functor extension defined in $\mathbf{Fun}(\mathbf{A})^N$ using any of the constructors of Definition 6.7 is a lax lifting.

6.4 Monadic Multirelational Maps

It is now possible for us to construct multirelational map operators for datatypes. We shall illustrate how to do this for lists, by constructing a lax lifting of the *list* functor, otherwise known as *map*.

Taking L_X as the base functor for lists as defined in Example 6.3, we have

$$\delta_{list\ X}^{L_X} = [\eta; \mathbf{N}\ iinl, (\eta \times id); \phi^N; \mathbf{N}\ iinr]$$

Then from the definition of δ (Definition 6.7), we obtain

$$\delta_X^{list} = ([\delta_{list\ X}^{L_X}; \mathbf{N}[nil, cons]_X])$$

In \mathbf{Set}^N this definition can be expanded as follows:

$$\delta_X^{list} = (list\ e_x; \Lambda((listr\ \ni); ((listr\ \in) \setminus \in)))'$$

where *list* was defined in Example 6.4, and *listr* is defined for relations as follows:

$$\begin{aligned} [] (listr\ r) [] &\equiv true \\ (x : xs) (listr\ r) (y : ys) &\equiv x\ r\ y \wedge xs\ (listr\ r)\ ys \end{aligned}$$

Having defined δ^{list} , Definition 6.8 can be used to calculate the monadic extension of the *list* functor in \mathbf{Set}^N . Suppose $p : X \rightarrow NY$, then we have

$$\begin{aligned} list^N\ p\ [] &= \uparrow \{ \{ [] \} \} \\ list^N\ p\ (x : xs) &= \{ U \mid \exists Z, V : Z \in p\ x \wedge V \in list^N\ p\ xs \\ &\quad \wedge (\forall w, ws : w \in Z \wedge ws \in V : (w : ws) \in U) \} \end{aligned}$$

This completes the derivation of the *list* (or *map*) functor for \mathbf{Set}^N where \mathbf{N} was defined in Lemma 5.9.

The corresponding definition for $list^N$ applied to an up-closed multifunction $p : X \rightarrow \mathbb{P}^2 Y$ is textually the same as the above, it is only the types that change, and when translated back to multirelations this gives the same definition as in [4]. There may be other choices for ϕ^N that generate alternative definitions for $list^N$ but it is beyond the scope of this paper to examine these.

The above illustrated a multirelational map operator for lists; it is of course possible to obtain multirelational map operators for other datatypes in a similar way, by using the construction functors for those datatypes in place of the $list$ functor.

6.5 Monadic Multirelational Folds

Folds in the Kleisli category can be derived from folds in the base category. This definition is the same as that in [14], except that the functor extension is replaced by a lax lifting:

Lemma 6.12. *Let $\langle M, \eta, \mu \rangle$ be a monad in category \mathcal{C} . Then if $F : \mathcal{C} \rightarrow \mathcal{C}$ has initial algebra α then the lax lifting $F^M : \mathcal{C}^M \rightarrow \mathcal{C}^M$ has initial algebra $\alpha ; \eta$. If $\psi : FX \rightarrow X$ in \mathcal{C}^M , then the fold in \mathcal{C}^M is given by $(\psi)_M = ([\delta_X^F ;_M \psi])$.*

Now the above lemma can be used to find the $foldr$ operator on lists in a Kleisli category, since $foldr$ is defined as the fold associated with the base functor for lists L_X of Example 6.3. Let $\langle N, \eta, \mu \rangle$ be the monad of Lemma 5.9 and let $f : L_X(Y) \rightarrow Y$ in \mathbf{Set}^N . If we let $g \doteq \delta_Y^{L_X} ;_N f$, then by Lemma 6.12, $(f)_N = ([g])$ and by Definition 6.2

$$[nil, cons]_X ; ([g]) = (1 + id \times ([g])) ; g$$

In a set-theoretic setting, this definition can be expanded to the point level using the definition from Section 6.4 and Kleisli composition to give the following (where the arrow $f : L_X(Y) \rightarrow Y$ in \mathbf{Set}^N has been replaced by two components p and a): Let $p : X \times Y \rightarrow NY$ and $a : NY$ then

$$\begin{aligned} foldr p a [] &= a \\ foldr p a (x : xs) &= \bigcup \{ \bigcap \{ p x y \mid y \in Y \} \mid Y \in foldr p a xs \} \end{aligned}$$

Intuitively, Y is a postcondition established from the fold so far on xs , y is an outcome satisfying Y , and $p x y$ is the (upclosed) set of possible postconditions after performing this step p of the fold on intermediate result y with the list element x . Thus the set $\bigcap \{ p x y \mid y \in Y \}$ contains postconditions that can be guaranteed whatever the choice of y satisfying Y , and the \bigcup ensures the inclusion of all possible such guaranteed postconditions, for all choices of Y .

It follows from the isomorphism between the category of multifunctions and the Kleisli category that the definition of $foldr$ for multifunctions is textually the same as the above, if $p : X \rightarrow \mathbb{P}^2 Y$ is a multifunction and $a : \mathbb{P}^2 Y$ is an up-closed

set. This multirelational definition of *foldr* is identical to that constructed by the alternative method of [4].

The above illustrated a multirelational fold operator for lists; similarly, the use of other functors to construct datatypes results in multirelational folds over more general datatypes.

It is interesting to analyse the relationship between maps and folds since in functional and relational programming algebras, it is the case that the map functor is an instance of a fold. The following lemma gives a lax analogue of some results of [14] and shows that for type functors, map is an instance of a fold in the base category but is only related to the monadic fold by an inequation:

Lemma 6.13. *Let \mathbb{T} be the type functor associated with base bifunctor \mathbb{F} with initial algebra $\alpha_X : \mathbb{F}(X, \mathbb{T}X) \rightarrow \mathbb{T}X$ and let \mathbb{F}^M be a lax lifting in the category \mathbb{C}^M which is order-enriched with \preceq , then if $f : X \rightarrow Y$ in \mathbb{C}^M*

$$\begin{aligned} \mathbb{T}^M f &= (\mathbb{F}(f, id_{\mathbb{M}\mathbb{T}Y}); \delta_{(Y, \mathbb{T}Y)}^{\mathbb{F}}; \mathbb{M}\alpha_Y) \\ \mathbb{T}^M f &\preceq (\mathbb{F}^M(f, \eta_{\mathbb{T}Y});_{\mathbb{M}}(\alpha_Y; \eta_{\mathbb{T}Y}))_{\mathbb{M}} \end{aligned}$$

6.6 Unfolds

The technique of extending folds as described in the previous section has been dualised to unfolds in [15] and [16], but unfortunately the methods used there do not apply here. The problem with [15] is that it only applies to a strong monad, and the monad described in Section 5 is not strong. The more recent method of [16] fails on two counts. First, it requires a strictness property that does not generally hold for multifunctions: it is not generally the case that $p;_{\mathbb{M}} abort = abort$ where $abort = (\lambda \emptyset)'$ and as a result the category of multifunctions does not have a terminal object. The other problem is that the extension of the *list* functor is not itself a functor, it is only a lax functor. So the search for the definition of unfold continues. It is hoped that one of these existing methods could be generalised to produce a lax unfold of some kind, but it is beyond the scope of this paper to construct such a definition.

7 Conclusions

Much of the material presented above is standard theory, included in order to provide sufficient background information to make it possible to present this paper's contribution of monadic multirelational maps and folds. To be precise, the definitions of allegories in Section 2 are standard [17], as are the definitions of Kleisli categories in Section 5 [19] and monadic extensions in Section 6.1 [14]. Moreover, the multifunctional model of Section 4 was previously discovered by Back and von Wright [9], the associated monad has been described by Dawson [22] and even the multirelational definitions of map and fold have been documented elsewhere [4].

However, the particular contributions of this paper are several. Firstly, looking from an overall perspective, this paper draws together these varied strands of work from within different semantic models into the setting of power allegories, thus contributing to unification work between different models of program semantics. In particular, through the isomorphism between up-closed multirelations and monotonic predicate transformers [8], multirelations have strong links to UTP [5] and predicate transformer semantics. Thus being able to provide algebraic semantics for multirelations, including all the algebraic definitions and datatypes and operators and laws that make it possible to reason about multirelations to use them for calculational program development, is a very important link.

Secondly, looking at the details of the work in this paper, this construction of maps and folds for multirelations using a monadic extension is also new. Whilst it may seem futile to construct new definitions of map and fold for multirelations when such operators already exist, this work should not be seen as an end in itself, but rather as a firm foundation from which to explore the existence of other operators and algebraic laws, in particular concerning the potentially useful unfold. This paper has given a rigorous treatment of monadic extensions for multirelations, which has resulted in a weakened version of the standard theory. The definition given here was chosen to be compatible with previous work [4], but the monadic extension is not unique, and preliminary investigations suggest that there are other definitions that may also yield useful operators. Through this work, it has been possible to see precisely why previous methods for dualising monadic catamorphisms fail here, but it remains to be seen whether these methods might also be weakened in some way that could be applied to multirelations.

Finally, this paper illustrates the use of point-free definitions and calculation for multirelations. This results in elegant definitions and a style of reasoning more transparent than that used in traditional set-theoretic proofs. In particular, the set-theoretic definition of composition is so cumbersome that it renders calculations at best error-prone and at worst virtually impossible. The point-free alternative is much more concise and manageable, and any doubts of its value can easily be dispelled by attempting any of the proofs using the set-theoretic definitions instead.

To summarise, this paper contains a contribution to the development of the algebra of multirelations, presenting a different view of maps and folds from that described in [4], and this has resulted in a weakening of the standard monadic extension. Future work includes establishing whether an unfold operator can be defined, and this would further contribute to unification of the functional, relational and multirelational calculi.

Acknowledgements

We would like to thank the anonymous referees for their helpful comments.

Appendix

To keep within space constraints, and also because pages of calculational proofs can be tedious, we include one sample proof to illustrate the point-free calculational style. The full proofs for all results contributed by this paper can be found online in a supplement for this paper [21].

Proof. of Lemma 5.8 To show that \mathbf{N} is a functor, we need to demonstrate that it preserves identities and distributes through composition. For identity preservation, we have:

$$\begin{aligned}
& \mathbf{N} id \\
&= \{\text{Action of } \mathbf{N} \text{ on arrows (Definition 5.7)}\} \\
& \quad (e_Y \star \widehat{id})' \\
&= \{\text{Definition (4.6)}\} \\
& \quad (e_Y \star \iota)' \\
&= \{\text{Identity of } \star\} \\
& \quad e'_Y \\
&= \{(5.6)\} \\
& \quad id
\end{aligned}$$

As for composition, first note that by Definition 5.6, every up-closed multifunction $p : X \rightarrow \mathbf{P}^2 Y$ has the property that $p' ; e = p$, and so using equation (4.1) we can see that for all $q : Y \rightarrow \mathbf{P}^2 Z$,

$$p' ; (e_Y \star q)' ; e_Z = p \star q \tag{A.1}$$

Definition 5.6 also implies that for any $p, q : X \rightarrow \mathbf{N} Y$,

$$(p = q) \equiv (p ; e_Y = q ; e_Y) \tag{A.2}$$

So now we can calculate

$$\begin{aligned}
& \mathbf{N} f ; \mathbf{N} g ; e_Z \\
&= \{\text{Action of } \mathbf{N} \text{ on arrows (Definition 5.7)}\} \\
& \quad (e_X \star \widehat{f})' ; (e_Y \star \widehat{g})' ; e_Z \\
&= \{\text{Equation (A.1)}\} \\
& \quad e_X \star \widehat{f} \star \widehat{g} \\
&= \{\text{Equation (4.4)}\} \\
& \quad e_X \star (\widehat{f ; g}) \\
&= \{\text{Actions of } \mathbf{N} \text{ (Definitions 5.5 and 5.7)}\} \\
& \quad \mathbf{N}(f ; g) ; e_Z
\end{aligned}$$

which establishes the result by equation (A.2). □

References

1. Rewitzky, I.: Binary multirelations. In H. de Swart, E. Orlowska, G.S., Roubens, M., eds.: *Theory and Application of Relational Structures as Knowledge Instruments*. Volume 2929. (2003) 259–274
2. Dunne, S.: Chorus angelorum. In: *B 2007: Formal Specification and Development in B*. Volume 4355 of *Lecture Notes in Computer Science*. (2007) 19–33
3. Martin, C.E., Curtis, S.A., Rewitzky, I.: Modelling angelic and demonic nondeterminism with multirelations. *Science of Computer Programming* **65**(2) (2007) 140–158
4. Martin, C.E., Curtis, S.A.: Nondeterministic folds. In Uustalu, T., ed.: *Proceedings of the 8th Mathematics of Program Construction conference*. Volume 4014 of *Lecture Notes in Computer Science*. (2006) 274–298
5. Hoare, C.A.R., He, Jifeng: *Unifying Theories of Programming*. Prentice Hall (1998)
6. Cavalcanti, A., Woodcock, J.C.P., Dunne, S.E.: Angelic nondeterminism in the unifying theories of programming. *Formal Aspects of Computing* **18**(3) (2006) 288–307
7. Cavalcanti, A., Woodcock, J.C.P.: Angelic nondeterminism and unifying theories of programming. In Derrick, J., Boiten, E., eds.: *REFINE 2005*. Volume 137., Elsevier (2005)
8. Hesselink, W.: Multirelations are predicate transformers. Technical Report, Dept. of Computing Science, Groningen, The Netherlands (2004)
9. Back, R., Wright, J.: *Refinement Calculus: A Systematic Introduction*. Springer-Verlag New York (1998)
10. Morgan, C.C.: *Programming from Specifications*. 2nd edn. Prentice-Hall (1994)
11. Bird, R.: *An Introduction to Functional Programming using Haskell*. Prentice Hall (1998)
12. Bird, R., Moor, O.: *The Algebra of Programming*. Prentice Hall (1997)
13. Gibbons, J., Jones, G.: The under-appreciated unfold. In: *Proceedings 3rd ACM SIGPLAN Int. Conf. on Functional Programming, ICFP'98*, Baltimore, MD, USA, 26–29 Sept. 1998. Volume 34(1). ACM Press, New York (1998) 273–279
14. Fokkinga, M.M.: Monadic maps and folds for arbitrary datatypes. *Memoranda Informatica* 94-28 (1994)
15. Pardo, A.: Monadic corecursion - definition, fusion laws, and applications. *Electronic Notes in Theoretical Computer Science* **11** (1998) 105–139
16. Hasuo, I., Jacobs, B., Sokolova, A.: Generic trace theory. *Electronic Notes in Theoretical Computer Science* **164**(1) (2006) 47–65
17. Freyd, P., Ščedrov, A.: *Categories, Allegories*. Springer Verlag (1993)
18. Martin, C.E., Curtis, S.A.: *The algebra of multirelations* (2009) (in preparation).
19. MacLane, S.: *Categories for the Working Mathematician*. 2nd edn. Springer (1998)
20. Davey, B.A., Priestley, H.A.: *Introduction to Lattices and Order*. 2nd edn. Cambridge University Press (2002)
21. Martin, C.E., Curtis, S.A.: Supplement to this paper (2009) <http://cms.brookes.ac.uk/staff/SharonCurtis/publications/mf-supp.pdf>.
22. Dawson, J.E.: Compound monads in specification languages. In: *Proceedings of Programming Languages meets Program Verification (PLPV) 2007*. ACM (2007) 3–10