Acknowledgements

Many people supported me in preparing this thesis, and their help is gratefully acknowledged. I would like to mention:

My Ph.D. supervisor Roland Backhouse, first, for accepting my as an “afstudeer”-student and later on offering me the possibility of doing research in the challenging area of Mathematics of Program Construction. I want to thank him for his guidance, his patience and the effort he put in improving preliminary versions of this thesis. I owe him a lot.

Oege de Moor, for his pleasant cooperation on the field of membership and fans. Although we communicated almost exclusively via email it was as if we were sharing an office and working side-by-side. I enjoyed it a lot.

My fellow Ph.D. student Henk Doornbos for many fruitful discussions and his willingness to serve as a “klank-bord”.

The other members and former members of the Friday morning club: Chritiene Aarts, Eerke Boiten, Joop van den Eijnde, Netty van Gasteren, Rik van Geldrop, Frans Rietman, Ed Voermans, and Jaap van der Woude.

The external members of the Ph.D. committee: Richard Bird, Wim Hesselink, and Lambert Meertens, for reviewing this thesis and giving a lot of detailed and useful comments.

Richard Verhoeven for providing assistance with MathSpad and LaTeX.

My family and friends, for their love and friendship.

To my parents, Ad and Magda.

Contents

Chapter 1

Aims and Motivation

1.1 Generic programming

The attribute “generic” —which means according to a dictionary definition “characteristic of a genus or class; applied to (any individual of) a large class; general, not specific or special”— is without doubt a sine qua non for computer software. The computer is, after all, a general-purpose device and its most successful applications are to software systems which because of their broad applicability can capture a huge market but which can be customized to the needs of specific clients. Examples of such systems are workflow management systems, logistic systems and systems for financial administration. Indeed, Simonyi [42] has argued that the cycle of abstraction and customization is the key to success in the computing market.

The macro economics of big business is of course not the concern of this thesis. Our concern is at the micro level of programming. But even at this level the ability to write so-called “generic” code capturing commonly occurring patterns is vital to reusability and thus to programmer productivity. Many programming languages, or programming methodologies, claim to support some kind of genericity. Examples are overloading, the generic class of ADA, polymorphism in functional languages, and inheritance in object oriented programming.

Reusability is not the only reason why generic programs are important. Another is the conceptual unification that they offer. As remarked by Meertens [35] “Which is more exciting: to find yet another algorithm, or to discover that two familiar algorithms are instances of one more abstract algorithm? It is the latter that sparks new insights and opens the way for finding further connections, that makes it possible to organize and systematize our knowledge and eventually set as routine exercises problems that once were feats of scientific discovery”.

Indeed, there are many additional reasons. Two that we regard as particularly important are the increased concision and precision afforded by a higher degree of abstraction, and the reduction of the burden of proof gained by eliminating the need to repeat essentially the same argument for many special cases.

Genericity is generally accepted to be important; on the other hand it is extremely difficult to be objective about measures of genericity. Like many buzz words the term “generic” can be applied with impunity to almost everything. The goal of this thesis is to contribute to the development of a mathematical theory of generic programming in which verifiable criteria are given for the notion of genericity.

1.2 Theorems for free

The viability of a generic theory of datatypes was made plausible by the pioneering work of Plotkin [41] and Reynolds [43] on the semantics of polymorphism, the importance of which was highlighted by Wadler [52] in a paper entitled "Theorems for free". Since "theorems for free" was the catalyst for our own work we use this section to briefly summarise developments in functional programming both preceding and up to the writing of this thesis. In the next section we present an overview of the thesis. We begin with a discussion of polymorphism in functional programming.

1.2.1 Polymorphism

Of the instances of genericity in current programming languages/paradigms mentioned above, one which has a relatively well-developed theoretical basis is the notion of parametric polymorphism first introduced by Strachey [48] and later incorporated in the language ML by Milner [37, 38] (and since then a more or less standard feature of all functional programming languages). The use of parametric polymorphism is one of the major success stories of functional programming since it eliminates the compulsion in languages like Pascal to provide irrelevant type information. For example, it is irrelevant to the computation of the length of a list whether the elements of list are integers, characters, or whatever. In Pascal this information must be supplied, thus enforcing the programmer to write essentially the same code each time a length function is required for a new element type.

1.2.2 Polytypy

It has long been observed that data structure influences program structure. For this reason the study of datatypes is an important issue. Bird has developed the theory of lists [8], a set of algebraic transformation rules involving operators like map, reduce, filter etc. Meertens calls this kind of program methodology "algorithmics". [34].

The theory of lists is polymorphic. For instance the map operator is defined irrespective of the type of its argument, i.e. the type of the function that is being mapped over a list. However, the theory was restricted only to lists and thus not applicable to arbitrary datatypes. The theory of lists has been extended by Malcolm [31, 32, 33] to arbitrary datatypes. For instance, he formulated a theorem expressing when two computations could be fused into one computation. Malcolm's fusion theorem was "polytypic" — that is the name given to it nowadays, Malcolm did not use this term — in that it was parameterized by a datatype constructor and so could be instantiated in a variety of ways. So, "polytypic" programs distinguish themselves from polymorphic programs in that the parameter is a datatype constructor like "list" or "tree" — a function from types to types — rather than a type like "boolean", "integer" or "list of integers". One could say that Malcolm developed a "theory of F's" with F being a parameter standing for an arbitrary datatype constructor, as a generalisation of the theory of lists. Malcolm exploited the — polytypic — notion of a "catamorphism" and introduced the "banana bracket" notation which was popularised and extended to the — polytypic — notions of "anamorphism" and "hylomorphism" by Fokkinga, Meijer and Paterson [36]. Since then the theme of polytypy has been explored in a variety ways. Several authors [6, 27, 35] have explored polytypic generalisations of existing programming problems, Doornbos [12, 13, 14] has developed a polytypic theory of program termination and the recently published book by Bird and De Moor [7] contains a wealth of material in which parameterisation by a datatype constructor plays a central rôle.

Functional programmers have a good intuitive understanding of what it means for a function to be polymorphic. Being able to experiment with the notion by writing and executing polymorphic programs is clearly enormously beneficial to understanding. Nevertheless, an unequivocal formal semantics of "parametric polymorphism" is still an active area of research [15]. The situation with regard to polytypy is worse, the emphasis at the current time being on demonstrating the practicality of the notion [27, 22] with the potential danger of unnecessary complication. *Parametric* polymorphism is a well-defined concept that provides the basis for a well-defined theory of *parametric* polytypy.

1.2.3 Parametric polymorphism

A polymorphic function is parametric if its behaviour does not depend on the type at which it is instantiated [48]. Reynolds [43] showed for a specific language that any parametrically polymorphic function instantiated at different types behaves in "related" ways. A polymorphic function satisfies a certain (di)naturality property that is derivable from the type of the function. For instance, for the polymorphic reverse function *rev* on lists it follows that, for $f : A \leftarrow B$,

$$map(f) \cdot rev_B = rev_A \cdot map(f)$$

where $map(f)$ denotes the function on lists which maps function f to each element of the list. Wadler [52] calls such a property a "theorem for free"; a property derived for a function by examining its type alone.

Note that Reynolds' abstraction theorem is a property of a particular language. For instance, it predicts the following property for a parametrically polymorphic equality function $=_A : Bool \leftarrow A \times A$. For all functions f,

$$x = y \quad \equiv \quad f(x) = f(y) \ .$$

In other words, if Reynolds' theorem holds for a given language then all functions are injective in that language or no polymorphic equality function can exist. Indeed, for some languages it is possible to define a polymorphic equality function using ad hoc polymorphism, i.e. including a clause for every type. Such a polymorphic equality function is then not truly polymorphic, i.e. not parametrically polymorphic.

So, although Reynolds' abstraction theorem does not hold for all languages, the notion of parametric polymorphism provides us with a *verifiable* way of checking whether a function is truly parametrically polymorphic. In other words, we consider a function to be truly parametrically polymorphic if the "free theorem", i.e. the (di)naturality property, predicted by its type holds.

1.2.4 Parametric polytypy

Recall that a polytypical function is a parameterised function in which the parameter is a datatype constructor. So, a logical question is: what does a "theorem for free" look like on this higher-level? And is the work of Reynolds applicable on this level? The answer to the second question is negative since we want to consider languages in which it is possible to define higher-order parametric functions inductively over the class of type constructors. The answer to the first question is that in most cases — at least the cases considered in this thesis — we can construct a property which one could call the "free" theorem of a polytypical function. Such a property is a higher-order naturality property expressing that a polytypical function instantiated at different type constructors behaves in related ways.

So, although we can not assume Reynolds' abstraction theorem on a higher-level, Reynolds' ideas, suitably generalized, provide us a *verifiable* way of checking whether a function is truly higher-order parametric or not. In other words, we consider a function to be truly polytypic if the higher-order "free theorem" i.e. a higher-order naturality property, predicted by its type holds.

The work of Jeuring and Jansson [27, 21, 22] differs in this respect from our approach since they consider a polytypic function to be a function which is defined by induction on the structure of the class of type constructors. So, although such a function has a type constructor as a parameter, they do not require – at least not explicitly — that the definition of a polytypic function for each of the different clauses behaves in related ways. In other words, one could argue that Jeuring and Jansson's definition of polytypy is "ad hoc" polytypy. Just like ad hoc polymorphism, the behaviour of a polytypic function instantiated at different types may not be related at all. In other words, it is not necessarily the case that the functions which Jeuring and Jansson define are truly higher-order parametric.

Also related to our work, is the theory of shape developed by Jay [24, 26, 25]. In the theory of shape, a data structure is separated into its shape and its contents.

1.3 Our contribution

In this thesis we try to formulate higher-order parametric characterizations of generic notions and programs instead of giving inductive definitions every time. The main criterion we use to check if a function is truly polytypic is whether the "free theorem" predicted by type considerations holds. An advantage of giving a generic characterization instead of an inductive definition is that if one wants to calculate with these notions and/or programs one can use the characterization directly instead of being forced to give cumbersome proofs inductively over the structure of the class of type constructors every time. However, for the final implementation of a higher-order parametric function we do need an inductive definition of the function on the class of type constructors. Using the polytypic characterization of a program, we can derive a class of programs which is defined by induction on the class of type constructors.

For instance, in chapter 5 the notion of higher-order naturality plays a central rôle in the construction of the polytypic function that commutes two datatypes. We give a non-inductive characterisation of this polytypic function by requiring the higher-order naturality property, i.e. its "free theorem", which we derive from type considerations.

Normally, the type of a *polymorphic* function does not imply uniqueness. Many different polymorphic functions of the same type can exist. However, in the case of *polytypic* functions we get some uniqueness results by investigating the higher-order naturality requirement. These uniqueness results we use for the construction of the polytypic function that commutes two datatypes.

1.4 Overview of the thesis

Chapter 4 reports on joint work with Oege de Moor [20]. We introduce the notion of a membership test for a datatype. We give a non-inductive characterisation of a membership test for an arbitrary datatype and show that it is a "good" characterisation in the sense that for a datatype there is at most one membership test. A new result, compared with [20], is that we give a more general definition for membership than given in [20]; we include the so-called non-endo datatypes as well. For this generalisation, we use the τ-Δ calculus as introduced in the second half of chapter 3. After that, we give the inductive definition of a membership test for the inductively defined class of the so-called regular datatypes.

In the second half of chapter 4, we introduce the notion of a so-called fan for a datatype. Again, we give a non-recursive characterisation of a fan for all datatypes. The existence of membership implies the existence of a unique fan. One can view a fan as the counterpart of membership: with a fan, one can construct a data structure. We give an inductive definition of the fans of the regular datatypes.

In chapter 5 we illustrate the use of the generic notions introduced so far to tackle the problem of "commuting datatypes". We formulate a characterisation of when two datatypes commute. This is done by giving a characterisation of a so-called zip operator which commutes the two datatypes. This characterisation is generic in the sense that the two datatypes are parameters of the program. As an example of the use of the theory of commuting datatypes, we show that a special class of zips, the so-called broadcast operations, are strengths. Furthermore, we tackle the problem of structure multiplication. After that, we give an inductive definition for the zip operations for the class of regular datatypes.

In chapter 6 we continue the investigation begun by Oege de Moor [20] of the relationship between fans and so-called strengths and copies maps of a datatype. Furthermore, we investigate the "free theorem" of the membership test and fans.

The formal setting in which we conduct our research is introduced in chapter 2. The setup of the theory is based on the book of Bird and de Moor [7] although there are differences in detail which we thought were necessary. Readers familiar with that book may skip chapter 2 on first reading except for subsection 2.2.3 and section 2.3. Furthermore, there are some notational differences for the split (fork) and junc (case). Chapter 2 is divided into two parts. First, in order to make this thesis as self-contained as possible, we introduce those elements of category theory that are needed. The category of total functions between sets will be the leading example with which we motivate the concepts we introduce. The most important notion is that of a functor. We argue that functors model datatypes. We define the functors corresponding with product (pairing), coproduct (disjoint union) and recursive datatypes. Connected with functors is the notion of a natural transformation. A natural transformation is a datatype transformer: it takes a datatype and transforms it into another. The term "natural" reflects the fact that such a transformation may not alter the value of the elements that the datatype carries, it is only a transformation of the structure itself.

In the second part of chapter 2, we introduce the notion of a so-called allegory: a point-free axiomatization of the structure of binary relations between sets. The functions, i.e. total and single-valued relations, form a sub-category of an allegory. We extend the notions introduced in the first half to relations. A relational extension of a categorical concept is something which is defined on all relations but if one takes the restriction to functions, one gets the original categorical concept for the sub-category of total functions. The relational extension of a functor is a so-called relator. Not all functors have a relational extension. However, we regard having a relational extension as a healthiness condition on a functor modelling a datatype. In other words, not all functors correspond to a datatype.

Chapter 3 is divided into two parts. In the first part we define the so-called "slok" category. This category plays an important rôle in chapter 5 on commuting relators. In the second half, we introduce the τ-Δ calculus. The τ-Δ calculus is used in the remainder of the thesis in order to reason about non-endo relators. A non-endo relator takes a vector of arguments and/or gives as result a vector of arguments.

In appendix A we have included a list of the typing rules for the τ-Δ calculus, membership, fans, and zips.

Chapter 2

Basic Notions

The formal setting in which we conduct our research is introduced in in this chapter. The setup of the theory is based on the book by Bird and De Moor [7] although there are differences in detail which we thought were necessary. Readers familiar with that book may skip this chapter on first reading except for subsection 2.2.3 and section 2.3. Furthermore, there are some notational differences for the split (fork) and junc (case).

In order to make this thesis as self-contained as possible, we introduce in section 2.1 those elements of category theory that are needed. In section 2.4 we introduce the notion of an allegory: a point-free axiomatization of the structure of binary relations between sets. The functions, i.e. total and single-valued relations, form a sub-category of an allegory. We extend the notions introduced in section 2.1 to relations. A relational extension of a categorical concept is something which is defined on all relations but if one takes the restriction to the total functions, one gets the original categorical concept defined for the sub-category of total functions.

2.1 Categories

A *category* consists of a class of *objects* and a class of *arrows*. Every arrow f comes equipped with two unique objects, its so-called *source* and *target*, denoted by $f\triangleright$ and $f\triangleleft$, respectively. We write $f : A \leftarrow B$ if $f\triangleright = B$ and $f\triangleleft = A$. For every pair of arrows $f : A \leftarrow B$ and $g : B \leftarrow C$ with matching target and source, their *composition* exists: $f \cdot g : A \leftarrow C$. We require composition to be associative, that is, for $f : A \leftarrow B$, $g : B \leftarrow C$ and $h : C \leftarrow D$,

$$(f \cdot g) \cdot h = f \cdot (g \cdot h) \ .$$

Furthermore, for every object A there exists a so-called *identity arrow* id_A which is the unit of composition. That is to say, for each $f : A \leftarrow B$,

$$\mathsf{id}_A \cdot f = f = f \cdot \mathsf{id}_B \ .$$

Functions and relations as categories

Two important examples of categories we encounter in this thesis are the category Map, the category of total functions between sets, and category Rel, the category of binary relations between sets.

In category Map, the objects are sets and an arrow is a triple $(A\,,f\,,B)$ of two sets A and B, target and source of the arrow, and a function f. For such a triple we require that set A contains the range of function f and set B is the domain of f. The identity arrow on A is the triple $(A\,,\mathsf{id}_A\,,A)$ where id_A is the identity function on the set A. The composition of $(A\,,f\,,B)$ and $(B\,,g\,,C)$ is $(A\,,f\cdot g\,,C)$ where $f\cdot g$ is the usual composition of functions: $(f\cdot g)x = f(g(x))$.

In category Rel, the objects are again sets, and an arrow is a triple of three sets $(A\,,R\,,B)$ where R is a subset of the cartesian product $A\times B$. So R is a set of pairs. We write $x\ R\ y$ if $(x\,,y)\in R$. The relation id_A is the identity arrow on A, i.e. $x\ \mathsf{id}_A\ x$ iff $x\in A$. The composition of $(A\,,R\,,B)$ and $(B\,,S\,,C)$ is $(A\,,R\cdot S\,,C)$ where $R\cdot S$ is the usual composition of binary relations i.e. $a\ (R\cdot S)\ c$ iff $\exists(b :: a\ R\ b \wedge b\ S\ c)$.

In the next section we introduce some additional structures on categories. Although many kinds of categories exist, the category Map will be the leading example with which we motivate the concepts we introduce.

Note that although binary relations between sets are indeed a category much more can be said about binary relations. In section 2.4 we introduce so-called allegories. By definition, an allegory is a category with some additional axioms to capture some of the structure of binary relations over sets. We will give the relational extension of some of the categorical notions defined on functions to relations. Again, category Rel will be the leading example with which we motivate the definition of the relational extensions.

2.1.1 Isomorphisms

A special kind of arrow is a so-called isomorphism. Arrow $f : A \leftarrow B$ is an *isomorphism* if there exists an arrow $g : B \leftarrow A$ such that

$$g\cdot f = \mathsf{id}_B \ \wedge\ f\cdot g = \mathsf{id}_A \ . \tag{2.1}$$

If such a g exists then it is necessarily unique. For, assume that $h\cdot f = \mathsf{id}_B$ and $f\cdot h = \mathsf{id}_A$. Then

$$\begin{array}{ll}
 & g \\
= & \quad \{ \quad f\cdot h = \mathsf{id}_A \quad \} \\
 & g\cdot f\cdot h \\
= & \quad \{ \quad g\cdot f = \mathsf{id}_B \quad \} \\
 & h
\end{array}$$

In Map, the function g is the inverse of f. Hence, in Map the function f is an isomorphism precisely when f is a bijection.

2.1.2 Terminal and initial objects

A special kind of object is a so-called terminal object. An object T is *terminal* if for every object A there exists precisely one arrow of type $T \leftarrow A$. In Map, the singleton sets are terminal. For singleton set $\{x\}$, the only function of type $\{x\} \leftarrow A$ is the constant function mapping every element of A to x.

Assume the existence of terminal objects. Let 1 denote some fixed terminal object and let $!_A$ denote the unique arrow of type $1 \leftarrow A$. Then terminality of object 1 can equally defined by the following universal property,

$$f = !_A \quad \equiv \quad f : 1 \leftarrow A \enspace .$$

The dual of terminality is initiality. Object I is *initial* if for every object A there is precisely one arrow of type $A \leftarrow I$. Let ¡_A denote this unique arrow. Then we have the following universal property,

$$f = \text{¡}_A \quad \equiv \quad f : A \leftarrow I \enspace .$$

The empty set is the only initial object in Map.

Duality

Many categorical concepts we encounter have a dual counter-part. We have just seen an example. The dual of terminality is initiality. For category $\mathcal{C}$, we can define its so-called *opposite* category $\mathcal{C}^{op}$ by turning the arrows around. Formally, the objects of $\mathcal{C}^{op}$ are the objects of $\mathcal{C}$, and $f : B \leftarrow A$ is an arrow of $\mathcal{C}^{op}$ if $f : A \leftarrow B$ is an arrow of $\mathcal{C}$. Furthermore, the order of composition is swapped, i.e. $f \circ g \triangleq g \cdot f$ is the composition of $\mathcal{C}^{op}$. So, if we have a categorical concept for a category then this categorical concept defined for category $\mathcal{C}^{op}$ is the dual concept for category $\mathcal{C}$. The dualized concept is derived from the original one by turning the arrows around. Similarly, properties proven for the original concept can be translated for the dual concept by turning the arrows around. In other words, if we have a categorical concept for which we have proven some properties, we get its dual counterpart together with its properties for “free”.

Diagrams

Instead of writing down an equation like,

$$f \cdot g = h \cdot i$$

one could depict the equation in a diagram:

$$\begin{tikzcd}
B \arrow[d, "f"'] & C \arrow[l, "g"'] \arrow[d, "i"] \\
A & D \arrow[l, "h"]
\end{tikzcd}$$

In the literature about category theory, it is common practice to reason with diagrams instead of using equational reasoning. We will not do so. Occasionally we will give a diagram instead of an equation because a diagram states the type information more explicitly. If any two paths between a pair of objects depict the same arrow, we say that the diagram *commutes*.

2.2 Datatypes and functors

Since the main objective of this thesis is to develop a theory in which it is possible to abstract from the data structure, we need an abstract notion of data structures. The requirement for this abstraction is that it be sufficiently abstract: we do not want to be bothered with too many details. But on the other hand, the abstraction should capture enough properties of a datatype in general such that it is possible to make other than trivial statements about datatypes.

The choice we have made can be summarized by the slogan "datatype = relator". A relator is the relational extension of a so-called functor on functions. Not all functors have a relational extension, but all functors corresponding with datatypes we encounter in daily life do have one. Hence, we will only consider functors which have a relational extension. In section 2.5, we give the formal definition of a relator. For the moment, we forget about the relational extension and just work with functors.

For a specific datatype, we assume the existence of two notions. The first is the notion of a datatype former. Using a datatype former, we can construct, from existing datatypes, new datatypes. An example of a datatype former is pairing or tupling. Having the types A and B, one can construct the type $A \times B$ whose elements are pairs of elements of type A and B. Another example is the list former. Having type A, $\mathit{List}(A)$ is the type of all finite lists over A.

The second notion, directly connected with the first notion, is the notion of a generalized map. For lists, we have the well-known function *map*: it takes a function and a list and applies the function to each element of the list. In general, associated with a datatype former we assume a map which takes a function and data structure and applies the function to each of the elements of the data structure.

Formalizing both notions for categories in general yields the notion of a functor.

2.2.1 Functors

A functor consists of two mappings[1], one mapping objects to objects and one mapping arrows to arrows, which respects identities and distributes over composition. Both mappings we usually denote by the same symbol. Formally, F is a *functor* to category $\mathcal{C}$ from $\mathcal{D}$, i.e. $F : \mathcal{C} \leftarrow \mathcal{D}$, iff

$$F\mathsf{id}_A = \mathsf{id}_{FA} \ ,$$

and, for each $f : A \leftarrow B$ and $g : B \leftarrow C$,

$$Ff \cdot Fg = F(f \cdot g) \ .$$

These two requirements imply the following typing rule:

$$Ff \ : \ FA \leftarrow FB \qquad \Leftarrow \qquad f : A \leftarrow B \ .$$

[1]We adopt the convention that we call functions on the meta level "mappings" and use the term "function" only if we mean an arrow of a category.

Note that we denote functor application by juxtaposition. It is trivial to verify that functors are closed under composition. Functor composition we usually denote also by juxtaposition. Hence, FGf means either $\mathsf{F}(\mathsf{Gf})$ or $(\mathsf{FG})\mathsf{f}$. However, if we want to make the composition of functors explicit, we write $\mathsf{F}\circ\mathsf{G}$ instead of FG.

The classical example of a functor is the map operator on lists known for functional programming. As we remarked above, for lists the object map is the type-constructor *List*, i.e. for type A, $\mathit{List}(\mathsf{A})$ is the type of lists over type A. The function *map* is the arrow map: it takes a function and a list and applies the function to each element of the list. Hence, for function $\mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}$, we have $\mathit{map}(\mathsf{f}) \ : \ \mathit{List}(\mathsf{A}) \leftarrow \mathit{List}(\mathsf{B})$. It is trivial to verify that map indeed respects identities and distributes over composition. Two other more trivial, but nevertheless important examples of a functor are the identity functor and the constant functor. The *identity functor* $\mathtt{Id}$ leaves both the object and arrow unchanged. The *constant functor* K_A, for some object A, maps each object to A, and each arrow to id_A. Functors we denote by F, G, ..., or by a capitalized identifier like $\mathtt{Id}$, $\mathtt{Outl}$, etc.

F-structures and shapes Elements of FA, for some A, we call F-*structures*. The interpretation of an F-structure is that it is a data-structure containing elements of type A. Then the interpretation of Ff, for $\mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}$, is that it transforms an F-structure of A's into an F-structure of B's by applying the function f to each element of the F-structure. For an F-structure of A's, the function $\mathsf{F}!_\mathsf{A} \ : \ \mathsf{F1} \leftarrow \mathsf{FA}$ yields its so-called *shape*. That is, $\mathsf{F}!_\mathsf{A}$ replaces each element of the F-structure of A's by the element 1. Now, two F-structures have the same shape if application of $\mathsf{F}!$ to both F-structures yields the same F-structure of 1's. Note that Ff respects shape since using the fact that functors distributes over composition and terminality of 1, we have,

$$\mathsf{F}! \cdot \mathsf{Ff} = \mathsf{F}(! \cdot \mathsf{f}) = \mathsf{F}! \ .$$

Hence, applying Ff to an F-structure and then computing its shape yields the same shape as the shape of the original F-structure.

2.2.2 Natural transformations

Directly related with functors is the notion of a natural transformation. For functors F and G to category $\mathcal{C}$ from category $\mathcal{D}$, a collection of arrows α of $\mathcal{C}$, indexed by objects of $\mathcal{D}$, is a *natural transformation* of type $\mathsf{F} \leftarrow \mathsf{G}$ if,

$$\mathsf{Ff} \cdot \alpha_\mathsf{B} = \alpha_\mathsf{A} \cdot \mathsf{Gf} \quad \text{for each } \mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}.$$

The interpretation of a natural transformation is that $\alpha_\mathsf{A} \ : \ \mathsf{FA} \leftarrow \mathsf{GA}$ transforms a G-structure of A's into an F-structure of A's without changing the value of the elements. In chapter 4 we give a formal justification for this claim.

An example of a natural transformation is the polymorphic reverse function on lists. For each type A, rev_A is the reverse function on a list of type $\mathit{List}(\mathsf{A})$. Indeed, for rev we have the property, for function $\mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}$,

$$\mathit{map}(\mathsf{f}) \cdot \mathsf{rev}_\mathsf{B} = \mathsf{rev}_\mathsf{A} \cdot \mathit{map}(\mathsf{f}) \ \ ,$$

since first applying function f to each element of the list and then reversing the list yields the same list as if we first reverse the list and then apply f to each element. Hence, rev is a natural transformation of type $List \leftarrow List$. Another example of a natural transformation is the collection of identity arrows id. We have that id is a natural transformation to the identity functor from the identity functor, i.e. $\mathsf{id} : \mathrm{Id} \leftarrow \mathrm{Id}$, since,

$$\mathrm{Id}(f) \cdot \mathsf{id}_B = \mathsf{id}_A \cdot \mathrm{Id}(f) \quad \text{for each } f : A \leftarrow B.$$

We will suppress the typing information if one can deduce from the context what the index is of a specific natural transformation. Natural transformations we denote either by Greek symbols α, β, . . ., or by sans serif identifiers like rev, id, etc.

2.2.3 Category Fun

As the notation $\alpha : F \leftarrow G$ suggests, it is the case that functors and natural transformations form a category. The *functor category*, denoted by Fun, has as objects functors and as arrows natural transformations between them. The identity arrow on functor F is id_F where id_F is defined by $(\mathsf{id}_F)_A = \mathsf{id}_{FA}$. Furthermore, the composition of $\alpha : F \leftarrow G$ and $\beta : G \leftarrow H$ is $\alpha \cdot \beta$ where $\alpha \cdot \beta$ is defined by $(\alpha \cdot \beta)_A = \alpha_A \cdot \beta_A$. It is not too difficult to verify that $\alpha \cdot \beta : F \leftarrow H$. Furthermore, composition of natural transformations is associative. Hence, Fun is indeed a category. Note that composition of natural transformations is defined as the pointwise lifting of the composition in the base category. In general, calculation steps involving pointwise liftings will be indicated by the hint "lifting".

The sub-category of Fun of functors to $\mathcal{C}$ from $\mathcal{D}$ we denote by $\mathsf{Fun}(\mathcal{C}, \mathcal{D})$.

If F is a functor to category $\mathcal{C}$ from $\mathcal{D}$, then functor F induces two functors on natural transformations. If $F : \mathcal{C} \leftarrow \mathcal{D}$ then $(F\circ) : \mathsf{Fun}(\mathcal{C}, \mathcal{E}) \leftarrow \mathsf{Fun}(\mathcal{D}, \mathcal{E})$ for each category $\mathcal{E}$, where $(F\circ)$ denotes post-composition of functor F. More precisely, the object map of $(F\circ)$ is post-composition of F, i.e. the application of $(F\circ)$ to a functor G is defined to be FG, and the application of $(F\circ)$ to a natural transformation α is defined to be $F\alpha$ i.e. the composition of (the arrow map of) functor F after the natural transformation α. (Recall that we view a natural transformation as a mapping from objects to arrows.) It is not too difficult to prove that $(F\circ)$ is indeed a functor on natural transformations. First of all, $(F\circ)$ applied to a natural transformation $\alpha : G \leftarrow H$ is a natural transformation of type $FG \leftarrow FH$ since, for $f : A \leftarrow B$,

$$FGf \cdot F\alpha_A = F(Gf \cdot \alpha_A) = F(\alpha_B \cdot Hf) = F\alpha_B \cdot FHf .$$

Furthermore, functor $(F\circ)$ respects identities, and distributes over composition:

$$(F\alpha \cdot F\beta)_A = F\alpha_A \cdot F\beta_A = F(\alpha_A \cdot \beta_A) = F(\alpha \cdot \beta)_A .$$

The second functor on natural transformations induced by functor $F : \mathcal{C} \leftarrow \mathcal{D}$ is $(\circ F) : \mathsf{Fun}(\mathcal{E}, \mathcal{D}) \leftarrow \mathsf{Fun}(\mathcal{E}, \mathcal{C})$, for each category $\mathcal{E}$, where $(\circ F)$ is pre-composition of F. Specifically, the application of $(\circ F)$ to a functor G is defined to be GF and the application of $(\circ F)$ to a natural transformation α is the mapping that maps object A to α_{FA}. Again, it is not too difficult to prove that $(\circ F)$ so defined is a functor on natural transformations. First of all, $(\circ F)$ applied to natural transformation $\alpha : G \leftarrow H$ is a natural transformation of type $GF \leftarrow HF$ since, for $f : A \leftarrow B$,

$$\alpha_{FA} \cdot GFf = HFf \cdot \alpha_{FB} .$$

Furthermore, functor $(\circ F)$ respects identities by definition, and distributes over composition since:

$$\alpha_{FA} \cdot \beta_{FA} = (\alpha \cdot \beta)_{FA} \enspace .$$

Basically, $(\circ F)$ distributes over the composition of natural transformations since this composition is defined as the pointwise lifting of the composition in the base category.

Like functor composition, we denote the composition of a functor after a natural transformation by juxtaposition. Hence, $F\alpha_A$ means either $F(\alpha_A)$ or $(F\alpha)_A$. However, the composition of a natural transformation after (the object map of) a functor we denote by α_F instead of αF. The reason for this is that the notation α_F is more consistent with the decision we made to suppress type information if the type of the natural transformation can be deduced from the context.

2.2.4 Product and coproduct

The first two non-trivial examples of datatype formers we define formally are product and coproduct. Given two types A and B we can construct their product, or tupling, $A \times B$ elements of which are pairs of elements of A and B. The coproduct, or disjoint sum, of A and B, denoted by $A + B$ is basically the union of A and B but constructed in such a way that it is always possible to decide whether an element of $A + B$ originates from A or from B.

Product

We begin with the abstract definition of product. A *product* of two objects consists of an object and two *projection arrows*. The object is denoted by $A \times B$ and the two arrows by $\mathsf{outl}_{A,B} : A \leftarrow A \times B$ and $\mathsf{outr}_{A,B} : B \leftarrow A \times B$. Furthermore, for each pair of arrows $f : A \leftarrow C$ and $g : B \leftarrow C$ with the same source we require the existence of the so-called *split* of f and g, denoted by $f \vartriangle g : A \times B \leftarrow C$, for which the following universal property holds:

$$h = f \vartriangle g \quad \equiv \quad \mathsf{outl}_{A,B} \cdot h = f \ \wedge \ \mathsf{outr}_{A,B} \cdot h = g \enspace , \tag{2.2}$$

for each $h : A \times B \leftarrow C$. We say that a category *has products* if each pair of objects has a product.

Category Map has products: $A \times B$ is the cartesian product of the sets A and B, and $\mathsf{outl}_{A,B}$ and $\mathsf{outr}_{A,B}$ are the normal projection functions. Then $f \vartriangle g$ is the function defined by $(f \vartriangle g)(x) = (f(x), g(x))$.

As with natural transformations, if one can deduce from the context what the type of a specific projection arrow is, we will suppress the typing information.

Taking $h := f \vartriangle g$, we get from the universal property the so-called computation rules for split:

$$\mathsf{outl} \cdot (f \vartriangle g) = f \ \wedge \ \mathsf{outr} \cdot (f \vartriangle g) = g \enspace . \tag{2.3}$$

Furthermore, taking $f := \mathsf{outl} \cdot h$ and $g := \mathsf{outr} \cdot h$, yields, for $h : A \times B \leftarrow C$,

$$h = (\mathsf{outl} \cdot h) \vartriangle (\mathsf{outr} \cdot h) \enspace . \tag{2.4}$$

Similarly, the following distribution law is easy to verify,

$$(f \vartriangle g) \cdot h = (f \cdot h) \vartriangle (g \cdot h) \enspace . \tag{2.5}$$

The product functor

If category $\mathcal{C}$ has products we can define a *product functor*: for all $f : A \leftarrow C$ and $g : B \leftarrow D$

$$f \times g \quad \triangleq \quad (f \cdot \mathsf{outl}_{C,D}) \vartriangle (g \cdot \mathsf{outr}_{C,D}) \; : \; A \times B \leftarrow C \times D \;\; .$$

For category Map, $f \times g$ is the function that maps the pair $(a\,,b)$ to $(f(a)\,,g(b))$.

That "$\times$" is indeed a functor is not too difficult to verify. From equation (2.4) it follows by taking $h := \mathsf{id}_{A \times B}$ that "$\times$" respects identities. That "$\times$" distributes over composition follows from the so-called *product-split fusion rule*:

$$(f \times g) \cdot (h \vartriangle i) = (f \cdot h) \vartriangle (g \cdot i)$$

since using the distribution property (2.5) and computation rules (2.3)

$$(f \times g) \cdot (h \vartriangle i) = (f \cdot \mathsf{outl} \cdot (h \vartriangle i)) \vartriangle (g \cdot \mathsf{outr} \cdot (h \vartriangle i)) = (f \cdot h) \vartriangle (g \cdot i) \;\; .$$

Using the product-split fusion rule we can verify that "$\times$" distributes over composition, i.e,

$$(f \times g) \cdot (h \times i) = (f \cdot h) \times (g \cdot i)$$

since $h \times i = (h \cdot \mathsf{outl}) \vartriangle (i \cdot \mathsf{outr})$. From the computation rule for split (2.3) it follows that the projection arrows distribute through "$\times$", for $f : A \leftarrow C$ and $g : B \leftarrow D$,

$$\mathsf{outl}_{A,B} \cdot (f \times g) = f \cdot \mathsf{outl}_{C,D} \;\wedge\; \mathsf{outr}_{A,B} \cdot (f \times g) = g \cdot \mathsf{outr}_{C,D}$$

since, for example,

$$\mathsf{outl}_{A,B} \cdot (f \times g) = \mathsf{outl}_{A,B} \cdot ((f \cdot \mathsf{outl}_{C,D}) \vartriangle (g \cdot \mathsf{outr}_{C,D})) = f \cdot \mathsf{outl}_{C,D}$$

In other words, outl and outr are natural transformations of type $\mathrm{Outl} \leftarrow \times$ and $\mathrm{Outr} \leftarrow \times$, respectively, where Outl and Outr denote the projection functors of type $\mathcal{C} \leftarrow \mathcal{C}^2$. That is, $\mathrm{Outl}(f\,,g) = f$ and $\mathrm{Outr}(f\,,g) = g$.

Coproduct

The definition of coproduct is the dual of product. So, the coproduct of A and B in category $\mathcal{C}$ is the product of A and B in the opposite category $\mathcal{C}^{op}$. Thus, a *coproduct* of two objects consists of an object and two so-called *injection arrows*. The object is denoted by $A + B$ and the two arrows by $\mathsf{inl}_{A,B} \; : \; A + B \leftarrow A$ and $\mathsf{inr}_{A,B} \; : \; A + B \leftarrow B$. Furthermore, for each pair of arrows $f : C \leftarrow A$ and $g : C \leftarrow B$ with the same target we require the existence of the so-called *junc* of f and g, denoted by $f \triangledown g \; : \; C \leftarrow A + B$, for which the following universal property holds:

$$h = f \triangledown g \quad \equiv \quad h \cdot \mathsf{inl}_{A,B} = f \;\wedge\; h \cdot \mathsf{inr}_{A,B} = g \;\; , \tag{2.6}$$

for each $h \; : \; C \leftarrow A + B$. We say that a category *has coproducts* if each pair of objects has a coproduct.

Category Map has coproducts: $A + B$ is a disjoint union of the sets A and B, i.e.

$$A + B = \{(a\,,0) \mid a \in A\} \cup \{(b\,,1) \mid b \in B\} \;\; ,$$

and the functions $\mathsf{inl}_{A,B}$ and $\mathsf{inr}_{A,B}$ add a 0 and 1 tag, respectively. Then $f \triangledown g$ is the function defined by $(f \triangledown g)(a\,,0) = f(a)$ and $(f \triangledown g)(b\,,1) = g(b)$.

Since coproduct is the dual of product, we get the properties for coproduct for "free" by turning the direction of the arrows around. The dual of computation rules (2.3) are the computation rules for junc:

$$(f \triangledown g) \cdot \mathsf{inl}_{A,B} = f \;\;\wedge\;\; (f \triangledown g) \cdot \mathsf{inr}_{A,B} = g \;\;, \tag{2.7}$$

and, the dual of (2.4) is, for $h : C \leftarrow A + B$,

$$h = (h \cdot \mathsf{inl}_{A,B}) \triangledown (h \cdot \mathsf{inr}_{A,B}) \;\;. \tag{2.8}$$

For junc we have the dual distribution law:

$$h \cdot (f \triangledown g) = (h \cdot f) \triangledown (h \cdot g) \;\;. \tag{2.9}$$

The coproduct functor

Just as for product, if category $\mathcal{C}$ has coproducts we can define a coproduct functor. Again, we dualize the results for product. Hence, the coproduct functor is defined by, for $f : A \leftarrow C$ and $g : B \leftarrow D$

$$f + g \;\;\triangleq\;\; (\mathsf{inl}_{A,B} \cdot f) \triangledown (\mathsf{inr}_{A,B} \cdot g) \;:\; A + B \leftarrow C + D \;\;.$$

For category Map, $f + g$ is the function which leaves the tagging of an element unchanged and applies f or g to the element if it is tagged with 0 or 1, respectively.

Since "$\times$" is a functor it follows from duality that "$+$" is a functor too. Dualising the product-split fusion rule, we get the so-called *junc-coproduct fusion rule*:

$$(f \triangledown g) \cdot (h + i) = (f \cdot h) \triangledown (g \cdot i) \;\;.$$

Like projections, inl and inr are natural transformations, for $f : A \leftarrow C$ and $g : B \leftarrow D$,

$$(f + g) \cdot \mathsf{inl}_{C,D} = \mathsf{inl}_{A,B} \cdot f \;\;\wedge\;\; (f + g) \cdot \mathsf{inr}_{C,D} = \mathsf{inr}_{A,B} \cdot g \;\;.$$

So, $\mathsf{inl} : + \leftarrow \mathrm{Outl}$ and $\mathsf{inr} : + \leftarrow \mathrm{Outr}$.

Product and coproduct in category Fun

If category $\mathcal{C}$ has products then so has category $\mathsf{Fun}(\mathcal{C}\,,\mathcal{D})$. The product of functors F and G is the functor $F \times G$ that maps arrow f to $Ff \times Gf$, and the two corresponding projections are $\mathsf{outl}_{F,G}$ and $\mathsf{outr}_{F,G}$ defined by $(\mathsf{outl}_{F,G})_A = \mathsf{outl}_{FA\,,GA}$ and $(\mathsf{outr}_{F,G})_A = \mathsf{outr}_{FA\,,GA}$, respectively. The split operator on natural transformations $\alpha : F \leftarrow G$ and $\beta : H \leftarrow G$ is defined as the pointwise lifting of the split operator "$\vartriangle$" in the base category $\mathcal{C}$. That is to say, $\alpha \vartriangle \beta$ is defined by $(\alpha \vartriangle \beta)_A = \alpha_A \vartriangle \beta_A$. It is straightforward to verify that $(F \times G\,, \mathsf{outl}_{F,G}\,, \mathsf{outr}_{F,G})$ defines a product for the functors F and G. The junc operator and coproduct are lifted similarly.

Throughout the remainder of this thesis, we silently lift operations defined on arrows to functors and natural transformations. For instance, for binary functor $\otimes$, the functor $F \otimes G$ is defined by $(F \otimes G) f = Ff \otimes Gf$. Similarly, the natural transformation $\alpha \otimes \beta$ is defined by $(\alpha \otimes \beta)_A = \alpha_A \otimes \beta_A$. Note that definition of the lifting of an operation defined on arrows to natural transformations is consistent with our convention to occasionally suppress the typing of natural transformations.

2.2.5 Recursive datatypes

Until now we have only considered non-recursive datatypes. In this section we describe how one can construct recursive datatypes. The two leading examples will be natural numbers and (cons) lists. A natural number is either zero or the successor of a natural number. Similarly, a list is either the empty list or an element concatenated to a list. That is to say, we have the following constructors, for naturals,

$$\mathsf{zero} : \mathit{Nat} \leftarrow 1 \ \wedge\ \mathsf{succ} : \mathit{Nat} \leftarrow \mathit{Nat} \ ,$$

and for lists over type B, denoted by $\mathit{List}(B)$,

$$\mathsf{nill} : \mathit{List}(B) \leftarrow 1 \quad \wedge \quad \mathsf{cons} : \mathit{List}(B) \leftarrow B \times \mathit{List}(B) \ .$$

Note that the constructors of a type have the same target, hence, using coproduct the typing of the individual constructors can be combined in a single statement:

$$\mathsf{zero} \triangledown \mathsf{succ} : \mathit{Nat} \leftarrow 1 + \mathit{Nat} \ ,$$

and

$$\mathsf{nill} \triangledown \mathsf{cons} : \mathit{List}(B) \leftarrow 1 + B \times \mathit{List}(B) \ .$$

Note that both "juncs" have the typing $A \leftarrow FA$, for some object A and some functor F. For the naturals, $A = \mathit{Nat}$ and $FX = \mathsf{id}_1 + X$, and for lists over type B, $A = \mathit{List}(B)$ and $FX = \mathsf{id}_1 + \mathsf{id}_B \times X$. An arrow of type $A \leftarrow FA$ is called an F-*algebra* and object A is the so-called *carrier* of the F-algebra.

Connected with F-algebras, is the notion of an F-homomorphism. An F-*homomorphism* to an algebra $f : A \leftarrow FA$ from an algebra $g : B \leftarrow FB$ is an arrow $h : A \leftarrow B$ such that

$$h \cdot g = f \cdot Fh \ .$$

For instance, the function which computes the size of a list of B's is an F-homomorphism where F is the functor used to define lists of B's, i.e. $FX = \mathsf{id}_1 + \mathsf{id}_B \times X$. We have that $\mathsf{size} : \mathit{Nat} \leftarrow \mathit{List}(B)$ and size satisfies the following recursive equation, where $[]$ denotes the empty list,

$$\begin{aligned} \mathsf{size}([]) &= 0 \\ \mathsf{size}(\mathsf{cons}(a, x)) &= \mathsf{succ}(\mathsf{size}(x)) \end{aligned}$$

or, equivalently without points, since $\mathsf{succ}(\mathsf{size}(x)) = (\mathsf{succ} \cdot \mathsf{outr} \cdot (\mathsf{id} \times \mathsf{size}))(a, x)$

$$\begin{aligned} \mathsf{size} \cdot \mathsf{nill} &= \mathsf{zero} \\ \mathsf{size} \cdot \mathsf{cons} &= \mathsf{succ} \cdot \mathsf{outr} \cdot (\mathsf{id} \times \mathsf{size}) \end{aligned}$$

Again, using coproduct both equations can be combined into one:

$$(\mathsf{size} \cdot \mathsf{nill}) \triangledown (\mathsf{size} \cdot \mathsf{cons}) = \mathsf{zero} \triangledown (\mathsf{succ} \cdot \mathsf{outr} \cdot (\mathsf{id} \times \mathsf{size})) \ . \tag{2.10}$$

Now, using the rules distribution (2.9) and junc-coproduct fusion, equation (2.10) can be rewritten as,

$$\mathsf{size} \cdot (\mathsf{nill} \triangledown \mathsf{cons}) = (\mathsf{zero} \triangledown (\mathsf{succ} \cdot \mathsf{outr})) \cdot (\mathsf{id}_1 + (\mathsf{id}_B \times \mathsf{size})) \ . \tag{2.11}$$

Hence, size is an F-homomorphism to F-algebra $\mathsf{zero} \triangledown (\mathsf{succ} \cdot \mathsf{outr})$ from F-algebra $\mathsf{nill} \triangledown \mathsf{cons}$.

The size function on lists is an example of a function which follows the structure of the recursively defined data structure. In general, a function which is recursively defined according to the structure of lists has the structure:

$$\begin{aligned} h([\,]) &= c \\ h(\mathsf{cons}(a\,,x)) &= f(a\,,h(x)) \end{aligned}$$

or, equivalently, without points,

$$\begin{aligned} h \cdot \mathsf{nill} &= c \\ h \cdot \mathsf{cons} &= f \cdot (\mathsf{id} \times h) \end{aligned}$$

Again, the two equations can be combined into one:

$$h \cdot (\mathsf{nill} \triangledown \mathsf{cons}) = (c \triangledown f) \cdot (\mathsf{id}_1 + (\mathsf{id}_B \times h)) \ . \tag{2.12}$$

In other words, h is an F-homomorphism to F-algebra $c \triangledown f$ from $\mathsf{nill} \triangledown \mathsf{cons}$. For lists, we know that the recursive equation uniquely defines function h. In functional programming, the function h is known to be an instance of *foldr*, specifically, $h = foldr(c\,,f)$. In general, if a recursive definition of a function according to the structure of a recursively defined data structure always has a unique solution, we say that the algebra corresponding to the data structure is *initial*.

Formally, let $\mathsf{in} : T \leftarrow FT$ be an F-algebra. Then in is *initial* iff for each F-algebra $f : A \leftarrow FA$ there exists an F-homomorphism of type $A \leftarrow T$, which we denote by $(\!|F;\ f|\!)$, for which the following universal property holds,

$$h = (\!|F;\ f|\!) \ \equiv \ h \cdot \mathsf{in} = f \cdot Fh \tag{2.13}$$

Hence, $(\!|F;\ f|\!)$ is the unique F-homomorphism to algebra f from in. Property (2.13) we call the *unique extension property*. Furthermore, we call $(\!|F;\ f|\!)$ the F-*catamorphism* of f. If it is clear from the context which functor is meant, we write $(\!|f|\!)$ instead of $(\!|F;\ f|\!)$.

Taking $h := \mathsf{id}_T$ and $f := \mathsf{in}$ in the unique extension property gives $\mathsf{id}_T = (\!|\mathsf{in}|\!)$. Similarly, taking $h := (\!|f|\!)$ gives the *computation rule*,

$$(\!|f|\!) \cdot \mathsf{in} = f \cdot F(\!|f|\!) \ .$$

Using the computation rule, we can prove the *fusion law*,

$$h \cdot (\!|f|\!) = (\!|g|\!) \ \Leftarrow \ h \cdot f = g \cdot Fh \ ,$$

since

$$
\begin{array}{ll}
 & h \cdot (\![f]\!) = (\![g]\!) \\
\equiv & \{ \quad \text{unique extension property} \quad \} \\
 & h \cdot (\![f]\!) \cdot \mathsf{in} = g \cdot F(h \cdot (\![f]\!)) \\
\equiv & \{ \quad \text{computation; } F\text{-functor} \quad \} \\
 & h \cdot f \cdot F(\![f]\!) = g \cdot Fh \cdot F(\![f]\!) \\
\Leftarrow & \{ \quad \text{Leibniz} \quad \} \\
 & h \cdot f = g \cdot Fh
\end{array}
$$

We claim that $\mathsf{in} : T \leftarrow FT$ is an isomorphism. In order to show that in is an isomorphism, we have to construct an arrow α of type $FT \leftarrow T$ such that

$$\alpha \cdot \mathsf{in} = \mathsf{id}_{FT} \;\wedge\; \mathsf{in} \cdot \alpha = \mathsf{id}_T \;. \tag{2.14}$$

Since the source of α is T, i.e. the carrier of the initial algebra in, we suspect that α is a catamorphism, i.e. $\alpha = (\![\beta]\!)$ for some β. We continue with the rhs of equation (2.14)

$$
\begin{array}{ll}
 & \mathsf{in} \cdot (\![\beta]\!) = \mathsf{id}_T \\
\Leftarrow & \{ \quad \mathsf{id}_T = (\![\mathsf{in}]\!), \text{ fusion} \quad \} \\
 & \mathsf{in} \cdot \beta = \mathsf{in} \cdot F\mathsf{in}
\end{array}
$$

Hence, if we take $\beta = F\mathsf{in}$, i.e. $\alpha = (\![F\mathsf{in}]\!)$, we have $\mathsf{in} \cdot \alpha = \mathsf{id}_T$. We verify whether for this choice of α the first conjunct of (2.14) is also true:

$$
\begin{array}{ll}
 & (\![F\mathsf{in}]\!) \cdot \mathsf{in} \\
= & \{ \quad \text{computation} \quad \} \\
 & F\mathsf{in} \cdot F(\![F\mathsf{in}]\!) \\
= & \{ \quad F \text{ functor} \quad \} \\
 & F(\mathsf{in} \cdot (\![F\mathsf{in}]\!)) \\
= & \{ \quad \text{above: } \mathsf{in} \cdot (\![F\mathsf{in}]\!) = \mathsf{id}_T, \; F \text{ functor} \quad \} \\
 & \mathsf{id}_{FT}
\end{array}
$$

Hence, an initial algebra is an isomorphism [29]. This fact we will exploit when we extend catamorphisms to relations. As in Map, a relation is an isomorphism iff it is a bijection.

Remark: the use of the term "initial" is because an initial F-algebra is initial in the category of F-algebras. The objects of this category are F-algebras and the arrows are F-homomorphisms.

Parameterized datatypes

Note that the example of lists is parameterized by an object. Recall that $\mathsf{nill} \triangledown \mathsf{cons}$ is an initial F-algebra of type $\mathit{List}(B) \leftarrow 1 + B \times \mathit{List}(B)$ where F is the functor $FX = \mathsf{id}_1 + \mathsf{id}_B \times X$. In general, and making the parameter B explicit, assume a collection of initial algebras in_B of type $TB \leftarrow B \otimes TB$ where T is some mapping on objects and $\otimes$ a binary functor. Hence, in_B

is a $(\mathsf{id}_B\otimes)$-algebra where $(\mathsf{id}_B\otimes)X = \mathsf{id}_B\otimes X$. For the example of lists, we have that T is the datatype former *List* and $\otimes$ is the binary functor defined by $X\otimes Y = 1+X\times Y$.

For lists, the map operator discussed in section (2.2.1) is the arrow map of the *List* functor. Such a map operator exists in general. For each arrow $f : A \leftarrow B$, we have to construct an arrow $\alpha\ :\ \mathsf{T}A \leftarrow \mathsf{T}B$. Since $\mathsf{T}B$ is the carrier of an initial algebra, we suspect that $\alpha = (\!|\mathsf{id}_B\otimes;\ \beta|\!)$ for some $\beta\ :\ \mathsf{T}A \leftarrow B\otimes \mathsf{T}A$. Since the target of in_A is $\mathsf{T}A$, we take $\beta = \mathsf{in}_A\cdot\gamma$ for some $\gamma\ :\ A\otimes\mathsf{T}A \leftarrow B\otimes\mathsf{T}A$. Now, f has the type $A\leftarrow B$, hence, $f\otimes\mathsf{id}_{\mathsf{T}A}\ :\ A\otimes\mathsf{T}A\leftarrow B\otimes\mathsf{T}A$. If we take $\gamma = f\otimes\mathsf{id}_{\mathsf{T}A}$, so $\alpha = (\!|\mathsf{id}_B\otimes;\ \mathsf{in}_A\cdot(f\otimes\mathsf{id}_{\mathsf{T}A})|\!)$, then α has the type $\mathsf{T}A\leftarrow\mathsf{T}B$, as desired.

Moreover,

$$\mathsf{T}f \ \triangleq\ (\!|\mathsf{id}_B\otimes;\ \mathsf{in}_A\cdot(f\otimes\mathsf{id}_{\mathsf{T}A})|\!)\ ,$$

defines a functor, the *tree type functor induced by* $\otimes$. Functor T respects identities since $(\!|\mathsf{in}_A|\!) = \mathsf{id}_{\mathsf{T}A}$. That T distributes over composition follows from the *tree type fusion rule*

$$(\!|f|\!)\cdot\mathsf{T}g = (\!|f\cdot(g\otimes\mathsf{id})|\!)$$

which we prove by

$$
\begin{array}{ll}
 & (\!|f|\!)\cdot\mathsf{T}g = (\!|f\cdot(g\otimes\mathsf{id})|\!)\\
\Leftarrow & \{\quad \text{definition } \mathsf{T}, \text{fusion}\quad\}\\
 & (\!|f|\!)\cdot\mathsf{in}_A\cdot(g\otimes\mathsf{id}) = f\cdot(g\otimes\mathsf{id})\cdot(\mathsf{id}\otimes(\!|f|\!))\\
\equiv & \{\quad \text{computation}\quad\}\\
 & f\cdot(\mathsf{id}\otimes(\!|f|\!))\cdot(g\otimes\mathsf{id}) = f\cdot(g\otimes\mathsf{id})\cdot(\mathsf{id}\otimes(\!|f|\!))\\
\equiv & \{\quad \otimes \text{ binary functor: } (\mathsf{id}\otimes(\!|f|\!))\cdot(g\otimes\mathsf{id}) = (g\otimes\mathsf{id})\cdot(\mathsf{id}\otimes(\!|f|\!))\quad\}\\
 & \mathit{true}
\end{array}
$$

Using the tree type fusion rule we can verify that T distributes over composition:

$$\mathsf{T}f\cdot\mathsf{T}g = (\!|\mathsf{in}\cdot(f\otimes\mathsf{id})\cdot(g\otimes\mathsf{id})|\!) = (\!|\mathsf{in}\cdot(f\cdot g)\otimes\mathsf{id}|\!) = \mathsf{T}(f\cdot g)\ .$$

2.3 Theorems for free

Having the notion of functors and natural transformation, we can formally define the notion of a "free theorem". For a natural transformation α of type $F\leftarrow G$ it follows that for each of the individual members of the collection of arrows α, we have $\alpha_A\ :\ FA\leftarrow GA$. In other words,

$$\alpha : F\leftarrow G\ \ \Rightarrow\ \ (\alpha_A\ :\ FA\leftarrow GA\ \ \text{for each } A)\ .$$

One can view α as a polymorphic function: it has a type as parameter. Now, for a collection of arrows — a mapping from objects to arrows — its "free theorem" holds if the other way around is also true. That is to say, if for some α,

$$\alpha : F\leftarrow G\ \ \Leftarrow\ \ (\alpha_A\ :\ FA\leftarrow GA\ \ \text{for each } A)\ . \tag{2.15}$$

we say that the "free theorem" predicted by the typing of α_A holds.

Reynolds [43] proved for a particular language that property (2.15) holds for *all* polymorphic functions that are definable in that language. He showed that all basic polymorphic functions are natural transformations, and he proved that all the language constructs preserves natural transformations.

One can not expect property (2.15) to hold for all polymorphic functions in all languages. For instance, if it is possible to define a polymorphic function in some ad hoc way, e.g. by giving a clause for each type, one can not expect property (2.15) to hold for all functions.

However, property (2.15) holds for a polymorphic function which is constructed using constructs that behave "nicely". For instance, if we construct a polymorphic function using natural transformations and constructions that preserve naturality, it follows that this function is a natural transformation.

So, we consider property (2.15) as a healthiness condition for a polymorphic function. The informal interpretation of natural transformation is that its behaviour does not depend on the type at which it is instantiated. In other words, we consider a polymorphic function for which property (2.15) holds to be *parametrically* polymorphic in the sense that the definition of the polymorphic function does not depend crucially on the actual value of the type at which the polymorphic function is being instantiated.

2.3.1 Higher-order naturality

Extending Reynolds' idea further we also use the higher-order naturality property predicted by the type of a *polytypic* function as a healthiness condition. That is, we lift property (2.15) to a higher-level. By a higher-level we mean that we instantiate parameter (object) A with a functor and functors F and G with functors on the functor category. Functor $\boldsymbol{F}$ is a functor on the functor category if it consists of two mappings, a mapping that maps functors to functors —the object map of $\boldsymbol{F}$— and a mapping that maps natural transformations to natural transformations —the arrow map of $\boldsymbol{F}$—. Furthermore, $\boldsymbol{F}$ should distribute over composition of natural transformations and respect identities.

Now, let β denote a polytypic natural transformation such that β_{F} is a natural transformation of type $\boldsymbol{G}\mathsf{F} \leftarrow \boldsymbol{H}\mathsf{F}$ for higher-order functors $\boldsymbol{G}$ and $\boldsymbol{H}$. Then β is a higher-order natural transformation of type $\boldsymbol{G} \leftarrow \boldsymbol{H}$ if

$$\boldsymbol{G}\alpha \cdot \beta_{\mathsf{L}} = \beta_{\mathsf{K}} \cdot \boldsymbol{H}\alpha \quad \text{for each } \alpha : \mathsf{K} \leftarrow \mathsf{L} \ . \tag{2.16}$$

So, if for some polytypic β we have,

$$\beta : \boldsymbol{G} \leftarrow \boldsymbol{H} \quad \Leftarrow \quad (\beta_{\mathsf{F}} : \boldsymbol{G}\mathsf{F} \leftarrow \boldsymbol{H}\mathsf{F} \quad \text{for each } \mathsf{F}) \ , \tag{2.17}$$

we say that the higher-order "free theorem" predicted by the typing of β_{F} holds.

For example, define $\beta_{\mathsf{F}} = \mathsf{F}\gamma$ for some fixed natural transformation $\gamma : \mathsf{G} \leftarrow \mathsf{H}$, i.e. β is post-composition with a functor. Then we have $\beta_{\mathsf{F}} : \boldsymbol{G}\mathsf{F} \leftarrow \boldsymbol{H}\mathsf{F}$, for each functor F, where $\boldsymbol{G} = (\circ\mathsf{G})$ and $\boldsymbol{H} = (\circ\mathsf{H})$, i.e. $\boldsymbol{G}$ and $\boldsymbol{H}$ are pre-composition with functors G and H, respectively. So, the "free theorem" predicted by the typing $\beta_{\mathsf{F}} : \boldsymbol{G}\mathsf{F} \leftarrow \boldsymbol{H}\mathsf{F}$, is the higher-order naturality property $\beta : \boldsymbol{G} \leftarrow \boldsymbol{H}$. Expanding the definition of higher-order naturality yields,

$$\boldsymbol{G}\alpha \cdot \beta_{\mathsf{L}} = \beta_{\mathsf{K}} \cdot \boldsymbol{H}\alpha \quad \text{for each } \alpha : \mathsf{K} \leftarrow \mathsf{L}$$

and instantiating with the definition of β and $\boldsymbol{G}$ and $\boldsymbol{H}$ gives us,

$$\alpha_{\mathsf{G}} \cdot \mathsf{L}\gamma = \mathsf{K}\gamma \cdot \alpha_{\mathsf{H}} \quad \text{for each } \alpha : \mathsf{K} \leftarrow \mathsf{L}$$

which follows trivially from the typing of γ and α.

So, we conclude that post-composition with an arbitrary functor is a truly polytypic operation since the higher-order naturality predicted by the typing of post-composition holds. Of course, post-composition is a trivial example. In Chapter 5 we use (2.17) as a healthiness condition for the polytypic operation that commutes two datatypes.

2.4 Allegories

Allegories are to binary relations between sets as categories are to functions between sets. An allegory $\mathcal{A}$ is a category with some additional structure to capture the properties of binary relations. This additional structure is inspired by the properties we have for binary relations over sets. Although other allegories than the category Rel exist, we talk about allegories as if we are working with binary relations over sets. That is to say, if we give an interpretation of a property, we give the interpretation with respect to the set theoretical model of relations. Furthermore, we use the same terminology for arbitrary allegories as for the category Rel. In particular, we talk about "relations" instead of "arrows".

The reason why we work with relations instead of functions is because it is easier to express partial and nondeterministic operations than working in a strict functional setting. The move from functions to relations increases our powers of expression. For instance, we can use the nondeterministic nature of relations in order to specify nondeterministic problems. Of course, it is always possible to use set-valued functions or binary predicates instead of relations. However, we find it more natural to work with the relations directly. Since we were already working in a categorical setting, i.e. working without elements (points), the price to pay in order to shift from functions to relations is not that high.

The calculus of relations has existed for a very long time. (See e.g. [49, 50, 47].) Our presentation of the calculus is based on the work of Backhouse et. al. [3, 2, 5, 1] and the book by Bird and De Moor [7]. The latter is based on the theory of allegories as set out in [17]. We use allegories too.

2.4.1 Galois connections

Before we introduce the notion of an allegory we first introduce the concept of a Galois connection. As we will see, we introduce several allegorical notions as instances of a Galois connection. The theory of Galois connections is not that deep but it helps us to shape the setup of the theory and our calculations.

Assume two partial orders $\leq$ and $\sqsubseteq$. Then we say that the functions f and g form a *Galois connection*, or are *Galois connected* if for all x and y,

$$f(x) \leq y \quad \equiv \quad x \sqsubseteq g(y) \ . \tag{2.18}$$

We call functions f and g the *lower adjoint* and *upper adjoint*, respectively, of the Galois connection. The first property we prove is that for a given function f there exists at most one upper adjoint. That is, assume that g and h are both an upper adjoint of f, then for each x and y,

$$\begin{array}{ll}
 & x \sqsubseteq g(y) \\
\equiv & \quad \{ \quad \text{f and g are Galois connected} \quad \} \\
 & f(x) \leq y \\
\equiv & \quad \{ \quad \text{f and h are Galois connected} \quad \} \\
 & x \sqsubseteq h(y)
\end{array}$$

Now, instantiating $x := g(y)$, and instantiating $x := h(y)$ gives $g(y) \sqsubseteq h(y)$ and $h(y) \sqsubseteq g(y)$, respectively. By anti-symmetry of $\sqsubseteq$ it follows that $g(y) = h(y)$ for each y, hence, $g = h$.

If we know that two functions are Galois connected we get a number of properties for free. For instance, we have the so-called *cancellation rules*,

$$x \sqsubseteq g(f(x)) \quad \text{and} \quad f(g(y)) \leq y \ .$$

which follow from (2.18) by instantiating $y := f(x)$, and by instantiating $x := g(y)$ such that the lhs and rhs, respectively, becomes true. Furthermore, the functions of a Galois connection are monotonic. For instance, we have,

$$\begin{array}{ll}
 & f(x) \leq f(y) \\
\equiv & \quad \{ \quad \text{f and g are Galois connected (2.18)} \quad \} \\
 & x \sqsubseteq g(f(y)) \\
\Leftarrow & \quad \{ \quad \text{cancellation: } y \sqsubseteq g(f(y))\text{; transivity of } \sqsubseteq \quad \} \\
 & x \sqsubseteq y
\end{array}$$

The function f distributes over arbitrary joins and g over arbitrary meets provided they exist. In particular, f is bottom-strict and g is top-strict. As an example, we prove that f distributes over arbitrary joins. That is, let $\cup$ and $\sqcup$ denote the join of $\leq$ and $\sqsubseteq$, respectively. Then we have to show that for each set $\mathcal{S}$ such that $\sqcup\mathcal{S}$ exists,

$$f(\sqcup\mathcal{S}) = \cup\{f(S) \mid S \in \mathcal{S}\} \ .$$

We calculate, for all X,

$$\begin{array}{ll}
 & f(\sqcup\mathcal{S}) \leq X \\
\equiv & \quad \{ \quad \text{f and g are Galois connected} \quad \}
\end{array}$$

$$
\begin{array}{ll}
& \sqcup\mathcal{S} \sqsubseteq g(X) \\
\equiv & \{ \quad \text{join} \quad \} \\
& \forall(S : S \in \mathcal{S} : S \sqsubseteq g(X)) \\
\equiv & \{ \quad f \text{ and } g \text{ are Galois connected} \quad \} \\
& \forall(S : S \in \mathcal{S} : f(S) \leq X) \\
\equiv & \{ \quad \text{join} \quad \} \\
& \cup\{f(S) \mid S \in \mathcal{S}\} \leq X
\end{array}
$$

Hence, by anti-symmetry of $\leq$ it follows that $f(\sqcup\mathcal{S}) = \cup\{f(S) \mid S \in \mathcal{S}\}$.

As mentioned earlier, we define a number of operators by means of a Galois connection. Subsequently, we use the fact that such operators are monotonic, distribute over arbitrary joins or meets, and obey certain cancellation properties as a matter of course.

2.4.2 Definition

An allegory is a category satisfying some additional axioms. Because it is a category, for every object A there exists an identity id_A, and every pair of relations $R : A \leftarrow B$ and $S : B \leftarrow C$, with matching source and target, can be composed: $R \cdot S : A \leftarrow C$. Composition is associative and has id as a unit.

The additional axioms are as follows. First of all, relations of the same type are ordered by the *partial order* $\subseteq$ and composition is monotonic with respect to this order, that is,

$$S_1 \cdot T_1 \subseteq S_2 \cdot T_2 \quad \Leftarrow \quad S_1 \subseteq S_2 \ \wedge \ T_1 \subseteq T_2 \ .$$

In Rel, the partial order $\subseteq$ is just set inclusion.

Secondly, for every pair of relations $R, S : A \leftarrow B$, its *intersection* (*meet*) $R \cap S$ exists and is defined by the following universal property, for each $X : A \leftarrow B$,

$$X \subseteq R \ \wedge \ X \subseteq S \quad \equiv \quad X \subseteq R \cap S$$

Using the definition of intersection it is easy to verify that intersection is idempotent, commutative and associative. Note that we do not assume the existence of arbitrary intersections. If a largest relation of type $A \leftarrow B$ exists we denote it by $\top\!\!\top_{A,B}$. In Rel, $\top\!\!\top_{A,B}$ is the cartesian product $A \times B$.

From the fact that composition is monotonic and the universal property of intersection, it follows that composition distributes over intersection with an inclusion. That is to say,

$$(R \cap S) \cdot T \subseteq (R \cdot T) \cap (S \cdot T) \ .$$

Finally, for each relation $R : A \leftarrow B$ its *converse* $R^\circ : B \leftarrow A$ exists. In Rel we have, $x \, R^\circ \, y$ iff $y \, R \, x$. The converse operator is defined by the requirements that it is its own Galois adjoint, that is,

$$R^\circ \subseteq S \quad \equiv \quad R \subseteq S^\circ$$

and is contravariant with respect to composition,

$$(R \cdot S)^\circ = S^\circ \cdot R^\circ \ .$$

Since converse is defined using a Galois connection, we have the following cancellation rules $R \subseteq (R^\circ)^\circ$ and $(R^\circ)^\circ \subseteq R$. Hence, converse is its own inverse: $(R^\circ)^\circ = R$. Furthermore, from the theory of Galois connections, it follows that converse distributes over arbitrary meets. Specifically, we have,

$$(R \cap S)^\circ = R^\circ \cap S^\circ \ .$$

As might be suspected, it is the case that $\mathsf{id} = \mathsf{id}^\circ$ since,

$$\mathsf{id} = (\mathsf{id}^\circ)^\circ = (\mathsf{id}^\circ \cdot \mathsf{id})^\circ = \mathsf{id}^\circ \cdot (\mathsf{id}^\circ)^\circ = \mathsf{id}^\circ \cdot \mathsf{id} = \mathsf{id}^\circ \ .$$

All the three operators of an allegory are connected by the *modular law*, also known as Dedekind's law [45]. We have as an axiom

$$(R \cdot S) \cap T \subseteq (R \cap (T \cdot S^\circ)) \cdot S \ .$$

Taking the converse of both sides, we have dually,

$$(R \cdot S) \cap T \subseteq R \cdot (S \cap (R^\circ \cdot T)) \ .$$

Note that allegory Rel has more structure than we have captured so far with our axioms. For instance, in Rel we can take arbitrary unions (joins) of relations. However, the basic definition of an allegory has already sufficient structure to define some useful concepts. In subsection 2.4.9 we give the axioms for the union of relations.

2.4.3 Partial identities

Relations below an identity relation, i.e. $X \subseteq \mathsf{id}_A$ for some A, we call *partial identities* (sometimes called "coreflexives" [17] or "monotypes" [1]). We use partial identities to represent subsets of object A. In Rel, there is a partial identity on A for every subset B of A. That is to say, for subset B we can define the partial identity X, by $b\,X\,b \equiv b \in B$. We say that X is the partial identity *corresponding to* B.

Partial identities are symmetric. First notice that for arbitrary relation R, we have $R \subseteq R \cdot R^\circ \cdot R$ since using the modular law,

$$R = (\mathsf{id} \cdot R) \cap R \subseteq (\mathsf{id} \cap (R \cdot R^\circ)) \cdot R \subseteq R \cdot R^\circ \cdot R \ .$$

Hence, for partial identity X, $X \subseteq X \cdot X^\circ \cdot X \subseteq X^\circ$, so $X = X^\circ$. Furthermore, intersection and composition of partial identities coincide. Using monotonicity it follows trivially for partial identities X and Y that $X \cdot Y \subseteq X \cap Y$. The other inclusion follows from the modular law,

$$X \cap Y = (\mathsf{id} \cdot X) \cap Y \subseteq (\mathsf{id} \cap (X \cdot Y^\circ)) \cdot Y \subseteq X \cdot Y \ .$$

Hence, $X \cdot Y = X \cap Y$.

2.4.4 Simple and total relations

A special sub-class of relations is formed by functions. Functions are total and single-valued relations, that is to say, relation f is a function iff for each x there is precisely one y such that $y\ f\ x$. Formally, relation $R : A \leftarrow B$ is *total* or entire iff

$$id_B \subseteq R^\circ \cdot R \ ,$$

and relation R is single-valued or *simple* iff

$$R \cdot R^\circ \subseteq id_A \ .$$

A *function* is a relation that is both total and simple. It is easy to verify that total and simple relations are closed under composition. Hence, functions are closed under composition too. In other words, the functions form a sub-category. For an allegory $\mathcal{A}$, we denote the sub-category of functions by $Map(\mathcal{A})$. In particular, we have $Map(\mathsf{Rel}) = \mathsf{Map}$.

We say that R is *surjective* iff R° is total, and R is *injective* iff R° is simple. Furthermore, R is a *cofunction* if R° is a function, and R is a *bijection* if R is both a function and a cofunction.

Note that arbitrary relations do not distribute over intersection, but simple relations do distribute over intersection from the right. We calculate:

$$
\begin{array}{ll}
 & (R \cap S) \cdot f \\
\subseteq & \quad \{ \quad \text{distribution} \quad \} \\
 & (R \cdot f) \cap (S \cdot f) \\
\subseteq & \quad \{ \quad \text{modular law} \quad \} \\
 & (R \cap (S \cdot f \cdot f^\circ)) \cdot f \\
\subseteq & \quad \{ \quad \text{assumption: } f \cdot f^\circ \subseteq id \quad \} \\
 & (R \cap S) \cdot f
\end{array}
$$

Hence, for f simple

$$(R \cap S) \cdot f \ = \ (R \cdot f) \cap (S \cdot f)$$

Taking the converse of both sides, we have dually

$$f^\circ \cdot (R \cap S) \ = \ (f^\circ \cdot R) \cap (f^\circ \cdot S)$$

Furthermore, for simple relations we have the modular law with equality called the *modular identity*. We have,

$$
\begin{array}{ll}
 & (f \cdot R) \cap S \\
\subseteq & \quad \{ \quad \text{modular law} \quad \} \\
 & f \cdot (R \cap (f^\circ \cdot S)) \\
\subseteq & \quad \{ \quad \text{distribution} \quad \} \\
 & (f \cdot R) \cap (f \cdot f^\circ \cdot S) \\
\subseteq & \quad \{ \quad \text{assumption: } f \cdot f^\circ \subseteq id \quad \} \\
 & (f \cdot R) \cap S
\end{array}
$$

Hence, for f simple

$$(\mathsf{f} \cdot \mathsf{R}) \cap \mathsf{S} = \mathsf{f} \cdot (\mathsf{R} \cap (\mathsf{f}^{\circ} \cdot \mathsf{S}))$$

Again, taking the converse of both sides, we have dually

$$(\mathsf{R} \cdot \mathsf{f}^{\circ}) \cap \mathsf{S} = (\mathsf{R} \cap (\mathsf{S} \cdot \mathsf{f})) \cdot \mathsf{f}^{\circ}$$

A rule we use frequently is the so-called *shunting rule*. For function f, we have

$$\mathsf{f} \cdot \mathsf{R} \subseteq \mathsf{S} \quad \equiv \quad \mathsf{R} \subseteq \mathsf{f}^{\circ} \cdot \mathsf{S} \ , \tag{2.19}$$

and dually,

$$\mathsf{R} \cdot \mathsf{f}^{\circ} \subseteq \mathsf{S} \quad \equiv \quad \mathsf{R} \subseteq \mathsf{S} \cdot \mathsf{f} \ . \tag{2.20}$$

We prove the first one:

$$
\begin{array}{ll}
 & \mathsf{f} \cdot \mathsf{R} \subseteq \mathsf{S} \\
\Rightarrow & \{ \quad \text{monotonicity} \ \} \\
 & \mathsf{f}^{\circ} \cdot \mathsf{f} \cdot \mathsf{R} \subseteq \mathsf{f}^{\circ} \cdot \mathsf{S} \\
\Rightarrow & \{ \quad \text{assumption: } \mathsf{id} \subseteq \mathsf{f}^{\circ} \cdot \mathsf{f} \ \} \\
 & \mathsf{R} \subseteq \mathsf{f}^{\circ} \cdot \mathsf{S} \\
\Rightarrow & \{ \quad \text{monotonicity} \ \} \\
 & \mathsf{f} \cdot \mathsf{R} \subseteq \mathsf{f} \cdot \mathsf{f}^{\circ} \cdot \mathsf{S} \\
\Rightarrow & \{ \quad \text{assumption: } \mathsf{f} \cdot \mathsf{f}^{\circ} \subseteq \mathsf{id} \ \} \\
 & \mathsf{f} \cdot \mathsf{R} \subseteq \mathsf{S}
\end{array}
$$

Hence, $\mathsf{f} \cdot \mathsf{R} \subseteq \mathsf{S} \ \equiv \ \mathsf{R} \subseteq \mathsf{f}^{\circ} \cdot \mathsf{S}$. As a matter of fact, relation f is a function *if and only if* the shunting rule (2.19) holds for each R and S.

Note that if we have to prove equality of functions, we only have to show one inclusion, that is,

$$\mathsf{f} \subseteq \mathsf{g} \ \equiv \ \mathsf{f} = \mathsf{g}$$

since using shunting rules (2.19) and (2.20),

$$\mathsf{f} \subseteq \mathsf{g} \ \equiv \ \mathsf{id} \subseteq \mathsf{f}^{\circ} \cdot \mathsf{g} \ \equiv \ \mathsf{g}^{\circ} \subseteq \mathsf{f}^{\circ} \ \equiv \ \mathsf{g} \subseteq \mathsf{f} \ .$$

By definition, in a category, an arrow $\mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}$ is an isomorphism if there exists a arrow $\mathsf{g} : \mathsf{B} \leftarrow \mathsf{A}$ such that

$$\mathsf{g} \cdot \mathsf{f} = \mathsf{id}_{\mathsf{B}} \ \wedge \ \mathsf{f} \cdot \mathsf{g} = \mathsf{id}_{\mathsf{A}} \ .$$

In an allegory, the isomorphisms and bijections coincide. It is trivial to verify that if R is a bijection then R is an isomorphism. (Take $\mathsf{g} := \mathsf{R}^{\circ}$.) For the other way around assume that

$$\mathsf{S} \cdot \mathsf{R} = \mathsf{id}_{\mathsf{B}} \ \wedge \ \mathsf{R} \cdot \mathsf{S} = \mathsf{id}_{\mathsf{A}} \ .$$

We intend to show that $\mathsf{S} = \mathsf{R}^{\circ}$, and so R is a function and a cofunction, hence, a bijection. Recall that for arbitrary relation P that $\mathsf{P} \subseteq \mathsf{P} \cdot \mathsf{P}^{\circ} \cdot \mathsf{P}$. Combining this with the assumption gives us,

$$\mathsf{R}^{\circ} = \mathsf{S} \cdot \mathsf{R} \cdot \mathsf{R}^{\circ} \cdot \mathsf{R} \cdot \mathsf{S} \supseteq \mathsf{S} \cdot \mathsf{R} \cdot \mathsf{S} = \mathsf{S}$$

Hence, $\mathsf{R}^{\circ} \supseteq \mathsf{S}$. Similarly, it follows that $\mathsf{S}^{\circ} \supseteq \mathsf{R}$. Hence, $\mathsf{S} = \mathsf{R}^{\circ}$.

2.4.5 Tabular allegories

We say that an object C and a pair of functions $f : A \leftarrow C$ and $g : B \leftarrow C$ is a *tabulation* of relation $R : A \leftarrow B$ if

$$R = f \cdot g^{\circ} \quad \wedge \quad (f^{\circ} \cdot f) \cap (g^{\circ} \cdot g) = \mathsf{id}_C \ .$$

An allegory is said to be *tabular* if every relation has a tabulation. Note that in general we do not assume tabularity. However, as we will see in subsection 2.5.2, the definition of relational product is precisely the existence of a tabulation of $\top\!\top$.

Allegory Rel is tabular. Given relation $R : A \leftarrow B$, define C to be the subset of the cartesian product $A \times B$ containing the pairs of elements for which $(x\,,y) \in R$. Then the pair of projection functions $\mathsf{outl} : A \leftarrow C$ and $\mathsf{outr} : B \leftarrow C$ is a tabulation of R.

In the remainder of this chapter we define the relational extension of some of the categorical concepts defined in section 2.1. Recall that a relational extension of a categorical concept is something which is defined on all relations but if one takes the restriction to functions, one gets the original categorical concept for the sub-category of total functions. Although we do not assume tabularity in general, we show that for the relational extension of functors and natural transformations, tabularity implies that the relational extension is in some sense canonical.

That tabularity is the key assumption in order to prove the relationship between total functions and relations is not that strange. It enables one to translate a statement about relations into a statement about functions by replacing relations by their corresponding tabulations.

2.4.6 Unit

We say that an object U is a *unit* if id_U is the largest relation of its type, i.e. $\mathsf{id}_U = \top\!\top_{U,U}$, and for every object A there exists a total relation $!_A : U \leftarrow A$. Together with the first requirement it follows that $!_A$ is a function: simplicity follows from $!_A \cdot !_A^{\circ} \subseteq \mathsf{id}_U$ since $!_A \cdot !_A^{\circ} : U \leftarrow U$.

If an allegory has a unit then all tops $\top\!\top_{A,B}$ exist since for all $R : A \leftarrow B$, we have:

$$\begin{array}{ll}
 & R \subseteq !_A^{\circ} \cdot !_B \\
\equiv & \qquad \{ \quad \text{! function, shunting} \quad \} \\
 & !_A \cdot R \cdot !_B^{\circ} \subseteq \mathsf{id}_U \\
\equiv & \qquad \{ \quad \mathsf{id}_U = \top\!\top_{U,U} \quad \} \\
 & \textit{true}
\end{array}$$

Hence, $\top\!\top_{A,B} = !_A^{\circ} \cdot !_B$. In particular, we have $!_A \subseteq \top\!\top_{U,A} = !_U^{\circ} \cdot !_A \subseteq !_A$ since $!_U \subseteq \mathsf{id}_U$. That is $!_A = \top\!\top_{U,A}$. In general we do not have the property that $\top\!\top_{A,B} \cdot \top\!\top_{B,C} = \top\!\top_{A,C}$ for arbitrary objects A, B and C. (For instance, in Rel the lhs is the empty relation if B is the empty set.) However, this property holds if $B = U$ since,

$$\top\!\top_{A,U} \cdot \top\!\top_{U,C} = !_A^{\circ} \cdot !_C = \top\!\top_{A,C} \ .$$

As a matter of fact, the existence of a unit can equally be guaranteed by the requirement

$$\top\!\top_{A,U} \cdot \top\!\top_{U,C} = \top\!\top_{A,C} \;\wedge\; \mathsf{id}_U \supseteq \top\!\top_{U,U} \tag{2.21}$$

The first conjunct expresses that the object U is not the empty type, i.e. contains at least one element. The second conjunct expresses that U contains at most one element. So in Rel, the units are the singleton sets.

We say that an allegory is *unitary* if it has a unit and we denote this unit by 1.

If allegory $\mathcal{A}$ has a unit 1 then this unit is terminal in the sub-category *Map*$(\mathcal{A})$. Since $!_A = \top\!\top_{1,A}$ it follows for function $f : 1 \leftarrow A$ that $!_A \supseteq f$. Hence, $!_A = f$ since $!_A$ is a function too. For a tabular allegory, the other way around is also true: if 1 is terminal in the sub-category *Map*$(\mathcal{A})$ then 1 is a unit of $\mathcal{A}$. We have to show that id_1 is the largest relation of its type. Let $(f\,,\,g)$ be a tabulation of relation R of type $1 \leftarrow 1$. Then,

$$
\begin{array}{cl}
 & R \subseteq \mathsf{id}_1 \\
\equiv & \{ \quad R = f \cdot g^\circ \quad \} \\
 & f \cdot g^\circ \subseteq \mathsf{id}_1 \\
\equiv & \{ \quad \text{shunting, } f,\ g \text{ functions} \quad \} \\
 & f = g \\
\equiv & \{ \quad f\,,\,g : 1 \leftarrow C, \text{ terminality of } 1 \quad \} \\
 & \textit{true}
\end{array}
$$

The definition of a unit is the first example of a relational extension of a categorical concept for functions. Without using tabularity, it follows that if we restrict our attention to functions, i.e. the sub-category *Map*, we get the normal categorical definition. For the other way around, we have to assume tabularity to show that the relational extension of a categorical concept on functions is in some sense canonical.

2.4.7 Range and domain

Associated with every relation $R : A \leftarrow B$ there are two partial identities, the *domain* of R, denoted by $R{>} : B \leftarrow B$ and the *range* of R, denoted by $R{<} : A \leftarrow A$.

The domain operator is defined by the following universal property [1],

$$R \subseteq \top\!\top_{A,B} \cdot X \;\equiv\; R{>} \subseteq X \quad \text{for each } X \subseteq \mathsf{id}_B. \tag{2.22}$$

The interpretation of the domain operator is that it represents the set of values for which there exists an image, i.e.

$$x \in R{>} \;\equiv\; \exists(y :: y\ R\ x)\ .$$

Note that the universal property is a Galois connection. Hence, we know immediately that "$>$" is monotonic. Dually, the range operator is defined by the universal property,

$$R \subseteq X \cdot \top\!\top_{A,B} \;\equiv\; R{<} \subseteq X \quad \text{for each } X \subseteq \mathsf{id}_A.$$

Note that $R> = (R^\circ)<$. From now on we concentrate on the domain operator; the properties of the range operator follow by duality.

The domain operator can equally well be defined by the following universal property

$$R = R \cdot X \;\equiv\; R> \subseteq X \quad \text{for each } X \subseteq \mathsf{id}_B . \tag{2.23}$$

In other words, $R>$ is the least partial identity which is a domain of R. The equivalence between (2.22) and (2.23) follows from the fact that (for all partial identities X)

$$R \subseteq \top\!\top \cdot X \;\equiv\; R = R \cdot X$$

which is verified using the modular identity.

There are closed formulae for the domain and range operator. For the domain operator we have

$$R> = (R^\circ \cdot R) \cap \mathsf{id} \tag{2.24}$$

Again, using the modularity law it is not too difficult to verify that for the closed formula (2.24), the universal property (2.23) holds.

For the domain operator, we have the following properties [1, 7],

$$(R \cdot S)> = (R> \cdot S)> \;, \tag{2.25}$$

$$(R \cdot S)> \subseteq S> \;, \tag{2.26}$$

$$(R \cap S)> = (R^\circ \cdot S) \cap \mathsf{id} \tag{2.27}$$

One of the cancellation laws of the Galois connection for domains is $R \subseteq \top\!\top_{A,B} \cdot R>$. Composing both sides with $\top\!\top_{C,A}$ gives us $\top\!\top_{C,A} \cdot R \subseteq \top\!\top_{C,B} \cdot R>$, since $\top\!\top_{C,A} \cdot \top\!\top_{A,B} \subseteq \top\!\top_{C,B}$. The converse inclusion follows from (2.24),

$$\top\!\top_{C,B} \cdot R> \;\subseteq\; \top\!\top_{C,B} \cdot R^\circ \cdot R \;\subseteq\; \top\!\top_{C,A} \cdot R \;.$$

Hence, we have the property, for $R : A \leftarrow B$ and object C,

$$\top\!\top_{C,A} \cdot R = \top\!\top_{C,B} \cdot R> \;. \tag{2.28}$$

Partial identities versus conditions Instead of representing subsets of object A by partial identities of id_A, they can equally well represented by so-called conditions. A *right-condition* is a relation below $!_A$, i.e. a relation of type $1 \leftarrow A$. Dually, a relation R is a *left-condition* if R° is a right-condition. The order isomorphism between partial identities and conditions is given by the following equation, for all partial identities $X : A \leftarrow A$ and (right) conditions $C : 1 \leftarrow A$,

$$X = C> \;\equiv\; !_A \cdot X = C$$

since we have the properties,

$$(! \cdot X)> = X \quad \wedge \quad ! \cdot C> = C$$

The first conjunct follows from (2.25) and $X> = X$, the second conjunct follows from (2.28) and $!_1 = \mathrm{id}_1$.

So, instead of representing range and domain as partial identities one could represent them using conditions. From the correspondence between partial identities and conditions and property (2.28) it follows that the right-condition representation of the domain is $! \cdot R$. Sometimes it is more handy to use conditions instead of the partial identities. Note that we have,

$$R> \subseteq S> \quad \equiv \quad ! \cdot R \subseteq ! \cdot S$$

2.4.8 Power allegories

Instead of using binary relations, one could work with set-valued functions in view of the theorem that, for each relation R, there exists a set-valued function f such that $x\ R\ y$ iff $x \in f(y)$. In this section we give a calculational presentation of this isomorphism between binary relations and set-valued functions.

Power transpose and membership

The object which represents the set of subsets of object A, called the *power object*, we denote by PA. The set-valued function corresponding to R, called the *power transpose*, is denoted by ΛR. That is to say, for each relation $R : A \leftarrow B$, ΛR is a function of type $PA \leftarrow B$. The membership relation corresponding with the power-object PA we denote by $\in_A\ :\ A \leftarrow PA$. For these three notions we demand the following universal property, for all $R : A \leftarrow B$ and functions $f\ :\ PA \leftarrow B$,

$$f = \Lambda R \quad \equiv \quad \in_A \cdot f = R \ . \tag{2.29}$$

The universal property implies the following two cancellation laws: $\in \cdot \Lambda R = R$, and for function $f\ :\ PA \leftarrow B$, $\Lambda(\in_A \cdot f) = f$. Hence, Λ is indeed an isomorphism.

Using the first cancellation law, it follows that we have the following distribution property $\Lambda R \cdot f = \Lambda(R \cdot f)$ for function f, since

$$
\begin{array}{ll}
 & \Lambda R \cdot f = \Lambda(R \cdot f) \\
\equiv & \quad \{ \quad f \text{ function, hence, } \Lambda R \cdot f \text{ function, (2.29)} \quad \} \\
 & \in \cdot \Lambda R \cdot f = R \cdot f \\
\equiv & \quad \{ \quad \text{cancellation} \quad \} \\
 & \textit{true}
\end{array}
$$

Existential image

As the notation PA for power objects already suggests, we can extend P to a functor on relations. We define the *existential image* functor by,

$$ER \quad \triangleq \quad \Lambda(R \cdot \in_B)\ :\ PA \leftarrow PB \quad \text{for each } R : A \leftarrow B.$$

The interpretation of $\mathsf{E}R$ is

$$a \in (\mathsf{E}R)(x) \equiv \exists(b\colon b \in x :\ a\, R\, b)$$

The fact that E distributes over composition follows trivially from the following fusion law,

$$\mathsf{E}R \cdot \Lambda S \;=\; \Lambda(R \cdot S) \ ,$$

whose validity is verified using distribution and cancellation as follows,

$$\mathsf{E}R \cdot \Lambda S \;=\; \Lambda(R \cdot \in) \cdot \Lambda S \;=\; \Lambda(R \cdot \in \cdot \Lambda S) \;=\; \Lambda(R \cdot S) \ .$$

The fact that E respects identities, i.e. $\Lambda{\in} = \mathsf{id}$, is easily verified using the universal property (2.29). So, E is a functor on relations.

2.4.9 Locally complete allegories

The definition of an allegory only mentions the intersection of two relations. In allegory Rel, the union of arbitrary relations also exists. We say that an allegory is *locally complete* if for each set $\mathcal{S}$ of relations of type $A \leftarrow B$, the *union* (join) $\cup\mathcal{S} : A \leftarrow B$ exists, defined by the following universal property, for all $X : A \leftarrow B$,

$$\cup\mathcal{S} \subseteq X \equiv \forall(S \in \mathcal{S} :: S \subseteq X) \ ,$$

and intersection and composition distribute over arbitrary unions. That is,

$$(\cup\mathcal{S}) \cap R = \cup\{S \cap R \mid S \in \mathcal{S}\} \ ,$$

$$R \cdot (\cup\mathcal{S}) = \cup\{R \cdot S \mid S \in \mathcal{S}\} \ .$$

and

$$(\cup\mathcal{S}) \cdot R = \cup\{S \cdot R \mid S \in \mathcal{S}\} \ .$$

The union of the empty set, i.e. smallest relation of type $A \leftarrow B$, we denote by $\perp\!\!\!\perp_{A,B}$. Furthermore, for the union of two relations we use the infix notion, i.e. $\cup\{R\,,S\} = R \cup S$. By taking $\mathcal{S}$ to be the empty set it follows that $\perp\!\!\!\perp$ is a zero both of composition and intersection. It is trivial to verify that union is idempotent, symmetric and associative.

Since converse and domains are both defined as a lower Galois adjoint it follows that they both distribute over arbitrary unions. Specifically, we have $\perp\!\!\!\perp^\circ = \perp\!\!\!\perp$, $(R \cup S)^\circ = R^\circ \cup S^\circ$, $\perp\!\!\!\perp{>} = \perp\!\!\!\perp$, and $(R \cup S){>} = R{>} \cup S{>}$.

2.4.10 Division

In this section we define the so-called *division* operators. (Others use the terms “residual” [11], “factor” [10], or “weakest pre-specification” [19].) By definition, we define “\” and “/” as the upper-adjoint of composition on the left and on the right, respectively. That is to say, we assume the following two Galois-connections, for $R : A \leftarrow B$, $S : B \leftarrow C$ and $T : A \leftarrow C$,

$$R \cdot S \subseteq T \quad \equiv \quad S \subseteq R\backslash T \ ,$$

and

$$R \cdot S \subseteq T \;\equiv\; R \subseteq T/S \;.$$

A relation of the shape $R\backslash T$ or T/S we call a *factor* of T. Note that $R\backslash T : B \leftarrow C$ and $T/S : A \leftarrow B$. The interpretation of the factors is

$$b\,(R\backslash T)\,c \;\equiv\; \forall(a\colon a\,R\,b\colon a\,T\,c) \;,$$

and

$$a\,(T/S)\,b \;\equiv\; \forall(c\colon b\,S\,c\colon a\,T\,c) \;.$$

For division we have the following cancellation rules,

$$R \cdot (R\backslash T) \subseteq T \;\wedge\; (T/S) \cdot S \subseteq T \;.$$

These cancellation rules are the reason why we call the upper Galois adjoint division.

Since $(R\backslash)$ and $(/S)$ are the upper adjoints of $(R\,\cdot)$ and $(\cdot\,R)$, respectively, they distribute over arbitrary intersection. In particular,

$$R\backslash(S \cap T) = (R\backslash S) \cap (R\backslash T) \;\wedge\; (R \cap T)/S = (R/S) \cap (T/S) \;.$$

Furthermore, combining both Galois-connections yields the following Galois-connection:

$$S \subseteq R\backslash T \;\equiv\; R \subseteq T/S \;. \tag{2.30}$$

To see why this is a Galois-connection, define $\preceq\, = \,\supseteq$, then equation (2.30) can be rewritten as,

$$S \subseteq R\backslash T \;\equiv\; T/S \preceq R \;. \tag{2.31}$$

Hence, $(\backslash T)$ and $(T/)$ are Galois-connected. Thus $(\backslash T)$ is $\subseteq$ to $\preceq$ monotonic and thus anti-monotonic, since $\preceq\, = \,\supseteq$. Similarly, $(\backslash T)$ takes $\preceq$-meets, i.e. $\subseteq$-joins, to $\subseteq$-meets. The same holds for $(T/)$. In particular, we have,

$$(R \cup S)\backslash T = (R\backslash T) \cap (S\backslash T) \;\wedge\; T/(R \cup S) = (T/R) \cap (T/S) \;.$$

Since composition is associative we have the following properties for factors,

$$R\backslash(S\backslash T) = (S \cdot R)\backslash T \;\wedge\; (R/S)/T = R/(T \cdot S) \;,$$

and,

$$R\backslash(S/T) = (R\backslash S)/T \;.$$

2.4.11 Extensionality

The last axiom we define is the so-called extensionality axiom. Note that our axiomatization of the relational calculus was point-free. Only when we gave an interpretation of a relation did we talk about elements (points), e.g. the elements related by a relation. The extensionality axiom enables us to give pointwise reasonings. First, we define the notion of a point.

Definition 2.32 We say that p is a *point* if relation p is a function of type $A \leftarrow 1$ for some A. We write $p \in A$ if p is a point of type $A \leftarrow 1$.
□

Recall that the interpretation of the object 1 is the singleton set $\{1\}$. So, since we require a point to be function, it means that point p is a function mapping 1 to some fixed element of object A. In Rel, binary relations of the form $\{(x\,,1)\}$, for some x of the set A, are points. In other words, points correspond to the notion of an element of a set. This explains the notation $p \in A$. We unify an element and its corresponding point.

Now we can state the *extensionality* [46, 44] axiom. That is, we assume, for $R\,,S : A \leftarrow B$, that

$$R \supseteq S \quad \equiv \quad R\cdot p \supseteq S\cdot p \quad \text{for all } p \in B.$$

or, equivalently,

$$R = S \quad \equiv \quad R\cdot p = S\cdot p \quad \text{for all } p \in B.$$

Note that for point x, $R\cdot x$ corresponds to the image set of R with respect to the element x. That is to say, $R\cdot x$ is the set $\{(y\,,1) \mid y\ R\ x\}$. In other words, the extensionality axiom expresses that two relations are equal iff the image sets of both relations with respect to all elements are equal. Compare this with the normal version of extensionality for functions,

$$f = g \quad \equiv \quad f(x) = g(x) \quad \text{for all } x.$$

Trivially, the extensionality axiom holds in Rel. In general, we will not assume the extensionality axiom. We only use the extensionality axiom if we want to calculate with the interpretation of relations. Note that we have the correspondence,

$$x\,R\,y \quad \equiv \quad x \subseteq R\cdot y \;\; (\equiv x\cdot y^{\circ} \subseteq R) \;\; .$$

2.4.12 Operator precedence

Now that we have introduced all the primitive operators in our calculus it is useful to give each a precedence in order to minimise parentheses.

A general rule we use is that metaoperators ("$\wedge$", "$=$", "$\leftarrow$" etc.) have the lowest precedence. Among these operators the precedence rules we use conform to standard mathematical practice. The highest precedence is given to subscripting followed by prefix and postfix operators (converse being the only instance so far) and application and composition of mappings (both denoted by juxtaposition). The object level infix operators have the following precedence from highest to lowest:

- division,
- split, junc, product and coproduct,
- composition,
- intersection and union.

For example, the expression $f \triangledown g \cdot h$ is parsed as $(f \triangledown g) \cdot h$ and $R \cap S \cdot T$ as $R \cap (S \cdot T)$. (Note that we endeavour to space the subexpressions in such a way that the correct structure is evident to the human eye.)

From time to time parentheses will be introduced even though this is not strictly necessary. In particular, when giving basic laws it is our practice to include parentheses. (See for example (2.41).)

2.5 Relational extensions

In this section we extend the notion of functors to so-called *relators*. A relator is a monotonic functor on relations which commutes with converse. A main property of relators is that they preserve functions. Hence, a relator is a relational extension of a functor on functions. As observed by De Moor [40], this extension is unique for a tabular allegory. That is to say, for every functor on functions, there exists at most one relator which agrees on functions with the original functor.

We introduce relational coproduct and product and their corresponding relators. Relational coproduct is the real categorical coproduct whereas this is only the case for relational product if we take the restriction to functions.

We also define the relational extension of the notion of a natural transformation between functors. And we define the notion of a relational catamorphism. Furthermore, we give the relational extension of the power functor.

For all the relational extensions of categorical constructs we give in this section, it is the case that when we take the restriction to functions, we get the normal categorical constructions on functions. We argued that we wanted to move from functions to relations in order to increase our powers of expression. Furthermore, one can employ relations to specify nondeterministic problems. For instance, consider the case that we have a nondeterministic specification of a problem and we want to find a functional solution. In our calculation we can freely use relations in combination with the relational extension of categorical constructs. Now, if we know that all the components of our final derivation are functions[2], the "program" can be "implemented" using the normal categorical constructions on the functions.

The way we introduce the relational extensions of categorical concepts is inspired by the way it is done in the book by Bird and de Moor [7] although there they assume tabularity. Furthermore, for relational coproduct and relational catamorphisms, they assume the existence of power transpose and membership (see subsection 2.4.8).We will not assume tabularity in general nor the existence of power transpose and membership.

[2]and these functions have a straightforward implementation

2.5.1 Relators

We start with the relational extension of a functor on functions,

Definition 2.33 (Relator) A *relator* is a monotonic functor which commutes with converse. That is, let $\mathcal{A}$ and $\mathcal{B}$ be allegories. Then the mapping $\mathsf{F} : \mathcal{A} \leftarrow \mathcal{B}$ is a relator iff,

$$\mathsf{F}R \cdot \mathsf{F}S = \mathsf{F}(R \cdot S) \quad \text{for each } R : A \leftarrow B \text{ and } S : B \leftarrow C, \tag{2.34}$$

$$\mathsf{F}id_A = id_{\mathsf{F}A} \quad \text{for each object } A, \tag{2.35}$$

$$\mathsf{F}R \subseteq \mathsf{F}S \quad \Leftarrow \quad R \subseteq S \ , \tag{2.36}$$

$$(\mathsf{F}R)^\circ = \mathsf{F}(R^\circ) \ . \tag{2.37}$$

□

Note that it is safe to write $\mathsf{F}R^\circ$ which means either $(\mathsf{F}R)^\circ$ or $\mathsf{F}(R^\circ)$. If allegory $\mathcal{B}$ is tabular, a functor is monotonic iff it commutes with converse [7]. So, if we define a relator on a tabular allegory, one has to prove either requirement (2.36) or (2.37). However, since we do not assume tabularity, we prove both properties (2.36) and (2.37) whenever we define a relator.

The interpretation of $\mathsf{F}R : \mathsf{F}A \leftarrow \mathsf{F}B$, for $R : A \leftarrow B$, is that when we apply $\mathsf{F}R$ to F-structure x, the relation R is applied to all the elements of x. For instance, as we will see, the product and the list functors can be extended to relators. For the interpretation of relational product, we have $(a\,,b)\ R \times S\ (c\,,d)$ iff $a\ R\ c$ and $b\ S\ d$. And for lists,

$$x\ (\mathit{List}R)\ y \quad \equiv \quad \#.x = \#.y \quad \wedge \quad \forall(i :: x_i\ R\ y_i)$$

In general, we have $x\ \mathsf{F}R\ y$ whenever F-structures x and y have the same shape and the corresponding elements of x and y are related by R. At the end of this section, we will formally prove that $\mathsf{F}R$ respects shapes.

Preservation of functions In general, a functor between allegories does not preserve functions. However, using the extra requirements for a relator (2.36) and (2.37), it follows that relators preserve simple and total relations, hence, respect functions. For simple relation f, we have,

$$\mathsf{F}f \cdot \mathsf{F}f^\circ = \mathsf{F}(f \cdot f^\circ) \subseteq \mathsf{F}id = id$$

hence, $\mathsf{F}f$ is simple, and preservation of totality follows from, for total relation R,

$$\mathsf{F}R^\circ \cdot \mathsf{F}R = \mathsf{F}(R^\circ \cdot R) \supseteq \mathsf{F}id = id$$

From the fact that a relator respects functions it follows that every relator $\mathsf{F} : \mathcal{A} \leftarrow \mathcal{B}$ is a functor between the corresponding sub-category of total functions, i.e. $\mathsf{F} : \mathit{Map}(\mathcal{A}) \leftarrow \mathit{Map}(\mathcal{B})$. In other words, every relator is a relational extension of a functor on functions. If we assume tabulations, this extension is in some sense canonical. That is to say, for every functor on *Map*, there exists at most one relator which agrees on functions with the original functor. This is expressed by the following lemma:

Lemma 2.38 Let F and G be two relators which agree on functions, i.e. $Ff = Gf$ for all functions f. Then $F = G$.

Proof Let R be an arbitrary relation with tabulation (f, g). Then

$$FR = F(f \cdot g^\circ) = Ff \cdot Fg^\circ = Gf \cdot Gg^\circ = GR$$

□

As we will see in subsection 2.5.2, the product and coproduct functor on functions can be extended to a relator. Some functors exist for which there is no relational extension. However, all regular functors have a relational extension. A precise condition is given in [40] for when a functor has a relational extension in the case that the allegory is tabular.

Shapes Another corollary of the fact that relators respect functions, is that the relation FR respects shapes. That is to say, for $R : A \leftarrow B$,

$$x\,FR\,y \quad\Rightarrow\quad F!_A \cdot x = F!_B \cdot y\ .$$

Recall that the shape of $x \in FA$ is defined as $F!_A \cdot x \in F1$. We calculate, for $R : A \leftarrow B$, and points $x \in FA$, $y \in FB$,

$$\begin{array}{ll}
 & x\,FR\,y \\
\equiv & \{ \quad \text{definition} \quad \} \\
 & x \cdot y^\circ \subseteq FR \\
\Rightarrow & \{ \quad \text{monotonicity, } R \subseteq \top\!\!\top_{A,B} = !_A^\circ \cdot !_B \quad \} \\
 & x \cdot y^\circ \subseteq F(!_A^\circ \cdot !_B) \\
\equiv & \{ \quad F \text{ relator, shunting of functions} \quad \} \\
 & F!_A \cdot x \subseteq F!_B \cdot y \\
\equiv & \{ \quad \text{functions} \quad \} \\
 & F!_A \cdot x = F!_B \cdot y
\end{array}$$

Thus if x and y are related by FR they have the same shape

Note that not all functors on relations respect shape. An example is existential image E. Although E is a functor on relations, it is not monotonic, hence, it is not a relator. The shape of sets is given by $E!$ which maps the empty set to the empty set and a non-empty set to the singleton set containing the unit element. Hence, the shape of a set, i.e. an E-structure, is the property whether or not the set is empty. Now, $E\bot\!\!\bot = \Lambda(\bot\!\!\bot \cdot \in) = \Lambda\bot\!\!\bot$ is an example of a relation which does not preserve shapes: $\Lambda\bot\!\!\bot$ maps every set to the empty set, hence, the shape of a non-empty set is not preserved.

Domains

We have proven that relators preserve total relations but we can do better. We can prove that relators commute with the domain operator. That is, $F(R>) = (FR)>$. From the fact that a relator respects identities and is monotonic it follows that a relator respects partial identities too. Now, using the isomorphism between partial identities and conditions, it follows that relators commute with the domain operator if we show that,

$$! \cdot F(R>) \;=\; ! \cdot (FR)> \;.$$

Since ! is a function and $!_{FA} = \top\!\top_{1,FA}$ it follows that $!_{FA} = !_{F1} \cdot F!_A$. Using this, we calculate, for $R : B \leftarrow A$,

$$
\begin{array}{ll}
 & !_{FA} \cdot F(R>) \\
= & \qquad \{ \qquad \text{property above, } F \text{ functor } \} \\
 & !_{F1} \cdot F(!_A \cdot R>) \\
= & \qquad \{ \qquad \text{domains (2.28): } !_A \cdot R> = !_B \cdot R \;\} \\
 & !_{F1} \cdot F(!_B \cdot R) \\
= & \qquad \{ \qquad \text{property above, } F \text{ functor } \} \\
 & !_{FB} \cdot FR \\
= & \qquad \{ \qquad \text{domains (2.28) } \} \\
 & !_{FA} \cdot (FR)>
\end{array}
$$

Hence, relators commute with the domain operator. Since relators commute with converse it follows that relators commute with the range operator as well. Thus it is safe to write $FR>$ meaning either $F(R>)$ or $(FR)>$.

2.5.2 Coproduct and product

In this section we introduce relational coproduct and product and define the corresponding relators. We begin with relational coproduct because this one is easier. Relational coproduct will be the real categorical coproduct on relations. We try to define relational product more or less as the dual of relational coproduct. Relational product will not be a real categorical product. However, if we take the restriction to total functions, relational product is a real categorical product. So, not only is the product relator the relational extension of the product functor on the sub-category *Map*, for relational product we also have in *Map* the usual universal property for products. This holds for coproduct too. In other words, the definition of relational coproduct and product are in some sense canonical extensions of the categorical definitions for total functions.

The way we introduce coproduct and product is inspired by [17] and [7], although we do not assume tabulations for every relation. In [1] relational coproduct and product are introduced in a different but equivalent way.

Coproducts

A *coproduct* of two objects consists of an object and two *injection* relations. The object is denoted by $A + B$ and the two relations by $\mathsf{inl}_{A,B} \;:\; A{+}B \leftarrow A$ and $\mathsf{inr}_{A,B} \;:\; A{+}B \leftarrow B$. For the injection relations we require that

$$\mathsf{inl}^{\circ}_{A,B} \cdot \mathsf{inl}_{A,B} = \mathsf{id}_A \quad \text{and} \quad \mathsf{inr}^{\circ}_{A,B} \cdot \mathsf{inr}_{A,B} = \mathsf{id}_B \tag{2.39}$$

$$\mathsf{inl}^{\circ}_{A,B} \cdot \mathsf{inr}_{A,B} = \perp\!\!\perp_{A,B} \tag{2.40}$$

and

$$(\mathsf{inl}_{A,B} \cdot \mathsf{inl}^{\circ}_{A,B}) \cup (\mathsf{inr}_{A,B} \cdot \mathsf{inr}^{\circ}_{A,B}) = \mathsf{id}_{A+B} \tag{2.41}$$

Equation (2.39) expresses that inl and inr are injective and total; from (2.41) it follows that they are simple too, hence, they are injective functions. Furthermore, from equation (2.40) it follows that inl and inr have disjoint ranges but from equation (2.41) it follows that the union of both ranges is id_{A+B}.

Having the functions inl and inr, we can define the *junc* operator, for $R : C \leftarrow A$ and $S : C \leftarrow B$,

$$R \triangledown S \;\triangleq\; (R \cdot \mathsf{inl}^{\circ}_{A,B}) \cup (S \cdot \mathsf{inr}^{\circ}_{A,B}) \;:\; C \leftarrow A + B\ . \tag{2.42}$$

Since junc is defined as a union and inl and inr are functions, it follows that we have the following universal property. For all $R : C \leftarrow A$, $S : C \leftarrow B$ and $X : C \leftarrow A + B$,

$$R \triangledown S \subseteq X \quad \equiv \quad R \subseteq X \cdot \mathsf{inl}_{A,B} \wedge S \subseteq X \cdot \mathsf{inr}_{A,B} \tag{2.43}$$

since for the rhs of above equation we have

$$\begin{array}{ll} & R \subseteq X \cdot \mathsf{inl} \wedge S \subseteq X \cdot \mathsf{inr} \\ \equiv & \quad \{ \quad \mathsf{inl} \text{ and } \mathsf{inr} \text{ are functions, shunting } \} \\ & R \cdot \mathsf{inl}^{\circ} \subseteq X \wedge S \cdot \mathsf{inr} \subseteq X \\ \equiv & \quad \{ \quad \text{universal property of union } \} \\ & R \cdot \mathsf{inl}^{\circ} \cup S \cdot \mathsf{inr}^{\circ} \subseteq X \end{array}$$

Hence, the universal property (2.43) follows from indirect equality. As a matter of fact, $R \triangledown S$ could be defined by the universal property (2.43) instead of the closed formula (2.42). The universal property (2.43) expresses that $R \triangledown S$ is the smallest relation for which the rhs holds. Specifically, we have the following computation rules

$$R \subseteq R \triangledown S \cdot \mathsf{inl}_{A,B} \;\wedge\; S \subseteq R \triangledown S \cdot \mathsf{inr}_{A,B}$$

However, we can do better. Using requirement (2.39) and (2.40) it follows that

$$R = R \triangledown S \cdot \mathsf{inl}_{A,B} \;\wedge\; S = R \triangledown S \cdot \mathsf{inr}_{A,B} \tag{2.44}$$

since, for instance,

$$R \triangledown S \cdot \mathsf{inl} = (R \cdot \mathsf{inl}^{\circ} \cup S \cdot \mathsf{inr}^{\circ}) \cdot \mathsf{inl} = R \cdot \mathsf{inl}^{\circ} \cdot \mathsf{inl} \cup S \cdot \mathsf{inr}^{\circ} \cdot \mathsf{inl} = R$$

Using this fact, we prove that we have a dual universal property for coproduct: that is to say, the universal property (2.43) with the inclusion the other way around. To begin, since composition distributes over union it follows from (2.41) that for $X : C \leftarrow A + B$,

$$X = X \cdot \mathsf{inl} \cdot \mathsf{inl}^{\circ} \cup X \cdot \mathsf{inr} \cdot \mathsf{inr}^{\circ} = (X \cdot \mathsf{inl}) \triangledown (X \cdot \mathsf{inr})\ . \tag{2.45}$$

Instantiating $X := R \triangledown S$ and $R := X \cdot \mathsf{inl}$ and $S := X \cdot \mathsf{inr}$ in the universal property (2.43) yields

$$(X \cdot \mathsf{inl}) \triangledown (X \cdot \mathsf{inr}) \subseteq R \triangledown S \quad \equiv \quad X \cdot \mathsf{inl} \subseteq R \triangledown S \cdot \mathsf{inl} \;\; \wedge \;\; X \cdot \mathsf{inr} \subseteq R \triangledown S \cdot \mathsf{inr}$$

Now, using fact (2.45) and the computation rules (2.44) it follows that we have the universal property:

$$X \subseteq R \triangledown S \quad \equiv \quad X \cdot \mathsf{inl}_{A,B} \subseteq R \;\; \wedge \;\; X \cdot \mathsf{inr}_{A,B} \subseteq S \tag{2.46}$$

In other words, $R \triangledown S$ is the largest relation for which the rhs of (2.46) holds. This suggests that $R \triangledown S$ can be expressed as an intersection:

$$\begin{array}{ll} & X \cdot \mathsf{inl} \subseteq R \wedge X \cdot \mathsf{inr} \subseteq S \\ \equiv & \{ \quad \text{factors} \quad \} \\ & X \subseteq R/\mathsf{inl} \wedge X \subseteq S/\mathsf{inr} \\ \equiv & \{ \quad \text{intersection} \quad \} \\ & X \subseteq R/\mathsf{inl} \cap S/\mathsf{inr} \end{array}$$

Hence, using indirect equality it follows from (2.46) that

$$R \triangledown S = R/\mathsf{inl} \cap S/\mathsf{inr} \tag{2.47}$$

Combining universal properties (2.43) and (2.46), it follows that $(A + B\,, \mathsf{inl}_{A,B}\,, \mathsf{inr}_{A,B})$ is a real categorical coproduct, i.e. we have the universal property

$$X = R \triangledown S \quad \equiv \quad X \cdot \mathsf{inl}_{A,B} = R \;\; \wedge \;\; X \cdot \mathsf{inr}_{A,B} = S \tag{2.48}$$

As a matter of fact, we could have defined coproduct equivalently starting with the above universal property and deriving properties (2.39) – (2.41) from it [17]. However, since this is not possible for product and we want to see more or less the dual between coproduct and product, we have chosen not to do so.

Since relational coproduct is a real categorical coproduct it follows that, for $R : A \leftarrow C$ and $S : B \leftarrow D$,

$$R + S \;\; \triangleq \;\; (\mathsf{inl}_{A,B} \cdot R) \triangledown (\mathsf{inr}_{A,B} \cdot S) \;:\; A + B \leftarrow C + D \;\;,$$

defines a functor. Furthermore, we have the properties that

$$R \cdot (S \triangledown T) = (R \cdot S) \triangledown (R \cdot T)$$

and

$$(R+S) \cdot (T \triangledown U) = (R \cdot T) \triangledown (S \cdot U) \;\; .$$

Since "$\triangledown$" is monotonic, it follows that "$+$" is monotonic too. Furthermore, it is easy to verify that the operator "$+$" commutes with converse, hence, "$+$" is a relator.

From the fact that "$+$" is a relator, it follows that "$+$" respects functions. Furthermore, $f \triangledown g = (f + g) \cdot (\mathsf{id} \triangledown \mathsf{id})$ is a function, if $\mathsf{id} \triangledown \mathsf{id}$ is a function too. That $\mathsf{id} \triangledown \mathsf{id}$ is indeed a function is easy to verify using properties (2.39) – (2.41). Hence, relation coproduct is not only a real categorical coproduct for an allegory, it is also a coproduct for the sub-category *Map*.

Product

A *product* of two objects consists of an object and two projection arrows. The object is denoted by $A \times B$ and the two arrows by $\mathsf{outl}_{A,B} : A \leftarrow A \times B$ and $\mathsf{outr}_{A,B} : B \leftarrow A \times B$. For the arrows we require them to be functions and that

$$\mathsf{outl}_{A,B} \cdot \mathsf{outr}^{\circ}_{A,B} = \top\!\!\top_{A,B} \tag{2.49}$$

and

$$\mathsf{outl}^{\circ}_{A,B} \cdot \mathsf{outl}_{A,B} \;\cap\; \mathsf{outr}^{\circ}_{A,B} \cdot \mathsf{outr}_{A,B} = \mathsf{id}_{A \times B} \tag{2.50}$$

In other words, we require $(\mathsf{outl}, \mathsf{outr})$ to be a tabulation of $\top\!\!\top$. Furthermore, recalling that for a unitary allegory each top exists, we conclude that a unitary tabular allegory has relational products. Note, however, that for the existence of relational product, we do not need to assume the existence of tabulations for every relation. Equation (2.49) expresses that all elements a and b can be paired, and equation (2.50) expresses that such a pairing is unique.

Having the projection functions outl and outr, we can define the split operator on relations,for $R : A \leftarrow C$ and $S : B \leftarrow C$,

$$R \vartriangle S \triangleq (\mathsf{outl}^{\circ}_{A,B} \cdot R \;\cap\; \mathsf{outr}^{\circ}_{A,B} \cdot S) \;:\; A \times B \leftarrow C \;\;, \tag{2.51}$$

Since split is defined as an intersection and outl and outr are functions, it follows that we have the following universal property, for all $X : A \times B \leftarrow C$, $R : A \leftarrow C$ and $S : B \leftarrow C$

$$X \subseteq R \vartriangle S \quad \equiv \quad \mathsf{outl}_{A,B} \cdot X \subseteq R \wedge \mathsf{outr}_{A,B} \cdot X \subseteq S \tag{2.52}$$

In detail, the proof of (2.52) is as follows. We have for the rhs of above equation

$$\begin{array}{ll}
& \mathsf{outl} \cdot X \subseteq R \wedge \mathsf{outr} \cdot X \subseteq S \\
\equiv & \quad \{ \quad \mathsf{outl}, \mathsf{outr} \text{ functions, shunting } \} \\
& X \subseteq \mathsf{outl}^{\circ} \cdot R \wedge X \subseteq \mathsf{outr}^{\circ} \cdot S \\
\equiv & \quad \{ \quad \text{universal property intersection } \} \\
& X \subseteq \mathsf{outl}^{\circ} \cdot R \cap \mathsf{outr}^{\circ} \cdot S
\end{array}$$

Hence, the universal property (2.52) follows from indirect equality. As a matter of fact, $R \vartriangle S$ could be defined by the universal property (2.52) instead of the closed formula (2.51). The universal property (2.52) expresses that $R \vartriangle S$ is the largest relation for which the rhs holds. Specifically, we have the following computation rules

$$\mathsf{outl} \cdot R \vartriangle S \subseteq R \;\wedge\; \mathsf{outr} \cdot R \vartriangle S \subseteq S$$

For the coproduct, we have computation rules (2.44) with equality. One can not expect equality for product as well. For instance, from the definition of "$\vartriangle$" it follows that $R \vartriangle \perp\!\!\perp = \perp\!\!\perp$, hence, $\mathsf{outl} \cdot R \vartriangle \perp\!\!\perp = \perp\!\!\perp$. This means that relational product is *not* a categorical product. Although in general we do not have computation rules with equality, we do have equality if we add a domain restriction. That is to say, we have the following rules

$$\mathsf{outl} \cdot R \vartriangle S = R \cdot S_{>} \;\wedge\; \mathsf{outr} \cdot R \vartriangle S = S \cdot R_{>} \tag{2.53}$$

since, for instance, using the modular identity and the fact that outl is a function

$$\mathsf{outl} \cdot (\mathsf{outl}^\circ \cdot R \cap \mathsf{outr}^\circ \cdot S) = R \cap \mathsf{outl} \cdot \mathsf{outr}^\circ \cdot S = R \cap \top\!\top \cdot S = R \cdot S_>$$

So, if we take the restriction to total relations, we have computation rules with equality. That is, for total relations R and S,

$$\mathsf{outl} \cdot R \vartriangle S = R \wedge \mathsf{outr} \cdot R \vartriangle S = S \tag{2.54}$$

For product, the dual of property (2.45), i.e. for $X : A \times B \leftarrow C$,

$$X = (\mathsf{outl} \cdot X) \vartriangle (\mathsf{outr} \cdot X)$$

does not hold in general since split is defined as an intersection and arbitrary relations do not distribute over intersection. However, if X is simple it follows from property (2.50) that the dual of property (2.45) does hold. That is,

$$X = (\mathsf{outl} \cdot X) \vartriangle (\mathsf{outr} \cdot X) \quad \text{if } X \text{ is simple,} \tag{2.55}$$

since using property (2.50) and the fact that simple X distribute over intersection,

$$\begin{aligned} X \;&=\; (\mathsf{outl}^\circ \cdot \mathsf{outl} \;\cap\; \mathsf{outr}^\circ \cdot \mathsf{outr}) \cdot X \;=\; \mathsf{outl}^\circ \cdot \mathsf{outl} \cdot X \;\cap\; \mathsf{outr}^\circ \cdot \mathsf{outr} \cdot X \\ &=\; (\mathsf{outl} \cdot X) \vartriangle (\mathsf{outr} \cdot X) \end{aligned}$$

For coproduct, the computation rules with equality (2.44) together with property (2.45) were used to prove a dual universal property for coproduct. Hence, if we restrict our attention to total and simple relations, i.e. functions, we can prove in the same way as for coproduct the dual universal property for product using properties (2.54) and (2.55).

For functions f, g and h, instantiating $X := f \vartriangle g$ and $R := \mathsf{outl} \cdot h$ and $S := \mathsf{outr} \cdot h$ in the universal property (2.52) yields

$$f \vartriangle g \subseteq (\mathsf{outl} \cdot h) \vartriangle (\mathsf{outr} \cdot h) \quad \equiv \quad \mathsf{outl} \cdot f \vartriangle g \subseteq \mathsf{outl} \cdot h \wedge \mathsf{outr} \cdot f \vartriangle g \subseteq \mathsf{outr} \cdot h$$

Now, using property (2.55), i.e. $h = (\mathsf{outl} \cdot h) \vartriangle (\mathsf{outr} \cdot h)$, and the computation rules (2.53) for $f \vartriangle g$, it follows that we have the universal property,

$$f \vartriangle g \subseteq h \quad \equiv \quad f \subseteq \mathsf{outl} \cdot h \wedge g \subseteq \mathsf{outr} \cdot h \tag{2.56}$$

Combining both universal properties (2.52) and (2.56), we have for functions f, g and h, the universal property

$$f \vartriangle g = h \quad \equiv \quad f = \mathsf{outl}_{A,B} \cdot h \wedge g = \mathsf{outr}_{A,B} \cdot h \tag{2.57}$$

Many properties for the relational coproduct follow from the fact that the relational coproduct is a real categorical coproduct. Since this is not the case for relational product, we have to prove some of its properties using requirements (2.49) and (2.50) and the definition of split (2.51). Since split is defined as an intersection, split inherits properties of intersection. For instance, we have the *distribution rules*

$$(R \vartriangle S) \cdot T \subseteq (R \cdot T) \vartriangle (S \cdot T) \;, \tag{2.58}$$

$$(R \vartriangle S) \cdot f = (R \cdot f) \vartriangle (S \cdot f) \quad \text{if f is simple.} \tag{2.59}$$

And the *modular law* and *modular identity*

$$R \vartriangle (S \cdot T) \subseteq ((R \cdot T^{\circ}) \vartriangle S) \cdot T \ , \tag{2.60}$$

$$R \vartriangle (S \cdot f) = ((R \cdot f^{\circ}) \vartriangle S) \cdot f \quad \text{if f is cosimple.} \tag{2.61}$$

Next we want to show that, although relational product is not a categorical product,

$$R \times S \ \triangleq \ (R \cdot \mathsf{outl}_{A,B}) \vartriangle (S \cdot \mathsf{outr}_{A,B}) \quad \text{for } R : C \leftarrow A \text{ and } S : D \leftarrow B$$

still defines a binary relator. The most difficult part is to prove that "$\times$" distributes over composition. It follows from the following property:

$$(R \vartriangle S)^{\circ} \cdot (T \vartriangle U) \ = \ (R^{\circ} \cdot T) \ \cap \ (S^{\circ} \cdot U) \tag{2.62}$$

We first prove this property for the case that one of the arguments is a function. For the moment assume that U is a function. We calculate:

$$
\begin{array}{ll}
 & (R \vartriangle S)^{\circ} \ \cdot \ T \vartriangle U \\
\subseteq & \qquad \{ \quad \text{definition split, distribution over intersection} \quad \} \\
 & R^{\circ} \ \cdot \ \mathsf{outl} \ \cdot \ T \vartriangle U \ \ \cap \ \ S^{\circ} \ \cdot \ \mathsf{outr} \ \cdot \ T \vartriangle U \\
\subseteq & \qquad \{ \quad \text{computation rules } \mathsf{outl}, \mathsf{outr} \quad \} \\
 & R^{\circ} \cdot T \ \cap \ S^{\circ} \cdot U \\
= & \qquad \{ \quad \text{computation } \mathsf{outl}, U \text{ total} \quad \} \\
 & R^{\circ} \ \cdot \ \mathsf{outl} \ \cdot \ T \vartriangle U \ \ \cap \ \ S^{\circ} \cdot U \\
\subseteq & \qquad \{ \quad \text{modular law} \quad \} \\
 & (R^{\circ} \cdot \mathsf{outl} \ \cap \ S^{\circ} \cdot U \cdot (T \vartriangle U)^{\circ}) \ \cdot \ T \vartriangle U \\
\subseteq & \qquad \{ \quad \text{definition split: } (T \vartriangle U)^{\circ} \subseteq U^{\circ} \cdot \mathsf{outr} \quad \} \\
 & (R^{\circ} \cdot \mathsf{outl} \ \cap \ S^{\circ} \cdot U \cdot U^{\circ} \cdot \mathsf{outr}) \ \cdot \ T \vartriangle U \\
\subseteq & \qquad \{ \quad U \text{ simple, definition split} \quad \} \\
 & (R \vartriangle S)^{\circ} \ \cdot \ T \vartriangle U
\end{array}
$$

So, property (2.62) holds if U is a function. Similarly, this property also holds if T is a function by duality between left and right, and if either R or S is a function by taking converses. Using property (2.62) if at least one of the arguments is a function, we can prove property (2.62) for arbitrary relations:

$$
\begin{array}{ll}
 & (R \vartriangle S)^{\circ} \ \cdot \ T \vartriangle U \\
= & \qquad \{ \quad \text{definition split} \quad \} \\
 & (R^{\circ} \cdot \mathsf{outl} \cap S^{\circ} \cdot \mathsf{outr}) \ \cdot \ T \vartriangle U
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \text{outl function, property above} \quad \} \\
& (\mathsf{R} \vartriangle \mathsf{id})^\circ \cdot \mathsf{outl} \vartriangle (\mathsf{S}^\circ \cdot \mathsf{outr}) \cdot \mathsf{T} \vartriangle \mathsf{U} \\
= & \{ \quad \text{definition split: } \mathsf{outl} \vartriangle (\mathsf{S}^\circ \cdot \mathsf{outr}) = (\mathsf{outl} \vartriangle (\mathsf{S} \cdot \mathsf{outr}))^\circ \quad \} \\
& (\mathsf{R} \vartriangle \mathsf{id})^\circ \cdot (\mathsf{outl} \vartriangle (\mathsf{S} \cdot \mathsf{outr}))^\circ \cdot \mathsf{T} \vartriangle \mathsf{U} \\
= & \{ \quad \text{outl function, property above} \quad \} \\
& (\mathsf{R} \vartriangle \mathsf{id})^\circ \cdot (\mathsf{outl}^\circ \cdot \mathsf{T} \cap \mathsf{outr}^\circ \cdot \mathsf{S}^\circ \cdot \mathsf{U}) \\
= & \{ \quad \mathsf{outl}^\circ \cdot \mathsf{T} \cap \mathsf{outr}^\circ \cdot \mathsf{S}^\circ \cdot \mathsf{U} = \mathsf{T} \vartriangle (\mathsf{S}^\circ \cdot \mathsf{U}) \\
& \quad\quad \text{id function, property above} \quad \} \\
& \mathsf{R}^\circ \cdot \mathsf{T} \cap \mathsf{S}^\circ \cdot \mathsf{U}
\end{array}
$$

Having the notion of cosplit, denoted by "$\blacktriangle$" as the converse-conjugate of split, i.e.

$$\mathsf{R} \blacktriangle \mathsf{S} \triangleq (\mathsf{R}^\circ \vartriangle \mathsf{S}^\circ)^\circ = \mathsf{R} \cdot \mathsf{outl} \cap \mathsf{S} \cdot \mathsf{outr} \quad ,$$

property (2.62) can be rewritten as

$$\mathsf{R} \blacktriangle \mathsf{S} \cdot \mathsf{T} \vartriangle \mathsf{U} = \mathsf{R} \cdot \mathsf{T} \cap \mathsf{S} \cdot \mathsf{U} \quad . \tag{2.63}$$

We call property (2.63), *cosplit-split elimination*.

As a matter of fact, we could have defined relational product equivalently using equation (2.62) together with equation (2.50). It is then possible to derive property (2.49) from it [1]. Although this is easier than to derive property (2.63) from our definition of relational product, we have chosen not to do so. The reason for this is that we wanted to show that a unitary tabular allegory has relational products.

From property (2.62) the so-called *product-split fusion* rule follows:

$$\mathsf{R} \times \mathsf{S} \cdot (\mathsf{T} \vartriangle \mathsf{U}) = (\mathsf{R} \cdot \mathsf{T}) \vartriangle (\mathsf{S} \cdot \mathsf{U})$$

since $\mathsf{R} \times \mathsf{S} = ((\mathsf{R}^\circ \cdot \mathsf{outl}) \vartriangle (\mathsf{S}^\circ \cdot \mathsf{outr}))^\circ$ and $\mathsf{outl}^\circ \cdot \mathsf{R} \cdot \mathsf{T} \cap \mathsf{outr}^\circ \cdot \mathsf{S} \cdot \mathsf{U} = (\mathsf{R} \cdot \mathsf{T}) \vartriangle (\mathsf{S} \cdot \mathsf{U})$. Using the product-split fusion rule it is easy to prove that "$\times$" distributes over composition. The requirement that "$\times$" preserves identities is precisely property (2.50), hence, "$\times$" is a functor. Furthermore, it is easy to verify that "$\times$" is monotonic and commutes with converse, hence, "$\times$" is a relator.

From the fact that "$\times$" is a relator, it follows that "$\times$" respects functions. Furthermore, $\mathsf{f} \vartriangle \mathsf{g} = \mathsf{f} \times \mathsf{g} \cdot \mathsf{id} \vartriangle \mathsf{id}$ is a function if $\mathsf{id} \vartriangle \mathsf{id}$ is a function too. This is easy to prove: simplicity follows from requirement (2.50):

$$\mathsf{id} \vartriangle \mathsf{id} \cdot (\mathsf{id} \vartriangle \mathsf{id})^\circ = (\mathsf{outl}^\circ \cap \mathsf{outr}^\circ) \cdot (\mathsf{outl} \cap \mathsf{outr}) \subseteq \mathsf{outl}^\circ \cdot \mathsf{outl} \cap \mathsf{outr}^\circ \cdot \mathsf{outr} = \mathsf{id}$$

Totality follows from (2.27) and requirement (2.49):

$$(\mathsf{id} \vartriangle \mathsf{id}){>} = (\mathsf{outl}^\circ \cap \mathsf{outr}^\circ){>} = \mathsf{id} \cap \mathsf{outl} \cdot \mathsf{outr}^\circ = \mathsf{id}$$

So, although the relational product is not a categorical product, $(\mathsf{A} \times \mathsf{B}\,, \mathsf{outl}_{\mathsf{A},\mathsf{B}}\,, \mathsf{outr}_{\mathsf{A},\mathsf{B}})$ is a real categorical product for functions, i.e. for the sub-category *Map*.

We have proven that

$$R \triangledown S = R/\mathsf{inl} \cap S/\mathsf{inr}$$

and since for function f we have $f^\circ \cdot R = f\backslash R$, it follows that

$$R \vartriangle S = \mathsf{outl}^\circ \cdot R \cap \mathsf{outr}^\circ \cdot S = \mathsf{outl}\backslash R \cap \mathsf{outr}\backslash S \ . \tag{2.64}$$

Hence, since $(R/S)^\circ = R^\circ\backslash S^\circ$, it follows that "$\triangledown$" and "$\vartriangle$" are in some sense converse duals of each other. This fact we exploit in chapter 4 on membership .

2.5.3 Natural transformations

Since relators are by definition functors it follows that the standard definition of a natural transformation between relators makes sense. That is to say, a collection of relations α indexed by objects (equivalently, a mapping α of objects to relations) is a natural transformation of type $F \leftarrow G$, for relators F and G iff

$$FR \cdot \alpha_B = \alpha_A \cdot GR \quad \text{for each } R : A \leftarrow B.$$

Note that we have the normal typing rules for natural transformations:

$$H\alpha \ : \ HF \leftarrow HG \ \Leftarrow \ \alpha : F \leftarrow G \tag{2.65}$$

and

$$\alpha_H \ : \ FH \leftarrow GH \ \Leftarrow \ \alpha : F \leftarrow G \tag{2.66}$$

However, many collections of relations we will encounter are not natural with equality but with an inclusion. That is why we define two other types of natural transformation denoted by $F \hookleftarrow G$ and $F \hookrightarrow G$, respectively. We have:

$$\alpha : F \hookleftarrow G \ \triangleq \ FR \cdot \alpha_B \supseteq \alpha_A \cdot GR \quad \text{for each } R : A \leftarrow B.$$

and

$$\alpha : F \hookrightarrow G \ \triangleq \ FR \cdot \alpha_B \subseteq \alpha_A \cdot GR \quad \text{for each } R : A \leftarrow B.$$

Note that $\alpha : F \hookleftarrow G \equiv \alpha^\circ : F \hookrightarrow G$ and $\alpha : F \leftarrow G \ \equiv \ \alpha : F \hookleftarrow G \wedge \alpha : F \hookrightarrow G$.

Since natural transformations of type $F \hookleftarrow G$ are more common than ones of type $F \leftarrow G$, we say that α is a *natural transformation* if $\alpha : F \hookleftarrow G$ and we say that α is a *proper* natural transformation if $\alpha : F \leftarrow G$. If $\alpha : F \hookrightarrow G$ then α is a *conatural transformation*.

Typing rules For natural transformations we still have the same typing rules as for proper natural transformations. That is,

$$H\alpha \ : \ HF \hookleftarrow HG \ \Leftarrow \ \alpha : F \hookleftarrow G \tag{2.67}$$

and

$$\alpha_H \ : \ FH \hookleftarrow GH \ \Leftarrow \ \alpha : F \hookleftarrow G \tag{2.68}$$

Property (2.68) is trivial to verify, property (2.67) follows from monotonicity of relator H :

$$HFR \ \cdot \ H\alpha_B \ = \ H(FR \cdot \alpha_B) \ \supseteq \ H(\alpha_A \cdot GR) \ = \ H\alpha_A \ \cdot \ HGR$$

Interpretation of natural transformations The interpretation of a natural transformation $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$, i.e,

$$\mathsf{F}R \cdot \alpha_B \supseteq \alpha_A \cdot \mathsf{G}R \quad \text{for each } R : A \leftarrow B, \tag{2.69}$$

is that α_B takes a G-structure containing elements of type B and transforms it into an F-structure without changing the value of the elements. That is to say, α_B can throw away or duplicate elements of the G-structure but it can not invent new values. In chapter 4 we give a formal justification for this interpretation.

Consider the case that α only duplicates elements. Then for equation (2.69), if R is nondeterministic, the lhs first applies R to an element and duplicates the nondeterministically chosen element. In contrast, the rhs first duplicates the element and then applies R to both copies. Hence, the rhs is more nondeterministic than the lhs. An example of a natural transformation which duplicates elements is $\mathsf{fork}_A \triangleq \mathsf{id}_A \vartriangle \mathsf{id}_A \; : \; A \times A \leftarrow A$, i.e. we have

$$R \times R \cdot \mathsf{fork} \supseteq \mathsf{fork} \cdot R$$

Now, take $R = (+1) \cup (-1)$, i.e. R nondeterministically increments or decrements a natural number. Then if we apply $\mathsf{fork} \cdot R$ to 0, we get either the pair $(+1\,,+1)$ or the pair $(-1\,,-1)$. On the other hand, if we apply $R \times R \cdot \mathsf{fork}$ to 0, we get one of the four possibilities $(+1\,,+1)$, $(-1\,,+1)$, $(+1\,,-1)$ or $(-1\,,-1)$. Indeed, if we take the restriction to deterministic relations, i.e. simple relations, the inclusion becomes an equality:

$$f \times f \cdot \mathsf{fork} = \mathsf{fork} \cdot f \quad \text{for simple } f.$$

Now, consider the case that α only throws elements away. Then for equation (2.69), if R is not total, the lhs is not defined on G-structures which contain elements on which R is not defined. On the other hand, the rhs is defined on G-structures containing elements which are either thrown away by α, or are kept by α and are in the domain of R. Hence, the rhs is more defined than the lhs. An example of a natural transformation which throws elements away is projection, i.e. we have

$$R \cdot \mathsf{outl} \supseteq \mathsf{outl} \cdot R \times R$$

Now, take $R = (0 <)$ i.e. R is the partial identity on natural numbers which is only defined on the positive numbers. Then $\mathsf{outl} \cdot R \times R$ is not defined on the pair $(+1\,,-1)$, whereas $R \cdot \mathsf{outl}$ yields $(+1)$ when applied to $(+1\,,-1)$. Indeed, if we take the restriction to relations which are defined everywhere, i.e. total relations, we get equality:

$$R \cdot \mathsf{outl} = \mathsf{outl} \cdot R \times R \quad \text{for total } R.$$

An arbitrary natural transformation can throw away some elements and at the same time duplicate some other elements. The combination of both examples suggests that for simple and total relations, i.e. functions, we have equality for arbitrary natural transformations. That is to say, we have the following lemma:

Lemma 2.70 If $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$ then,

$$\mathsf{F}f \cdot \alpha_B = \alpha_A \cdot \mathsf{G}f \quad \text{for each function } f : A \leftarrow B.$$

Proof The $\supseteq$ part follows from the definition of $\mathsf{F} \hookleftarrow \mathsf{G}$. For the remaining inclusion we calculate:

$$
\begin{array}{ll}
 & \mathsf{F}f \cdot \alpha \subseteq \alpha \cdot \mathsf{G}f \\
\equiv & \{ \quad \text{relators respect functions, shunting} \quad \} \\
 & \alpha \cdot \mathsf{G}f^{\circ} \subseteq \mathsf{F}f^{\circ} \cdot \alpha \\
\equiv & \{ \quad \text{converse} \quad \} \\
 & \mathsf{G}f \cdot \alpha^{\circ} \subseteq \alpha^{\circ} \cdot \mathsf{F}f \\
\equiv & \{ \quad \alpha : \mathsf{F} \hookleftarrow \mathsf{G} \equiv \alpha^{\circ} : \mathsf{G} \hookrightarrow \mathsf{F} \quad \} \\
 & \mathit{true}
\end{array}
$$

□

If the allegory is tabular, the other way around is also true:

Lemma 2.71

$$\alpha : \mathsf{F} \hookleftarrow \mathsf{G} \;\Leftarrow\; (\mathsf{F}f \cdot \alpha_B = \alpha_A \cdot \mathsf{G}f \quad \text{for each function } f : A \leftarrow B) \;.$$

Proof Assume that $\mathsf{F}f \cdot \alpha = \alpha \cdot \mathsf{G}f$ for each function f. Let (f, g) be a tabulation of R, then

$$
\begin{array}{ll}
 & \mathsf{F}R \cdot \alpha \supseteq \alpha \cdot \mathsf{G}R \\
\equiv & \{ \quad (f, g) \text{ tabulates } R, \mathsf{F} \text{ and } \mathsf{G} \text{ relators} \quad \} \\
 & \mathsf{F}f \cdot \mathsf{F}g^{\circ} \cdot \alpha \supseteq \alpha \cdot \mathsf{G}f \cdot \mathsf{G}g^{\circ} \\
\equiv & \{ \quad \text{assumption: } \alpha \cdot \mathsf{G}f = \mathsf{F}f \cdot \alpha \quad \} \\
 & \mathsf{F}f \cdot \mathsf{F}g^{\circ} \cdot \alpha \supseteq \mathsf{F}f \cdot \alpha \cdot \mathsf{G}g^{\circ} \\
\Leftarrow & \{ \quad \text{monotonicity} \quad \} \\
 & \mathsf{F}g^{\circ} \cdot \alpha \supseteq \alpha \cdot \mathsf{G}g^{\circ} \\
\equiv & \{ \quad \text{relators respect functions, shunting} \quad \} \\
 & \alpha \cdot \mathsf{G}g \supseteq \mathsf{F}g \cdot \alpha \\
\Leftarrow & \{ \quad \text{assumption} \quad \} \\
 & \mathit{true}
\end{array}
$$

□

Recall that relators respect functions, i.e. relators are functors on the sub-category *Map*. From the combination of both lemmas, it follows that for a *functional* collection of arrows α, we have,

$$\alpha : \mathsf{F} \hookleftarrow \mathsf{G} \text{ in } \mathcal{A} \;\equiv\; \alpha : \mathsf{F} \leftarrow \mathsf{G} \text{ in } \mathit{Map}(\mathcal{A})$$

for tabular allegory $\mathcal{A}$. In this way, the notion of $\mathsf{F} \hookleftarrow \mathsf{G}$ for relations is the natural[3] generalization of the notion of a natural transformation for functions.

[3]Pun intended

2.5.4 Power relator

From the definition of existential image, i.e. $\mathsf{E}R = \Lambda(R \cdot \in)$, it follows that $\mathsf{E}R$ is a function for every R. Hence, E is a functor on functions, i.e. E is a functor on the sub-category *Map*. However, E is not monotonic, nor does it commute with converse. So, E is not a relator. However, in [7] it is shown that for a tabular allegory, the restriction of E to functions does have a relational extension. This relational extension, called the *power relator*, we denote by P. In [7], Bird and De Moor give a closed formula for P, i.e.

$$\mathsf{P}R = \in\backslash(R \cdot \in) \cap (\in^\circ \cdot R)/\in^\circ \tag{2.72}$$

but in order to show that this defines a relator, tabularity is assumed. Since P is a relational extension it follows that for all tabulations (f, g) of relation R,

$$\mathsf{P}R = \mathsf{E}f \cdot (\mathsf{E}g)^\circ \tag{2.73}$$

since

$$\mathsf{P}R = \mathsf{P}(f \cdot g^\circ) = \mathsf{P}f \cdot (\mathsf{P}g)^\circ = \mathsf{E}f \cdot (\mathsf{E}g)^\circ$$

In this thesis, we use only equation (2.73). So, although we do not assume tabularity in general, for results which mention the power relator, tabularity is assumed.

From the closed formula (2.72) the following interpretation for $\mathsf{P}R$ follows. For sets x and y,

$$x\,\mathsf{P}R\,y \;\equiv\; \forall(a \in x :: \exists(b \in y :: a\ R\ b)) \wedge \forall(b \in y :: \exists(a \in x :: a\ R\ b))$$

2.5.5 Recursive datatypes

In this subsection we define the relational extension of a tree type functor. Since tree type functors are defined using catamorphisms we first define the relational extension of a catamorphism.

Relational catamorphisms

Recall that an allegory is a category and a relator is a functor on relations. We define the relational catamorphism as the normal categorical notion of a catamorphism in the category of relations. So, suppose that relation $\mathsf{in} : A \leftarrow \mathsf{F}A$ is an initial algebra. That is to say, for a relation $R : B \leftarrow \mathsf{F}B$, i.e. R is an F-algebra, there exists a unique $\mathsf{F}-$homomorphism to R from in. We denote this unique homomorphism by $(\!|R|\!)$. In other words, $(\!|R|\!)$ is characterized by the universal property, called the *unique extension property*, that for each relation $X : B \leftarrow A$,

$$X = (\!|R|\!) \;\equiv\; X \cdot \mathsf{in} = R \cdot \mathsf{F}X \tag{2.74}$$

If the allegory is a power allegory it can be shown that a relational initial F-algebra coincide with the initial F-algebra for the sub-category of total functions.

Since relational catamorphism is the standard notion of catamorphism defined on relations, it follows that an initial algebra is an isomorphism. Recall that a relation is an isomorphism iff it is a bijection. That is to say, for in we have, $\mathsf{in}^\circ \cdot \mathsf{in} = \mathsf{Fid}$ and $\mathsf{in} \cdot \mathsf{in}^\circ = \mathsf{id}$. Using this, the unique extension property (2.74) can be rewritten as

$$X = (\!|R|\!) \quad \equiv \quad X = R \cdot \mathsf{F}X \cdot \mathsf{in}^\circ$$

In other words, $(\!|R|\!)$ is the unique, so also the least and greatest, fixed-point of the mapping $\lambda X.(R \cdot \mathsf{F}X \cdot \mathsf{in}^\circ)$. Since this mapping is monotonic, it follows from Knaster-Tarski that we have,

$$X \subseteq (\!|R|\!) \quad \Leftarrow \quad X \subseteq R \cdot \mathsf{F}X \cdot \mathsf{in}^\circ \tag{2.75}$$

and

$$X \supseteq (\!|R|\!) \quad \Leftarrow \quad X \supseteq R \cdot \mathsf{F}X \cdot \mathsf{in}^\circ \ . \tag{2.76}$$

For relational catamorphisms we have the normal fusion rule. Furthermore, the equations (2.75) and (2.76) imply two other fusion rules with inclusion. That is to say, we have,

$$R \cdot (\!|T|\!) = (\!|S|\!) \ \Leftarrow \ R \cdot T = S \cdot \mathsf{F}R \ , \tag{2.77}$$

$$R \cdot (\!|T|\!) \subseteq (\!|S|\!) \ \Leftarrow \ R \cdot T \subseteq S \cdot \mathsf{F}R \ , \tag{2.78}$$

$$R \cdot (\!|T|\!) \supseteq (\!|S|\!) \ \Leftarrow \ R \cdot T \supseteq S \cdot \mathsf{F}R \ . \tag{2.79}$$

Application of the rules (2.77) – (2.79) will be indicated by the hint "fusion". Note that from (2.78) or (2.79) it follows by taking $R := \mathsf{id}$ that the catamorphism operator $(\!|\_|\!)$ is monotonic.

So, relational catamorphisms are defined using the standard categorical construction in the category of relations. However, the notion of a relational catamorphism is an extension of the notion of a catamorphism on functions, since the mapping $(\!|\_|\!)$ respects functions. Simplicity of $(\!|f|\!)$ for simple f follows from,

$$
\begin{array}{ll}
 & (\!|f|\!) \cdot (\!|f|\!)^\circ \subseteq \mathsf{id} \\
\equiv & \qquad \{ \quad \text{factors} \quad \} \\
 & (\!|f|\!) \subseteq \mathsf{id}/(\!|f|\!)^\circ \\
\Leftarrow & \qquad \{ \quad \text{property (2.76)} \quad \} \\
 & f \cdot \mathsf{F}(\mathsf{id}/(\!|f|\!)^\circ) \cdot \mathsf{in}^\circ \subseteq \mathsf{id}/(\!|f|\!)^\circ \\
\equiv & \qquad \{ \quad \text{factors} \quad \} \\
 & f \cdot \mathsf{F}(\mathsf{id}/(\!|f|\!)^\circ) \cdot \mathsf{in}^\circ \cdot (\!|f|\!)^\circ \subseteq \mathsf{id} \\
\equiv & \qquad \{ \quad \text{computation, } \mathsf{F} \text{ functor} \quad \} \\
 & f \cdot \mathsf{F}(\mathsf{id}/(\!|f|\!)^\circ \cdot (\!|f|\!)^\circ) \cdot f^\circ \subseteq \mathsf{id} \\
\equiv & \qquad \{ \quad \text{factors, } f \text{ simple} \quad \} \\
 & \mathit{true}
\end{array}
$$

That the operator $(\!|\_|\!)$ respects total relations, i.e. $(\!|R|\!)^\circ \cdot (\!|R|\!) \supseteq \mathsf{id}_A$ for total R, is not difficult to verify using fusion law (2.79) since $\mathsf{id}_A = (\!|\mathsf{in}|\!)$.

Tree type relators

Since relational catamorphism is the standard notion of a catamorphism on relations, the relational extension of a type functor suggests itself. We take the same definition as for functions. That is, let $\otimes$ be a binary relator and let, for each A, $\mathsf{in}_A \;:\; \mathsf{T}A \leftarrow A \otimes \mathsf{T}A$ be an initial algebra of $(A\otimes)$. Then for $R : A \leftarrow B$,

$$\mathsf{T}R = (\!|A\otimes;\ \mathsf{in}_B \cdot R\otimes \mathsf{id}_{\mathsf{T}B}|\!)$$

defines a relator, the *tree type relator induced by* $\otimes$. From subsection 2.2.5 it follows that T is a functor. That T commutes with converse follows from,

$$\begin{array}{ll}
 & (\mathsf{T}R)^\circ = \mathsf{T}(R^\circ) \\
\equiv & \quad \{ \quad \text{definition } \mathsf{T}\text{, unique extension} \quad \} \\
 & (\mathsf{T}R)^\circ \cdot \mathsf{in} = \mathsf{in} \cdot R^\circ \otimes \mathsf{id} \cdot \mathsf{id} \otimes (\mathsf{T}R)^\circ \\
\equiv & \quad \{ \quad \text{converse, } \otimes \text{ relator} \quad \} \\
 & \mathsf{in}^\circ \cdot \mathsf{T}R = \mathsf{id} \otimes \mathsf{T}R \cdot R \otimes \mathsf{id} \cdot \mathsf{in}^\circ \\
\equiv & \quad \{ \quad \mathsf{in} \text{ isomorphism, } \otimes \text{ functor} \quad \} \\
 & \mathsf{T}R \cdot \mathsf{in} = \mathsf{in} \cdot R \otimes \mathsf{id} \cdot \mathsf{id} \otimes \mathsf{T}R \\
\equiv & \quad \{ \quad \text{definition } \mathsf{T} \quad \} \\
 & \textit{true}
\end{array}$$

Finally, monotonicity of T follows from monotonicity of catamorphism. Hence, T is a relator.

Cocatamorphism

We define the notion of a *cocatamorphism* as the converse-conjugate of a catamorphism. That is, for coalgebra $R \;:\; \mathsf{F}A \leftarrow A$, we define

$$[\![\mathsf{F};\ R]\!] \;\triangleq\; (\!|\mathsf{F};\ R^\circ|\!)^\circ \ .$$

We call the mapping $[\![\_]\!]$ a cocatamorphism instead of an anamorphism as done in [1]. An anamorphism [36] is by definition the dual of catamorphism. The reason why we prefer the term cocatamorphism instead of anamorphism is because $[\![\_]\!]$ is *not* the relational extension of the anamorphism operator on functions. For instance, $[\![\_]\!]$ does not respect functions in general.

Chapter 3

Calculational Preliminaries

In this chapter we introduce some additional, non-standard concepts which we need in the remainder of this thesis. We start with the introduction of the notion of a slok property. A slok property is a kind of healthiness condition on natural transformations. A naturality requirement does not characterize a natural transformation uniquely whereas, under some conditions, a slok property uniquely characterizes a natural transformation. This fact we exploit in Chapter 5. Here we show that slok properties are preserved by composing natural transformations, and by composing functors and natural transformations. This is done by defining a slok property as a property of an arrow of the slok category and investigating the typing rules of this category.

In the second half of this chapter we introduce the so-called τ-Δ calculus. This calculus provides us with a variable-free way of constructing and calculating with relators which take a vector of arguments and/or give as result a vector of arguments.

3.1 Category "Slok"

Natural transformations and isomorphisms in the literature are often introduced by just giving their type. In general, however, natural transformations are not uniquely defined by their type. So a phraseology like "the obvious natural transformation" or the "canonical natural transformation" is used, hereby avoiding the need to give the (well-known) closed formula. An example is the "obvious natural isomorphism"

$$\mathsf{swap} : \mathsf{B} \times \mathsf{A} \leftarrow \mathsf{A} \times \mathsf{B}$$

The intended natural transformation is: $\mathsf{swap} = \mathsf{outr} \vartriangle \mathsf{outl}$. Furthermore, one assumes that the "obvious natural isomorphism" has some well-known properties, for instance swap has the property that

$$\mathsf{g} \vartriangle \mathsf{f} \quad = \quad \mathsf{swap}_{\mathsf{A},\mathsf{B}} \cdot \mathsf{f} \vartriangle \mathsf{g} \quad \text{for all } \mathsf{f} : \mathsf{A} \leftarrow \mathsf{C} \text{ and } \mathsf{g} : \mathsf{B} \leftarrow \mathsf{C}. \tag{3.1}$$

and indeed this property can be proven for the formula given above. We call a property like (3.1), a *slok* property, "slok" being the dutch word for "to gobble". We consider a slok property as a healthiness property of a natural transformation. That is to say, a specific arrow should not only be a natural transformation (or even a natural isomorphism) but it should also satisfy a certain slok property. In some categories there are many natural isomorphisms of type $B \times A \leftarrow A \times B$ but there exists only one natural transformation which has slok property (3.1).

In this section, we define the category Slok. An arrow in this category is a collection of arrows in the base category which has a slok property. We give typing rules for this category: we give the slok properties of several combinations of arrows from category Slok. Under certain conditions, a slok property has at most one solution. Furthermore, we give a condition under which such a unique arrow in category Slok is a natural transformation i.e. an arrow in category Fun.

So, under certain conditions, if one has to construct a natural transformation which should satisfy a corresponding slok property then the slok property predicts the only possible candidate. Having verified that this candidate indeed satisfies the slok property it then follows that the collection of arrows is also a natural transformation.

Definition of Slok A mapping to $\mathcal{C}$-arrows from the collection of $\mathcal{B}$-arrows is an object of category Slok if it behaves nicely with respect to typing: mapping F is an object if the target of Ff, for all f, only depends on the target of f, i.e.

$$(Ff)\lhd = (Fg)\lhd \quad \Leftarrow \quad f\lhd = g\lhd \ \text{ for all } f \text{ and } g. \tag{3.2}$$

Recall that "$\_\lhd$" denotes the target operator. That is, $f\lhd = A$ if $f : A \leftarrow B$. Note that we do not assume that the collection of $\mathcal{B}$-arrows form a category.

From (3.2) it follows that mapping F induces a mapping on objects, say $F^{\lhd}$, such that $(Ff)\lhd = F^{\lhd}f\lhd$. Specifically, $F^{\lhd}A = (Fid_A)\lhd$, or equivalently, $F^{\lhd} = \lhd\circ F\circ id$.

Examples of objects of Slok are functors; for functor F, the mapping $F^{\lhd}$ is the object mapping of F. Another example is the composition of a functor and a natural transformation, e.g. if α is a natural transformation $F \leftarrow G$, then the mapping $Hf \triangleq Ff \cdot \alpha \; (= \alpha \cdot Gf)$ satisfies condition (3.2): $H^{\lhd}$ is again the object mapping of F. Note that the objects of category Slok are closed under function composition; $(FG)^{\lhd}$ is $F^{\lhd}G^{\lhd}$.

Recalling (3.1), the mapping "$\vartriangle$" is an object of Slok since it is the composition of the product functor and the natural transformation $id \vartriangle id$. We now define the arrows of Slok in such a way that swap is such an arrow.

The arrows between the objects F and G, both mappings to $\mathcal{C}$-arrows from $\mathcal{B}$-arrows, are collections of $\mathcal{C}$-arrows indexed by the objects of $\mathcal{B}$ with a so-called *slok property* i.e. for collection α, $\alpha : F \leftarrow G$ iff

$$Ff = \alpha_A \cdot Gf \quad \text{for each } f : A \leftarrow B. \tag{3.3}$$

Note the resemblance with the definition of a natural transformation: with a natural transformation α appears on both sides; with a slok property α is "gobbled up" when being composed with Gf. This explains the term "slok": "slok" being the dutch word for "to gobble".

Notice that

$$\alpha_A \ : \ (Ff)\lhd \leftarrow (Gf)\lhd \quad \text{for each } f : A \leftarrow B,$$

or equivalently, since F and G satisfy condition (3.2),

$$\alpha_A \;:\; F^{\triangleleft}A \leftarrow G^{\triangleleft}A \quad \text{for all objects } A.$$

The composition of collections of arrows $\alpha : F \leftarrow G$ and $\beta : G \leftarrow H$ is defined in the usual way: $(\alpha \cdot \beta)_A = \alpha_A \cdot \beta_A$. Then, $\alpha \cdot \beta : F \leftarrow H$ since $\alpha_A \cdot \beta_A \cdot Hf = \alpha_A \cdot Gf = Ff$ for each $f : A \leftarrow B$. That the above definition indeed defines a category is easily verified: associativity of composition is immediate from the definition and the identity arrow id_F on object F is $(\mathsf{id}_F)_A \triangleq \mathsf{id}_{F^{\triangleleft}A}$.

Before we state some typing rules, we define two subclasses of objects of the category Slok.

Definition 3.4 Mapping F is *fuseable* into mapping G iff

$$Ff \cdot Gg = G(f \cdot g) \quad \text{for all } f \text{ and } g.$$

□

Definition 3.5 Mapping F is *recoverable from* mapping G *with* the collection of arrows β iff $Ff = G(f \cdot \beta_B)$ for each $f : A \leftarrow B$. We say that F is *recoverable from* G if there exists a collection of arrows with which F can be recovered.
□

Trivially, a functor is fuseable into itself, and if mapping F is fuseable into mapping H and mapping G is fuseable into I then FG is fuseable into HI since

$$FGf \cdot HIg = H(Gf \cdot Ig) = HI(f \cdot g) \ .$$

Similarly, a functor is recoverable from itself with id and if mapping F is recoverable from mapping G with β then HF is recoverable from HG with β since $HFf = HG(f \cdot \beta)$.

A property we use very often is that if functor F is fuseable into mapping G then F is recoverable from G with β iff $F\mathsf{id} = G\beta$ since

$$Ff = Ff \cdot F\mathsf{id} = Ff \cdot G\beta = G(f \cdot \beta)$$

As an example, product is fusable into split and product is recoverable from split with the pair $(\mathsf{outl}\,,\mathsf{outr})$.

3.1.1 Typing rules

Next we state the typing rules for category Slok. We start with an unconditional one; for all H,

$$\alpha_{H^{\triangleleft}} \;:\; FH \leftarrow GH \quad \Leftarrow \quad \alpha : F \leftarrow G \tag{3.6}$$

where $(\alpha_{H^{\triangleleft}})_A = \alpha_{H^{\triangleleft}A}$, since $\alpha_{H^{\triangleleft}A} \cdot FHf = GHf$ for each $f : A \leftarrow B$. Furthermore, if mapping H' is fuseable into mapping H, then

$$H'\alpha \;:\; HF \leftarrow HG \quad \Leftarrow \quad \alpha : F \leftarrow G \tag{3.7}$$

where $(\mathsf{H}'\alpha)_\mathsf{A} = \mathsf{H}'(\alpha_\mathsf{A})$, since $\mathsf{H}'\alpha_\mathsf{A} \cdot \mathsf{HGf} = \mathsf{H}(\alpha_\mathsf{A} \cdot \mathsf{Gf}) = \mathsf{HFf}$ for each $\mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}$. Now, since a functor is fuseable into itself, it follows that for every functor H,

$$\mathsf{H}\alpha : \mathsf{HF} \leftarrow \mathsf{HG} \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \ . \tag{3.8}$$

Combining the rules (3.6) and (3.7) we get, if mapping G' is fuseable into mapping G, then

$$\alpha_{\mathsf{H}^{\triangleleft}} \cdot \mathsf{G}'\beta : \mathsf{FH} \leftarrow \mathsf{GI} \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \wedge \beta : \mathsf{H} \leftarrow \mathsf{I} \tag{3.9}$$

since $\alpha_{\mathsf{H}^{\triangleleft}} : \mathsf{FH} \leftarrow \mathsf{GH}$ and $\mathsf{G}'\beta : \mathsf{GH} \leftarrow \mathsf{GI}$. Similarly, if mapping F' is fuseable into mapping F, then

$$\mathsf{F}'\beta \cdot \alpha_{\mathsf{I}^{\triangleleft}} : \mathsf{FH} \leftarrow \mathsf{GI} \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \wedge \beta : \mathsf{H} \leftarrow \mathsf{I} \tag{3.10}$$

since $\mathsf{F}'\beta : \mathsf{FH} \leftarrow \mathsf{FI}$ and $\alpha_{\mathsf{I}^{\triangleleft}} : \mathsf{FI} \leftarrow \mathsf{GI}$.

The reader is encouraged to compare the typing rules for category Slok with the typing rules known for category Fun, i.e. the category with functors as objects and natural transformations as arrows between them. For instance, vertical composition of natural transformations, i.e. composition in category Fun, corresponds to composition in the category Slok, horizontal composition of natural transformations corresponds to the rule (3.9) or (3.10).

3.1.2 Uniqueness and naturality

Having the notion of recoverability, we can express a rule which states the uniqueness of arrows in category Slok:

Lemma 3.11 If $\alpha : \mathsf{F} \leftarrow \mathsf{G}$ and there exists a *functor* that is recoverable from G with β then $\alpha = \mathsf{F}\beta$. Hence, α is the unique arrow of type $\mathsf{F} \leftarrow \mathsf{G}$.

Proof Assume it is functor G' that is recoverable from G with β, then

$$\begin{array}{ll} & \alpha_\mathsf{A} \\ = & \{ \quad \mathsf{G}'\mathsf{id}_\mathsf{A} = \mathsf{G}\beta_\mathsf{A}, \text{functors preserve identity arrows.} \ \} \\ & \alpha_\mathsf{A} \cdot \mathsf{G}\beta_\mathsf{A} \\ = & \{ \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \ \} \\ & \mathsf{F}\beta_\mathsf{A} \end{array}$$

□

If we also assume that functor F' is fuseable into mapping F, we get our main lemma:

Lemma 3.12 If $\alpha : \mathsf{F} \xleftarrow{\mathsf{Slok}} \mathsf{G}$ and functor F' is fuseable into F and functor G' is recoverable from G then α is a natural transformation of type $\mathsf{F}' \xleftarrow{\mathsf{Fun}} \mathsf{G}'$.

Proof Assume functor G' is recoverable from G with β. From lemma (3.11) it then follows that $\alpha_\mathsf{A} = \mathsf{F}\beta_\mathsf{A}$. For naturality of α, we calculate, for each $\mathsf{f} : \mathsf{A} \leftarrow \mathsf{B}$,

$$
\begin{array}{ll}
& \alpha_A \cdot G'f \\
= & \{ \quad G'f = G(f \cdot \beta_B) \quad \} \\
& \alpha_A \cdot G(f \cdot \beta_B) \\
= & \{ \quad \alpha : F \leftarrow G \quad \} \\
& F(f \cdot \beta_B) \\
= & \{ \quad F' \text{ is fuseable into } F \quad \} \\
& F'f \cdot F\beta_B \\
= & \{ \quad \alpha_B = F\beta_B \quad \} \\
& F'f \cdot \alpha_B
\end{array}
$$

□

The above lemma shows that, under certain conditions, an arrow in the category Slok is an arrow in the category Fun, i.e. a natural transformation. For instance, the only possible candidate which has slok property

$$g \vartriangle f \quad = \quad \alpha_{A,B} \cdot f \vartriangle g \quad \text{for all } f : A \leftarrow C \text{ and } g : B \leftarrow C. \tag{3.13}$$

is given by lemma (3.11): $\alpha_{A,B} = \mathsf{outr}_{A,B} \vartriangle \mathsf{outl}_{A,B}$, i.e. swap, since product is recoverable from “$\vartriangle$” with $\beta_{A,B} = (\mathsf{outl}_{A,B}, \mathsf{outr}_{A,B})$. Having verified that this candidate indeed satisfies property (3.13) it follows from lemma (3.12) that $\alpha_{A,B}$ is a natural transformation. Lemma (3.11) may viewed as a way of predicting α such that $\alpha : F \leftarrow G$. If a functor is recoverable from G with β then the only candidate for α is $F\beta$. The lemma, however, does not guarantee that $F\beta$ has the slok property $F \leftarrow G$.

3.1.3 Category CoSlok

In Chapter 5, we need the opposite category of Slok as well; we denote this category by CoSlok. For reference purposes, we state the definition of category CoSlok and its dual properties explicitly.

A mapping F to $\mathcal{C}$ from $\mathcal{B}$ is an object of the category CoSlok if the source of Ff, for all f, only depends on the source of f, i.e.

$$(Ff)\triangleright = (Fg)\triangleright \quad \Leftarrow \quad f\triangleright = g\triangleright \quad \text{for all } f \text{ and } g. \tag{3.14}$$

Recall that “$\_\triangleright$” denotes the source operator. That is, $f\triangleright = B$ if $f : A \leftarrow B$.

From (3.14) it follows that mapping F induces a mapping on objects, say $F^{\triangleright}$, such that $(Ff)\triangleright = F^{\triangleright} f\triangleright$ i.e. $F^{\triangleright}A = (F\mathsf{id}_A)\triangleright$, or similarly, $F^{\triangleright} = \triangleright \circ F \circ \mathsf{id}$.

The arrows between the objects F and G, both mappings to $\mathcal{C}$-arrows from $\mathcal{B}$-arrows, are collections of arrows in $\mathcal{C}$ indexed by objects of $\mathcal{B}$ with a so-called coslok property i.e. for collection α, $\alpha : F \leftarrow G$ iff

$$Ff \cdot \alpha_A = Gf \quad \text{for each } f : A \leftarrow B. \tag{3.15}$$

For category CoSlok we have the dual notions of fuseable and recoverable:

Definition 3.16 Mapping G is *cofuseable* into mapping F iff

$$\mathsf{F}f \cdot \mathsf{G}g = \mathsf{F}(f \cdot g) \quad \text{for all } f \text{ and } g.$$

□

and

Definition 3.17 Mapping F is *corecoverable from* mapping G *with* the collection of arrows β iff $\mathsf{F}f = \mathsf{G}(\beta_A \cdot f)$ for each $f : A \leftarrow B$.
□

For CoSlok we have the dual typing rules of (3.6), (3.7) and (3.8). For all mappings H, we have,

$$\alpha_{\mathsf{H}^{\triangleright}} : \mathsf{FH} \leftarrow \mathsf{GH} \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \tag{3.18}$$

If mapping H' is cofuseable into mapping H, then

$$\mathsf{H}'\alpha : \mathsf{HF} \leftarrow \mathsf{HG} \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \tag{3.19}$$

For every functor H,

$$\mathsf{H}\alpha : \mathsf{HF} \leftarrow \mathsf{HG} \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G}\ . \tag{3.20}$$

Similarly, we have the duals of lemma's (3.11) and (3.12):

Lemma 3.21 If $\alpha : \mathsf{F} \leftarrow \mathsf{G}$ and there exists a *functor* that is corecoverable from F with β then $\alpha = \mathsf{G}\beta$. Hence, α is the unique arrow of type $\mathsf{F} \leftarrow \mathsf{G}$.
□

and

Lemma 3.22 If $\alpha : \mathsf{F} \xleftarrow{\mathsf{CoSlok}} \mathsf{G}$ and functor F' is corecoverable from F and functor G' is cofuseable into G then α is a natural transformation of type $\mathsf{F}' \xleftarrow{\mathsf{Fun}} \mathsf{G}'$.
□

3.2 The τ-Δ calculus

In this section we develop the tools to define and calculate with functors without the use of variables. Often functors (datatypes) that occur in programming are single-valued, but in general they are not. For instance, multi-valued, or vector-valued, functors occur in the definition of single-valued functors. The construct $\mathsf{F}\otimes\mathsf{G}$ is an example of such a definition: the (single-valued) functor $\mathsf{F}\otimes\mathsf{G}$ is the composition of the functor $\otimes$ after the (pair-valued) split (F, G) defined by $(\mathsf{F}, \mathsf{G})f = (\mathsf{F}f, \mathsf{G}f)$. A more complicated example is the binary functor which maps the pair (f, g) to $f + g \times g$. We want to be able to define such functors without mentioning the arguments explicitly. We identify a number of constructs with which we can construct these functors. The example of the binary functor, for instance, is constructed using projection as well as repetition (split), product and coproduct. Normally, one is not usually aware of the use of multi-valued functors and projections because their use is camouflaged by the use of variables. Having identified these constructs, we can give inductive definitions on the whole class of definable functors by giving a clause for each of the constructs. The purpose of this section is thus to develop the tools to define and calculate with functors in a variable-free way. This is achieved by introducing mechanisms for tupling of arbitrary variables and taking projections.

3.2.1 Arbitrary products of a category

For two categories $\mathcal{C}$ and $\mathcal{D}$ we can define the so-called product category $\mathcal{C} \times \mathcal{D}$. The arrows and the objects are pairs of arrows and objects, respectively, of $\mathcal{C}$ and $\mathcal{D}$. Composition is defined componentwise:

$$(f, g) \cdot (h, k) \;=\; (f \cdot h \,,\, g \cdot k) \;.$$

The identity on the object $(A \,, B)$ is the pair $(\mathsf{id}_A \,, \mathsf{id}_B)$. Furthermore, we have two projection functors Outl and Outr defined by

$$\mathsf{Outl}(f, g) = f \quad\wedge\quad \mathsf{Outr}(f, g) = g \;.$$

Note that if categories $\mathcal{C}$ and $\mathcal{D}$ are allegories then so is $\mathcal{C} \times \mathcal{D}$. All operations on allegory $\mathcal{C} \times \mathcal{D}$ are defined componentwise. For instance, the union of (R, S) and (T, U) is $(R \cup T, S \cup U)$ and the converse of (R, S) is $(R^\circ \,, S^\circ)$. Furthermore, Outl and Outr are relators.

Next we generalize the notion of a binary product of categories, to a k-fold product of a category. In the remainder of this thesis we only need vectors of arrows of the same category. So, $\mathcal{C}^k$ denotes the kth power of category $\mathcal{C}$. Every object of $\mathcal{C}^k$ is a vector of k objects A_i, $0 \le i < k$, of $\mathcal{C}$ and every arrow of $\mathcal{C}^k$ is a vector of k arrows f_i, $0 \le i < k$, of $\mathcal{C}$. We denote these vectors by $\Delta(i\colon 0 \le i < k\colon A_i)$ and $\Delta(i\colon 0 \le i < k\colon f_i)$, respectively. Composition in $\mathcal{C}^k$ we define componentwise, that is to say,

$$\Delta(i\colon 0 \le i < k\colon f_i) \;\cdot\; \Delta(i\colon 0 \le i < k\colon g_i) \;=\; \Delta(i\colon 0 \le i < k\colon f_i \cdot g_i) \tag{3.23}$$

and the identity arrow on $\Delta(i\colon 0 \le i < k\colon A_i)$ is $\Delta(i\colon 0 \le i < k\colon \mathsf{id}_{A_i})$. In other words, we assume that the mapping $\Delta(i\colon 0 \le i < k\colon \_)$ is a functor.

Furthermore, we assume the existence of k projections $\mathsf{Proj}_i : \mathcal{C} \leftarrow \mathcal{C}^k$, $0 \le i < k$, such that we have the following universal property, for all k arrows f_i, $0 \le i < k$, and each arrow g from $\mathcal{C}^k$,

$$g = \Delta(i\colon 0 \le i < k\colon f_i) \quad\equiv\quad \forall(i\colon 0 \le i < k\colon \mathsf{Proj}_i(g) = f_i) \;. \tag{3.24}$$

From now on we abbreviate $\Delta(i\colon 0 \le i < k\colon f_i)$ to $\Delta_k f_k$, and similarly $\Delta(i\colon 0 \le i < k\colon A_i)$ to $\Delta_k A_k$. Other vector notations that are sometimes used in the literature are the boldface notation, $\mathbf{f}$, or the notation $\vec{f}$. We have chosen for the notation $\Delta_k f_k$ because in the remainder of this thesis we consider vectors of vectors, i.e. matrices, and this notation shows explicitly how the different variables are being bound. For instance, we consider matrices like $\Delta_k \Delta_l f_{k,l}$ and $\Delta_l \Delta_k f_{k,l}$. Furthermore, the notation $\Delta_k f_k$ enables us to consider the properties of the *mapping* Δ_k on its own.

Just as we suppressed the need for the introduction of an index variable in the notation $\Delta_k f_k$, we abbreviate $\forall(i\colon 0 \leq i < k\colon \mathsf{Proj}_i(g) = f_i)$ to $\mathsf{Proj}_k(g) = f_k$. In general, we adopt the convention that all free occurrences of variable k within a formula are bound by a universal quantification with the smallest possible scope.

So, the universal property (3.24) is abbreviated to

$$g = \Delta_k f_k \quad \equiv \quad \mathsf{Proj}_k(g) = f_k \ . \tag{3.25}$$

From (3.25) follow the following two cancellation rules,

$$g = \Delta_k(\mathsf{Proj}_k(g)) \quad \wedge \quad \mathsf{Proj}_k(\Delta_k f_k) = f_k \ .$$

Now, since Δ_k is a functor it follows that Proj_k is a functor too:

$$\mathsf{Proj}_k(\Delta_k f_k \cdot \Delta_k g_k) = \mathsf{Proj}_k(\Delta_k(f_k \cdot g_k)) = f_k \cdot g_k = \mathsf{Proj}_k(\Delta_k f_k) \cdot \mathsf{Proj}_k(\Delta_k g_k)$$

A special instance of Δ_k we use throughout the remainder of this thesis is $\Delta_k f$, i.e. Δ_k applied to an argument which does not depend on k. So, $\Delta_k f$ is a vector containing k copies of f. If it is clear from the context what the value of k is, we drop the subscript k. So, we write Δf instead of $\Delta_k f$.

Note that the universal property (3.25) for $k = 2$ is *not* an instance of the universal property of products (2.2) since $\mathsf{Proj}_k(g)$ is the application of functor Proj_k to the arrow g and *not* a composition.

We can extend Δ_k to work on functors and natural transformations as well. We start with the extension on functors. That is, we define *tupling* of k functors $F_k : \mathcal{C} \leftarrow \mathcal{D}$ by,

$$\Delta_k F_k \ \triangleq \ \lambda f . \Delta_k(F_k f)$$

So, $\Delta_k F_k$ is a mapping of type $\mathcal{C}^k \leftarrow \mathcal{D}$. Note that it is safe to write $\Delta_k F_k f$ meaning either $(\Delta_k F_k) f$ or $\Delta_k(F_k f)$. The mapping $\Delta_k F_k$ takes an arrow f and delivers a vector of length k, with each ith component being $F_i f$. From the fact that Δ_k (on arrows of $\mathcal{C}$) is a functor it follows that $\Delta_k F_k$ is a functor too since

$$\Delta_k F_k f \cdot \Delta_k F_k g = \Delta_k(F_k f \cdot F_k g) = \Delta_k F_k(f \cdot g) \ ,$$

and

$$\Delta_k F_k \mathsf{id}_A = \Delta_k \mathsf{id}_{F_k A} = \mathsf{id}_{\Delta_k F_k A} \ .$$

Similarly as for functors, we define tupling of k natural transformations $\alpha_k : F_k \leftarrow G_k$ with $F_k, G_k : \mathcal{C} \leftarrow \mathcal{D}$ by

$$\Delta_k \alpha_k \ \triangleq \ \lambda A . \Delta_k((\alpha_k)_A) \ .$$

Again, it is safe to write $\Delta_k(\alpha_k)_A$ meaning either $\Delta_k((\alpha_k)_A)$ or $(\Delta_k(\alpha_k))_A$. We have that $\Delta_k \alpha_k$ is a natural transformation too, that is,

$$\Delta_k \alpha_k \ : \ \Delta_k F_k \leftarrow \Delta_k G_k \ ,$$

since for $R : A \leftarrow B$,

$$\Delta_k F_k R \cdot \Delta_k(\alpha_k)_B = \Delta_k(F_k R \cdot (\alpha_k)_B) = \Delta_k((\alpha_k)_A \cdot G_k R) = \Delta_k(\alpha_k)_A \cdot \Delta_k G_k R$$

Now, it follows that Δ_k is a functor on the functor category. The fact that Δ_k distributes over the composition of natural transformations follows from the fact that Δ_k distributes over the composition of arrows of $\mathcal{C}$,

$$(\Delta_k \alpha_k \cdot \Delta_k \beta_k)_A = \Delta_k(\alpha_k)_A \cdot \Delta_k(\beta_k)_A = \Delta_k((\alpha_k)_A \cdot (\beta_k)_A) = \Delta_k(\alpha_k \cdot \beta_k)_A$$

For the tupling of two functors and two natural transformations, we use the notation $\langle \_ , \_ \rangle$ thus $\langle F , G \rangle$ is the functor that maps arrow f to the pair of arrows (Ff , Gf), and $\langle \alpha , \beta \rangle$ maps object A to the pair of arrows (α_A , β_A).

For Δ_k on functors we have the following universal property, for all k functors $F_k : \mathcal{C} \leftarrow \mathcal{D}$ and all functors $G : \mathcal{C}^k \leftarrow \mathcal{D}$,

$$G = \Delta_k F_k \quad \equiv \quad \mathtt{Proj}_k G = F_k \tag{3.26}$$

since

$$\begin{array}{ll}
 & G = \Delta_k F_k \quad \equiv \quad \mathtt{Proj}_k G = F_k \\
= & \qquad \{ \quad \text{extensionality, definition } \Delta_k F_k \quad \} \\
 & \forall (f :: Gf = \Delta_k F_k f \quad \equiv \quad \mathtt{Proj}_k G f = F_k f) \\
= & \qquad \{ \quad \text{universal property (3.24)} \quad \} \\
 & \mathit{true}
\end{array}$$

Note that the universal property is a generalisation of the universal property of (binary) product (2.2) with respect to the functor category. (In contrast to (3.24), $\mathtt{Proj}_k G$ *is* the composition of $\mathtt{Proj}_k$ and G, not the application of $\mathtt{Proj}_k$ to G).This explains why we have chosen the Δ notation: it is the generalisation of the binary split "$\vartriangle$" operator. Like the universal property of binary products, the universal property (3.26) implies a number of properties. For instance, we obtain the following two cancellation laws: the reflection law,

$$G = \Delta_k(\mathtt{Proj}_k G) \ ,$$

and the computation rule

$$\mathtt{Proj}_k(\Delta_k F_k) = F_k \ .$$

Furthermore, we have the distribution law,

$$(\Delta_k F_k) G = \Delta_k(F_k G) \ . \tag{3.27}$$

The proof of all these facts proceeds the same as for binary products.

In view of rule (3.27) it is safe to write $\Delta_k F_k G$ meaning either $(\Delta_k F_k) G$ or $\Delta_k(F_k G)$.

If all k-fold vectors exist, we can define a so-called k-*fold product functor*, for k functors $F_k : \mathcal{C} \leftarrow \mathcal{D}$,

$$\Pi_k F_k \ \triangleq \ \Delta_k(F_k \mathtt{Proj}_k) : \mathcal{C}^k \leftarrow \mathcal{D}^k \ .$$

So, $\Pi_k F_k$ takes a vector of length k of arrows of $\mathcal{D}$ and applies F_i to the ith element of the vector resulting in a vector of length k of arrows of $\mathcal{C}$. The mapping Π_k is a functor on functors, i.e. we have,

$$(\Pi_k F_k)(\Pi_k G_k) = \Pi_k(F_k G_k) \quad \text{and} \quad \Pi_k \mathrm{Id} = \mathrm{Id} \ .$$

For Π_k and Δ_k, we have the generalized product-split fusion rule,

$$(\Pi_k F_k)(\Delta_k G_k) = \Delta_k(F_k G_k) \ .$$

A special instance of Π is the so-called kth power of a single functor. We define for a single functor $F : \mathcal{C} \leftarrow \mathcal{D}$,

$$F^k \ \triangleq \ \Pi_k F : \mathcal{C}^k \leftarrow \mathcal{D}^k \ .$$

Most often we use the constructions Δ_k, Π_k and "$\_^k$" on functors which have as source and target a power of $\mathcal{C}$. If it is clear from the context what the base category is, we write $k \leftarrow l$ instead of $\mathcal{C}^k \leftarrow \mathcal{C}^l$. We write $k{*}l$ for $(\mathcal{C}^k)^l$, and $(k{*}l){*}m$ for $((\mathcal{C}^k)^l)^m$, etc. We call k, $k{*}l$, etc., the *arity* of a category. So an arity is either a natural number or $k{*}l$ where k is an arity and l a natural number. Note that $*$ is *not* associative. We do however identify $1{*}k$ and $k{*}1$ with k. If $F : k \leftarrow l$, we say that $k \leftarrow l$ is the *arity* of functor F. A functor with arity $1 \leftarrow 1$ is called an *endofunctor* and a functor with arity $1 \leftarrow k$ for some k is called *single-valued*.

So, for k functors F_k with arity $l \leftarrow m$, $\Delta_k F_k$ has arity $l{*}k \leftarrow m$, and $\Pi_k F_k$ has arity $l{*}k \leftarrow m{*}k$. In particular, if F has arity $l \leftarrow m$ then F^k has arity $l{*}k \leftarrow m{*}k$.

Recall that a natural transformation $\alpha : F \leftarrow G$ for some functors $F, G : \mathcal{C} \leftarrow \mathcal{D}$ is a mapping of type $\alpha : \mathcal{C} \leftarrow \mathcal{D}$, i.e. α maps objects of $\mathcal{D}$ to arrows of $\mathcal{C}$. Just as for functors, we have a universal property for natural transformations. For all sets of k natural transformations $\alpha_k : \mathcal{C} \leftarrow \mathcal{D}$ and natural transformations $\beta : \mathcal{C}^k \leftarrow \mathcal{D}$,

$$\beta = \Delta_k \alpha_k \quad \equiv \quad \mathtt{Proj}_k \beta = \alpha_k \ . \tag{3.28}$$

From this universal property follows the same properties for natural transformations as for functors. For instance, we have the rule,

$$(\Delta_k \alpha_k)\beta = \Delta_k(\alpha_k \beta) \ , \tag{3.29}$$

and we have the functor "$\_^k$" on natural transformations, i.e. $\alpha^k : \mathcal{C}^k \leftarrow \mathcal{D}^k$ if $\alpha : \mathcal{C} \leftarrow \mathcal{D}$.

In view of rule (3.29) it is safe to write $\Delta_k \alpha_k \beta$ meaning either $(\Delta_k \alpha_k)\beta$ or $\Delta_k(\alpha_k \beta)$. By omitting the brackets, we can use rule (3.29) silently in a calculation. Depending on which calculation rule we want to apply we can choose how we want to interpret a formula. For instance, the formula,

$$\Delta_k F \alpha$$

may be interpreted in three different, but equivalent, ways: as $\Delta_k \circ F \circ \alpha$, the composition of the functor Δ_k, F and α, as $(\Delta_k . F) \circ \alpha$, the composition of the k-fold split of F after α, and as $\Delta_k.(F \circ \alpha)$, the k-fold split of the natural transformation $F \circ \alpha$.

In the next section, we prove results for general mappings of type $\mathcal{C} \leftarrow \mathcal{D}$, not necessarily functors or natural transformations. We use the same universal property as for functors and natural transformations but we drop the requirement that the mappings be functors or natural transformations. That is, we define for k *mappings* $F_k : \mathcal{C} \leftarrow \mathcal{D}$,

$$\Delta_k F_k \triangleq \lambda f . \Delta_k (F_k f)$$

which implies the universal property that for all k *mappings* $F_k : \mathcal{C} \leftarrow \mathcal{D}$ and *mappings* $G : \mathcal{C}^k \leftarrow \mathcal{D}$,

$$G = \Delta_k F_k \quad \equiv \quad \mathsf{Proj}_k G = F_k \ .$$

3.2.2 Matrix transposition

In this section we define matrix transposition. Using the universal properties of "$\triangle$" and "$\triangledown$" it is not difficult to show that $(f \triangledown g) \triangle (h \triangledown k) \ = \ (f \triangle h) \triangledown (g \triangle k)$. We can depict this rule by the following diagram

$$\begin{array}{ccc} f & \triangledown & g \\ & \triangle & \\ h & \triangledown & k \end{array} \quad = \quad \begin{array}{ccc} f & & g \\ \triangle & \triangledown & \triangle \\ h & & k \end{array}$$

Richard Bird calls such a rule an *abide law* [8], the contraction of the words "above" and "beside". Rewriting the equation in a variable-free way, we have that,

$$(\triangle)(\triangledown)^2 = (\triangledown)(\triangle)^2 \tau$$

where τ denotes the functor which maps $((f\,,g)\,,(h\,,k))$ to $((f\,,h)\,,(g\,,k))$. In other words, τ is matrix transposition on a matrix of dimension 2 by 2.

In the remainder of this thesis, we need matrix transposition for arbitrary dimensions. In this section we give a formal characterization of matrix transposition and use it to derive several properties.

3.2.3 Definition of matrix transposition

We define the functor[1] τ, $\tau \ : \ k{*}l \leftarrow l{*}k$, by the following slok property. We require for each k times l matrix of mappings $F_{k,l}$, $F_{k,l} : \mathcal{C} \leftarrow \mathcal{D}$, that,

$$\Delta_l \Delta_k F_{k,l} = \tau \Delta_k \Delta_l F_{k,l} \quad ,$$

or, equivalently,

$$\tau \ : \ \Delta_l \Delta_k F_{k,l} \xleftarrow{\mathsf{Slok}} \Delta_k \Delta_l F_{k,l} \ .$$

Note that the functor $\Pi_k \Pi_l$ is recoverable from $\Delta_k \Delta_l$ with $\alpha_{k,l} = \mathsf{Proj}_l \mathsf{Proj}_k$ since

$$\Delta_k \Delta_l \mathsf{Proj}_l \mathsf{Proj}_k \ = \ \Delta_k (\mathsf{Id}^l) \mathsf{Proj}_k \ = \ (\mathsf{Id}^l)^k$$

[1] Normally, we denote functors by capital Roman letters but since τ will be a natural transformation too, we denote it by a Greek letter.

(Note that $\mathsf{Proj}_l : 1 \leftarrow l$ and $\mathsf{Proj}_k : l \leftarrow l{*}k$.) Now, from lemma (3.11) it follows that the only possible candidate for τ is

$$\tau = \Delta_l \Delta_k (\mathsf{Proj}_l \mathsf{Proj}_k)$$

Indeed, for this choice of τ we have $\tau \ : \ \Delta_l\Delta_k F_{k,l} \xleftarrow{\mathsf{Slok}} \Delta_k \Delta_l F_{k,l}$ since

$$\tau \Delta_k \Delta_l F_{k,l} = \Delta_l \Delta_k \mathsf{Proj}_l \mathsf{Proj}_k \Delta_k \Delta_l F_{k,l} = \Delta_l \Delta_k F_{k,l}$$

Furthermore, since functor $\Pi_l \Pi_k$ is fusable into $\Delta_l \Delta_k$ it follows from lemma (3.12) that the functor τ is also a natural transformation of type $\Pi_l \Pi_k \leftarrow \Pi_k \Pi_l$, i.e.

$$(\Pi_l \Pi_k F_{k,l})\tau = \tau(\Pi_k \Pi_l F_{k,l})$$

Note that τ is an isomorphism. The inverse of τ, τ°, is derived from the definition of τ by interchanging k and l, i.e. $\tau^\circ = \Delta_k \Delta_l (\mathsf{Proj}_k \mathsf{Proj}_l)$. Indeed, τ° is the inverse of τ since using the defining slok property of τ and distribution,

$$\tau(\tau^\circ) \ = \ \tau \Delta_k \Delta_l \mathsf{Proj}_k \mathsf{Proj}_l \ = \ \Delta_l \Delta_k \mathsf{Proj}_k \mathsf{Proj}_l \ = \ (\mathrm{Id}^k)^l$$

Not only has τ its defining slok property, it has also the slok property

$$\Delta_l \Pi_k F_{k,l} = \tau \Pi_k \Delta_l F_{k,l} \ \ ,$$

since using the definition of Π,

$$\tau \Pi_k \Delta_l F_{k,l} \ = \ \tau \Delta_k \Delta_l F_{k,l} \mathsf{Proj}_k \ = \ \Delta_l \Delta_k F_{k,l} \mathsf{Proj}_k \ = \ \Delta_l \Pi_k F_{k,l}$$

Similarly, we have the slok property

$$\Pi_l \Delta_k F_{k,l} = \tau \Delta_k \Pi_l F_{k,l} \ \ .$$

Since the functor "$\_^k$" is an instance of Π, it follows that we have the following naturality property,

$$((F^k)^l)\tau = \tau((F^l)^k) \ \ .$$

and the two slok properties,

$$\Delta_l((F_l)^k) = \tau(\Delta_l F_l)^k \quad \wedge \quad (\Delta_k F_k)^l = \tau \Delta_k((F_k)^l) \ \ .$$

We adopt the convention that we use τ for arbitrary k and l. We rely on the context to determine what the dimensions of τ are. Note that if $k = 1$ or $l = 1$ then τ is the identity functor on $\mathcal{C}^l$ or $\mathcal{C}^k$, respectively.

Recall that if functor F has arity $l \leftarrow m$ then F^k has arity $l{*}k \leftarrow m{*}k$. Dually, we define

$${}^kF \quad \triangleq \quad \tau(F^k)\tau \ \ .$$

We write the superscript before its argument in order to facilitate the use of the arity rule: if F has arity $l \leftarrow m$ then ${}^kF$ has arity $k{*}l \leftarrow k{*}m$. (A disadvantage is that we have to be careful about parenthesisation since F^kG can be read as $(F^k)G$ or as $F({}^kG)$.)

Note that ${}^kF$ is also a functor since it is by definition the composition of three functors. Since τ is an isomorphism we can shunt one or both of the τ's to the other side. So, we have,

$$({}^kF)\tau = \tau(F^k) \quad , \quad \tau({}^kF) = (F^k)\tau \quad \text{and} \quad F^k \ = \ \tau({}^kF)\tau \ \ .$$

In this section we have listed a number of properties of τ. Most of these properties can be remembered or reconstructed by arity considerations. For instance, $\Delta_l((F_l)^k)$ has arity $k{*}l \leftarrow k$ and $(\Delta_l F_l)^k$ has arity $l{*}k \leftarrow k$. So this suggests that the relationship between the two constructs is matrix transposition τ.

3.3 Extending to allegories.

Just as for the product $\mathcal{C} \times \mathcal{D}$ of two allegories, it is the case that the kth power of an allegory is again an allegory if we define all operations componentwise. That is to say, we postulate that Δ_k (on relations) is a homomorphism. For instance, we have $(\Delta_k R_k)^\circ = \Delta_k(R_k{}^\circ)$, $\Delta_k R_k \cap \Delta_k S_k = \Delta_k(R_k \cap S_k)$, $\Delta_k R_k \setminus \Delta_k S_k = \Delta_k(R_k \backslash S_k)$, etcetera. In the same way as we proved that $\mathtt{Proj}_k$ is a functor it follows from the universal property (3.24) that $\mathtt{Proj}_k$ is a homomorphism too.

Now, it follows trivially that Δ_k and $\mathtt{Proj}_k$ are relators. Furthermore, the mapping Δ_k on relators, respects relators. That is to say, $\Delta_k F_k$ for k relators F_k is again a relator. Hence, the functor τ defined as $\Delta_l \Delta_k(\mathtt{Proj}_l \mathtt{Proj}_k)$ is a relator too. More specifically, since Δ_k respects homomorphisms, it follows that τ is a homomorphism. In particular, τ distributes over factors, i.e. $\tau(R\backslash S) = \tau R \setminus \tau S$.

Recall that for binary intersection we have the following universal property,

$$X \subseteq R \cap S \quad \equiv \quad X \subseteq R \ \wedge \ X \subseteq S$$

Similarly, we can define the intersection of a *vector* of relations. That is to say, we define the mapping $\cap : 1 \leftarrow k$ by, for $X : A \leftarrow B$ and $R \ : \ \Delta_k A \leftarrow \Delta_k B$

$$X \subseteq \cap R \quad \equiv \quad \Delta_k X \subseteq R$$

In other words, $\cap$ is the upper-adjoint of the relator Δ_k. Note that $\cap$ is a partial function: it is only defined on a vector of relations each with the same target and same source. Dually, we can define union $\cup : 1 \leftarrow k$ by

$$\cup R \subseteq X \quad \equiv \quad R \subseteq \Delta_k X$$

So, $\cup$ is the lower-adjoint of the relator Δ_k.

As we mentioned before, the reason why we investigated powers of categories, matrix transposition etc. was because we wanted to have a variable-free calculus of functors. But we can do more. For instance, we can express the statement

$$\Delta_k R_k \cap \Delta_k S_k = \Delta_k(R_k \cap S_k) \ \ , \tag{3.30}$$

without variables. Recall that for each relation X of $\mathcal{C}^k$, we have $X = \Delta_k X_k$ where $X_k = \mathtt{Proj}_k X$. So, statement (3.30) can be rewritten as, for all X and Y from $\mathcal{C}^k$,

$$X \cap Y = \Delta_k(X_k \cap Y_k) \ \ . \tag{3.31}$$

This is the requirement for a binary intersection. For the intersection of three relations of $\mathcal{C}^k$, we have

$$X \cap Y \cap Z = \Delta_k(X_k \cap Y_k \cap Z_k) \ \ . \tag{3.32}$$

So, for the intersection of l relations R_l of $\mathcal{C}^k$, we have

$$\cap(\Delta_l R_l) = \Delta_k(\cap \Delta_l (R_l)_k) \tag{3.33}$$

We have included the subscripts k and l in (3.33) in order to indicate the correct binding. Note that the arity of the intersection operator on the lhs is $k \leftarrow k{*}l$. With the goal of defining this operator in terms of single-valued intersection operators we continue with the rhs:

$$
\begin{array}{ll}
 & \Delta_k \cap \Delta_l (R_l)_k \\
= & \quad \{ \quad \text{product-split fusion: } \Delta_k \cap \ = (\cap^k)\Delta_k \quad \} \\
 & (\cap^k)\Delta_k\Delta_l(R_l)_k \\
= & \quad \{ \quad \text{matrix transposition} \quad \} \\
 & (\cap^k)\tau\Delta_l\Delta_k(R_l)_k \\
= & \quad \{ \quad \Delta_k(R_l)_k = R_l \quad \} \\
 & (\cap^k)\tau\Delta_l R_l
\end{array}
$$

Combining this with (3.33) and abstracting from $\Delta_l R_l = R$ yields

$$\cap = (\cap^k)\tau \ . \tag{3.34}$$

Note that the $\cap$ operator of the lhs has arity $k \leftarrow k{*}l$, whereas the $\cap$ operator of the rhs has arity $1 \leftarrow l$. Thus property (3.34) is what we expect from arity considerations.

Because τ is an isomorphism, equation (3.34) is equivalent to,

$$\cap\tau = \cap^k \ . \tag{3.35}$$

Similarly, we have for union,

$$\cup = (\cup^k)\tau \quad \text{and} \quad \cup\tau = \cup^k \ . \tag{3.36}$$

And we have for Proj_l,

$$\mathsf{Proj}_l = (\mathsf{Proj}_l{}^k)\tau \quad \text{and} \quad \mathsf{Proj}_l\tau = \mathsf{Proj}_l{}^k \ . \tag{3.37}$$

where Proj_l of the lhs has arity $k \leftarrow k{*}l$, whereas Proj_l of the rhs has arity $1 \leftarrow l$. We prove the second conjunct using the definition of τ,

$$\mathsf{Proj}_l\tau = \mathsf{Proj}_l\Delta_l\Delta_k\mathsf{Proj}_l\mathsf{Proj}_k = \Delta_k\mathsf{Proj}_l\mathsf{Proj}_k = (\mathsf{Proj}_l)^k$$

The last property we conclude this section with is the generalisation of the property that

$$(R \cap S) \cap (T \cap U) = (R \cap T) \cap (S \cap U)$$

In other words, $\cap$ abides with itself. In general, we have the property that,

$$\cap\cap = \cap\cap\tau$$

where the arities of the four intersection operators are, in order from left to right, $1 \leftarrow k$, $k \leftarrow k{*}l$, $1 \leftarrow l$, $l \leftarrow l{*}k$ and τ is of arity $l{*}k \leftarrow k{*}l$.

3.3.1 Regular datatypes

Having the notion of powers of categories we can define the notion of a regular relator. First, let $\mathcal{F}$ denote the class of relators which contains the basic relators — the constant relators K_A for all objects A, coproduct "$+$", product "$\times$" —, which also contains all the projection relators $\mathsf{Proj}_i : k \leftarrow k{*}l$ for $0 \leq i < l$, and is closed under composition, tupling (Δ_k) and the tree type construction.

Note that $\mathcal{F}$ contains all identity relators of type $k \leftarrow k$, for all k, since an identity relator is a projection relator on a vector of length 1. Furthermore $\mathcal{F}$ contains the relator τ and constructs like F^k and ${}^kF$. Now, we call the subclass of relators of $\mathcal{F}$ with arity $k \leftarrow l$ where k and l are natural numbers, the class of *regular* relators.

Chapter 4

Membership

In section 2.2 we argued that "datatype = relator" and we showed that for coproduct, product and recursively defined datatypes, a corresponding relator exists. For these relators a so-called membership relation exists. When we introduced the notion of a functor and when we generalized it to the notion of a relator, we talked about F-structures, and "F-structures containing elements" etc. In general, one would expect for a relator F the existence of a collection of membership relations $\mathsf{mem}_A \ : \ A \leftarrow FA$ for which we have the interpretation that $a \ \mathsf{mem}_A \ x$ iff a is an element of the F-structure x.

For instance, the interpretation of the pairing relator $FA = A \times A$ is that a is an element of $b \times c$ if $a = b$ or $a = c$. So, $\mathsf{outl} \cup \mathsf{outr}$ is the corresponding membership relation. Similarly, for the relator *List*, we want that a is an element of the list $[a_0, a_1, \ldots, a_{n-1}]$ when $a = a_i$ for some i , $0 \leq i < n$.

Several scientists have suggested that the datatypes used in computing *cannot* be equated with the notion of a relator or functor since these notions are too general. Oege de Moor proposed the idea that a datatype is the combination of a relator and a membership relation. This proposal, summarised by the slogan " datatype = relator + membership" was investigated jointly by Hoogendijk and De Moor and reported in [20]. In this chapter we extend that investigation further in order to include relators of arbitrary arity.

Most often, it is not too difficult to come up with a membership relation for a specific relator. For instance, it is possible to define membership relations inductively for the polynomial relators. However, we want to have a *generic* characterization of membership. Having a generic characterization, it is possible to prove properties for membership relations in general. If we only have an inductive definition for membership, every time we want to prove a property for membership, we have to consider each of the cases of the inductive definition. Such a case analysis is undesirable.

We start with the definition of a membership relation for an endorelator. Later on we generalize this definition to single-valued relators and then to arbitrary relators.

4.1 Definition of membership

What should a *generic* definition of a collection of membership relations mem_A for an arbitrary relator F look like? For the moment, we concentrate on a single element of a collection of membership relations, i.e. we try to find a characterisation of $\mathsf{mem}_A : A \leftarrow FA$ for some fixed relator F and object A.

Recall that the interpretation of FR is a relation between two F-structures of the same shape such that the corresponding elements of the two $F-$structures are related by R. In particular, the interpretation of FX for partial identity X, $X \subseteq \mathsf{id}_A$, is that FX is the partial identity defined on F-structures containing elements related by X. In other words, the partial identity FX represents the set of F-structures only containing elements of the subset of A which X represents. This interpretation leads to the following two requirements. First of all, for the set of F-structures FX, the membership relation should only return elements of set X. In a formula,

$$(\mathsf{mem}_A \cdot FX)_< \subseteq X \quad \text{for all } X, X \subseteq \mathsf{id}_A . \tag{4.1}$$

On the other hand, all F-structures for which the membership relation only returns elements of X, should be elements of FX. That is,

$$(\mathsf{mem}_A \cdot Y)_< \subseteq X \;\Rightarrow\; Y \subseteq FX \quad \text{for all } X \text{ and } Y, X \subseteq \mathsf{id}_A \text{ and } Y \subseteq \mathsf{id}_{FA}. \tag{4.2}$$

Since by monotonicity equation (4.1) is equivalent to $(\mathsf{mem}_A \cdot Y)_< \subseteq X \Leftarrow Y \subseteq FX$, we can combine both requirements in the following way,

$$(\mathsf{mem}_A \cdot Y)_< \subseteq X \;\equiv\; Y \subseteq FX \quad \text{for all } X \text{ and } Y, X \subseteq \mathsf{id}_A \text{ and } Y \subseteq \mathsf{id}_{FA}. \tag{4.3}$$

Note that requirement (4.3) is a Galois connection. We require relator F restricted to partial identities to be the upper adjoint of the mapping on partial identities $\lambda Y.\,(\mathsf{mem}_A \cdot Y)_<$. So, the interpretation of equation (4.3) is that FX is the largest set of F-structures for which membership mem_A only returns elements of set X.

A factor is by definition the upper Galois-adjoint of composition. Using the isomorphism between partial identities and conditionals we can get rid of the domain operator in equation (4.3) and restate it as an equality using a factor. So, rewriting equation (4.3) in terms of left-conditions, we get

$$\mathsf{mem}_A \cdot Y \cdot !^\circ \subseteq X \cdot !^\circ \quad \equiv \quad Y \cdot !^\circ \subseteq FX \cdot !^\circ$$

Introducing a factor in the lhs this becomes

$$Y \cdot !^\circ \subseteq \mathsf{mem}_A \backslash (X \cdot !^\circ) \quad \equiv \quad Y \cdot !^\circ \subseteq FX \cdot !^\circ$$

Since left-factors respect left-conditions the above equation (for all Y), is equivalent to

$$\mathsf{mem}_A \backslash (X \cdot !^\circ) \;=\; FX \cdot !^\circ \quad \text{for all } X, X \subseteq \mathsf{id}_A. \tag{4.4}$$

For the moment, we take equation (4.4) as the generic definition of membership mem_A for all objects A. Next we show that by using extensionality, definition (4.4) can be generalized in the following way: For each $R : B \leftarrow A$,

$$FR \;\cdot\; \mathsf{mem}_A \backslash \mathsf{id}_A \;=\; \mathsf{mem}_B \backslash R \tag{4.5}$$

which follows by extensionality from, for all points $p \in A$,

$$
\begin{array}{ll}
 & \mathsf{F}R \cdot \mathsf{mem}_A \backslash \mathsf{id}_A \cdot p \\
= & \quad \{ \quad \text{function } p\text{, factors} \quad \} \\
 & \mathsf{F}R \cdot \mathsf{mem}_A \backslash p \\
= & \quad \{ \quad p = p_< \cdot !^\circ\text{, (4.4)} \quad \} \\
 & \mathsf{F}R \cdot \mathsf{F}p_< \cdot !^\circ \\
= & \quad \{ \quad \text{relator } \mathsf{F}\text{, domains} \quad \} \\
 & \mathsf{F}(R \cdot p)_< \cdot !^\circ \\
= & \quad \{ \quad \text{(4.4), } (R \cdot p)_< \cdot !^\circ = R \cdot p \quad \} \\
 & \mathsf{mem}_B \backslash (R \cdot p) \\
= & \quad \{ \quad \text{function } p\text{, factors} \quad \} \\
 & \mathsf{mem}_B \backslash R \cdot p
\end{array}
$$

Since F is a functor, equation (4.5) is equivalent to

$$\mathsf{F}R \cdot \mathsf{mem} \backslash S \;=\; \mathsf{mem} \backslash (R \cdot S) \; . \tag{4.6}$$

That equation (4.6) is indeed a generalization of the definition of membership (4.4) follows by taking $R := X$ and $S := !^\circ$ and the fact that $\mathsf{mem} \backslash !^\circ = !^\circ$. Since this direction, i.e. showing that equation (4.6) implies equation (4.4), does *not* depend on extensionality, we take equation (4.5) instead of (4.4) as the definition of membership for relator F:[1]

Definition 4.7 (Membership) A collection of arrows mem is a *membership* relation of relator F if for each $R : B \leftarrow A$,

$$\mathsf{F}R \cdot \mathsf{mem}_A \backslash \mathsf{id}_A \;=\; \mathsf{mem}_B \backslash R \; . \tag{4.8}$$

□

Whenever it is necessary to differentiate between membership relations for different relators, we denote the membership relation for relator F by $\mathsf{mem}.\mathsf{F}$.

4.1.1 Uniqueness of membership (endorelators)

Next we show that definition (4.7) is a "good" definition. We show that a relator has at most one membership relation. To be more precise, we prove that membership $\mathsf{mem}.\mathsf{F}$ is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \mathsf{F}$. That is, we have $\mathsf{mem}.\mathsf{F} : \mathrm{Id} \hookleftarrow \mathsf{F}$, and

$$\alpha \subseteq \mathsf{mem}.\mathsf{F} \;\Leftarrow\; \alpha : \mathrm{Id} \hookleftarrow \mathsf{F} \; . \tag{4.9}$$

The proof of this fact depends on the so-called *identification axiom*. The identification axiom states that id is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \mathrm{Id}$. We have not encountered

[1] There are non-extensional allegories for which equation (4.5) for some relator F is really stronger than equation (4.4).

the identification axiom in the literature so far. However, it is not difficult to show that extensionality implies the identification axiom. Furthermore, the identification axiom is a necessary condition for the proof of property (4.9). Since id is a membership of the identity relator Id, property (4.9) for relator Id implies the identification axiom.

So, we prove the following lemma assuming identification:

Lemma 4.10 If mem is a membership relation of relator F, then mem is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \mathsf{F}$.

Proof We first prove that $\mathsf{mem} : \mathrm{Id} \hookleftarrow \mathsf{F}$,

$$
\begin{array}{ll}
 & \mathsf{mem} \cdot \mathsf{F}R \subseteq R \cdot \mathsf{mem} \\
\equiv & \quad \{ \quad \text{factors} \quad \} \\
 & \mathsf{F}R \subseteq \mathsf{mem} \backslash (R \cdot \mathsf{mem}) \\
\equiv & \quad \{ \quad \mathsf{mem} \text{ membership (4.6)} \quad \} \\
 & \mathsf{F}R \subseteq \mathsf{F}R \cdot \mathsf{mem} \backslash \mathsf{mem} \\
\equiv & \quad \{ \quad \mathsf{id} \subseteq \mathsf{mem} \backslash \mathsf{mem} \quad \} \\
 & \mathit{true}
\end{array}
$$

Next, we show that mem is the largest natural transformation of its type. Let α be a natural transformation of type $\mathrm{Id} \hookleftarrow \mathsf{F}$. It is easy to prove that $\mathsf{mem} \backslash \mathsf{id} : \mathsf{F} \hookleftarrow \mathrm{Id}$, since for $R : A \leftarrow B$,

$$
(\mathsf{mem} \backslash \mathsf{id})_A \cdot R \subseteq \mathsf{mem}_A \backslash R = \mathsf{F}R \cdot (\mathsf{mem} \backslash \mathsf{id})_B
$$

That $\alpha \subseteq \mathsf{mem}$ now follows from the identification axiom:

$$
\begin{array}{ll}
 & \alpha \\
\subseteq & \quad \{ \quad \mathsf{id}_\mathsf{F} \subseteq \mathsf{mem} \backslash \mathsf{mem} \quad \} \\
 & \alpha \cdot \mathsf{mem} \backslash \mathsf{mem} \\
= & \quad \{ \quad \mathsf{mem} \text{ membership (4.8)} \quad \} \\
 & \alpha \cdot \mathsf{F}\mathsf{mem} \cdot (\mathsf{mem} \backslash \mathsf{id})_\mathsf{F} \\
\subseteq & \quad \{ \quad \alpha : \mathrm{Id} \hookleftarrow \mathsf{F} \quad \} \\
 & \mathsf{mem} \cdot \alpha_\mathsf{F} \cdot (\mathsf{mem} \backslash \mathsf{id})_\mathsf{F} \\
\subseteq & \quad \{ \quad \alpha : \mathrm{Id} \hookleftarrow \mathsf{F},\ \mathsf{mem} \backslash \mathsf{id} : \mathsf{F} \hookleftarrow \mathrm{Id}, \text{ hence, } \alpha \cdot \mathsf{mem} \backslash \mathsf{id} : \mathrm{Id} \hookleftarrow \mathrm{Id}, \\
 & \quad\quad\quad \mathsf{id} \text{ is the largest natural transformation of this type} \quad \} \\
 & \mathsf{mem}
\end{array}
$$

□

Note that we assumed that mem_A has type $A \leftarrow \mathsf{F}A$. So, we assumed that relator F is an endorelator. In the following subsection we extend the definition of membership to non-endo relators.

4.1.2 Membership for non-endo relators

Next we generalize the definition of membership for arbitrary relators. We first derive the definition of membership for single-valued relators, i.e. relators of arity $1 \leftarrow k$ for some k. For the moment, we assume $k = 2$. What should the definition of the membership relation $\mathsf{mem}.\otimes$ for a binary relator $\otimes$ look like? The interpretation of a binary relator $\otimes$ as a datatype-former is that a structure of type $A_0{\otimes}A_1$, for objects A_0 and A_1, contains data at two places: the left and right argument. In other words, the membership relation for $\otimes$ has two components, $\mathsf{mem}_0 \,:\, A_0 \leftarrow A_0{\otimes}A_1$ and $\mathsf{mem}_1 \,:\, A_1 \leftarrow A_0{\otimes}A_1$, one for each argument. Just as in the endo case, for all $\otimes$-structures that are elements of the set $X_0{\otimes}X_1$, for partial identities X_0 and X_1, the component for the left argument should only return elements of X_0, and the component for the right argument only elements of X_1. In formulae, for all $X_0 \subseteq \mathsf{id}_{A_0}$and $X_1 \subseteq \mathsf{id}_{A_1}$

$$(\mathsf{mem}_0 \cdot X_0{\otimes}X_1){<} \subseteq X_0 \quad \text{and} \quad (\mathsf{mem}_1 \cdot X_0{\otimes}X_1){<} \subseteq X_1 \ .$$

On the other hand, all $\otimes$-structures for which the left component and right component of the membership relation only return elements of X_0 and X_1, respectively, should be elements of the set $X_0{\otimes}X_1$. That is, for all $Y \subseteq \mathsf{id}_{A_0 \otimes A_1}$

$$(\mathsf{mem}_0 \cdot Y){<} \subseteq X_0 \ \wedge \ (\mathsf{mem}_1 \cdot Y){<} \subseteq X_1 \quad \Rightarrow \quad Y \subseteq X_0{\otimes}X_1$$

Combining both equations we demand that for all partial identities $X_0 \subseteq \mathsf{id}_{A_0}$, $X_1 \subseteq \mathsf{id}_{A_1}$ and $Y \subseteq \mathsf{id}_{A_0 \otimes A_1}$,

$$(\mathsf{mem}_0 \cdot Y){<} \subseteq X_0 \ \wedge \ (\mathsf{mem}_1 \cdot Y){<} \subseteq X_1 \quad \equiv \quad Y \subseteq X_0{\otimes}X_1 \tag{4.11}$$

The lhs of above equation can be rewritten as

$$((\mathsf{mem}_0 \,, \mathsf{mem}_1) \cdot \Delta_2 Y){<} \ \subseteq \ (X_0 \,, X_1)$$

Recall that Δ_2 denotes the doubling functor: $\Delta_2 Y = (Y\,,Y)$. Now, writing $A = (A_0\,,A_1)$, $X = (X_0\,,X_1)$ and $\mathsf{mem} = (\mathsf{mem}_0\,,\mathsf{mem}_1)$, equation (4.11) becomes, for all partial identities $X \subseteq \mathsf{id}_A$ and $Y \subseteq \mathsf{id}_{(\otimes)A}$,

$$(\mathsf{mem} \cdot \Delta_2 Y){<} \subseteq X \ \equiv \ Y \subseteq (\otimes)X$$

The above equation for a membership relation for a binary relator suggests the equation for an arbitrary single-valued relator F of arity $1 \leftarrow k$. Specifically, we demand that mem be of arity $k \leftarrow k$ and is such that for all objects A, $A \in \mathcal{C}^k$, we have for all partial identities $X \subseteq \mathsf{id}_A$ and $Y \subseteq \mathsf{id}_{FA}$,

$$(\mathsf{mem}_A \cdot \Delta_k Y){<} \subseteq X \ \equiv \ Y \subseteq FX$$

with $\mathsf{mem}_A \ : \ A \leftarrow \Delta_k FA$.

As in the case of endorelators we want to get rid of the domain operator and restate the requirement as an equality using factors. Rewriting the above equation in terms of conditions, we get

$$\mathsf{mem}_A \cdot \Delta_k Y \cdot !^{\circ} \subseteq X \cdot !^{\circ} \ \equiv \ Y \cdot !^{\circ} \subseteq FX \cdot !^{\circ} \tag{4.12}$$

Let us concentrate on the lhs. Since all operations in allegory $\mathcal{C}^k$ are defined componentwise, it follows that $!_{\Delta_k A} = \Delta_k !_A$. Hence, since Δ_k is a relator,

$$\Delta_k Y \cdot !^\circ = \Delta_k Y \cdot \Delta_k !^\circ = \Delta_k (Y \cdot !^\circ)$$

Using this and introducing a factor, the lhs of (4.12) can be rewritten as,

$$\Delta_k (Y \cdot !^\circ) \subseteq \mathsf{mem}_A \backslash (X \cdot !^\circ)$$

Exploiting the Galois connection between Δ and $\cap$, we get

$$Y \cdot !^\circ \subseteq \cap (\mathsf{mem}_A \backslash (X \cdot !^\circ))$$

Hence, in combination with equation (4.12), our requirement for mem_A becomes

$$\cap (\mathsf{mem}_A \backslash (X \cdot !^\circ)) = FX \cdot !^\circ \quad \text{for each } X,\, X \subseteq \mathsf{id}_A. \tag{4.13}$$

Again, by a similar calculation as for the endo case, we can prove using extensionality that equation (4.13) is equivalent to, for each $R : B \leftarrow \Delta_k A$,

$$FR \cdot \cap (\mathsf{mem} \backslash \mathsf{id})_{\Delta_k A} = \cap (\mathsf{mem}_B \backslash R) \tag{4.14}$$

since for all points $p \in A$,

$$
\begin{array}{ll}
 & FR \cdot \cap (\mathsf{mem} \backslash \mathsf{id})_{\Delta_k A} \cdot p \\
= & \quad \{ \quad \text{function } p\text{, factors, intersection} \quad \} \\
 & FR \cdot \cap (\mathsf{mem}_{\Delta_k A} \backslash \Delta_k p) \\
= & \quad \{ \quad \Delta_k p \;=\; \Delta_k p_< \;\cdot\; !^\circ\text{, (4.13)} \quad \} \\
 & FR \;\cdot\; F\Delta_k p_< \;\cdot\; !^\circ \\
= & \quad \{ \quad \text{relator } F\text{, domains} \quad \} \\
 & F(R \cdot \Delta_k p)_< \;\cdot\; !^\circ \\
= & \quad \{ \quad \text{(4.13), } (R \cdot \Delta_k p)_< \cdot !^\circ = R \cdot \Delta_k p \quad \} \\
 & \cap (\mathsf{mem}_B \backslash (R \cdot \Delta_k p)) \\
= & \quad \{ \quad \text{function } p\text{, factors, intersection} \quad \} \\
 & \cap (\mathsf{mem}_{\Delta_k A} \backslash R) \cdot p
\end{array}
$$

We take equation (4.14) as the definition of membership for a single-valued relator F :

Definition 4.15 (Membership) A collection of arrows mem of arity $k \leftarrow k$ such that, for each A, $\mathsf{mem}_A \;:\; A \leftarrow \Delta_k FA$ is a *membership* relation of single-valued relator $F : 1 \leftarrow k$, if for each $R : B \leftarrow \Delta_k A$,

$$FR \cdot \cap (\mathsf{mem} \backslash \mathsf{id})_{\Delta_k A} = \cap (\mathsf{mem}_B \backslash R) \;. \tag{4.16}$$

□

Note that definition (4.15) is indeed a generalization of definition (4.7) since $\Delta_1 = \mathrm{Id}$ and $\cap : 1 \leftarrow 1 = \mathrm{Id}$. Furthermore, using the fact that F is a functor, it follows that,

$$\mathsf{F}R \cdot \cap(\mathsf{mem}\backslash S) = \cap(\mathsf{mem}\backslash(R \cdot S)) \tag{4.17}$$

Instantiating the definition of membership for binary relator $\otimes$, it follows that the pair of collections of relations $\mathsf{meml}_{A,B} : A \leftarrow A \otimes B$ and $\mathsf{memr}_{A,B} : B \leftarrow A \otimes B$ is a membership relation of $\otimes$ iff, for each $R : A \leftarrow C$ and $S : B \leftarrow C$,

$$R \otimes S \cdot (\mathsf{meml}_{C,C}\backslash \mathsf{id}_C \cap \mathsf{memr}_{C,C}\backslash \mathsf{id}_C) = \mathsf{meml}_{A,B}\backslash R \cap \mathsf{memr}_{A,B}\backslash S\ . \tag{4.18}$$

For instance, it follows that the pair $(\mathsf{outl}, \mathsf{outr})$ is a membership of product since property (2.64) states that $R \vartriangle S = \mathsf{outl}_{A,B}\backslash R \cap \mathsf{outr}_{A,B}\backslash S$. Hence, requirement (4.18) is just the product-split fusion rule:

$$R \times S \cdot \mathsf{id}_C \vartriangle \mathsf{id}_C = R \vartriangle S\ .$$

The decomposition lemma

The construction of a membership relation for a relator F of arity $1 \leftarrow k$, $k > 1$, was guided by the idea that mem_A should be a vector of length k of membership relations for each of the k arguments of F. Indeed, this can be proven using the definition (4.15). For instance, define relator G by $\mathsf{G}R = \mathsf{F}(R, \mathsf{id}_A)$ for some object $A \in \mathcal{C}^{k-1}$. Hence, G contains the elements of the first argument of F. Let mem denote a membership of F. Assume that $\mathsf{mem} = (\alpha, \beta)$ for $\alpha \in \mathcal{C}$ and $\beta \in \mathcal{C}^{k-1}$, i.e. α is the first component of mem and β the remainder. Now, the claim is that the collection of relations γ defined by $\gamma_B = \alpha_{(B,A)}$ is a membership relation of G. To prove this claim we first observe that for $R : C \leftarrow B$,

$$
\begin{array}{cl}
 & \gamma_C\backslash R \\
= & \{\quad \text{definition } \gamma,\ X\backslash\top\!\!\top = \top\!\!\top,\ \cap\top\!\!\top = \top\!\!\top, \text{ intersection}\quad\} \\
 & \alpha_{(C,A)}\backslash R \ \cap\ \cap(\beta_{(C,A)}\backslash\top\!\!\top_{A,\Delta_{k-1}B}) \\
= & \{\quad S \cap (\cap X) = \cap(S, X),\ \backslash \text{ componentwise}\quad\} \\
 & \cap((\alpha, \beta)_{(C,A)}\backslash(R, \top\!\!\top_{A,\Delta_{k-1}B})) \\
= & \{\quad \text{definition } \alpha \text{ and } \beta\quad\} \\
 & \cap(\mathsf{mem}_{(C,A)}\backslash(R, \top\!\!\top_{A,\Delta_{k-1}B}))
\end{array}
$$

Using this, the proof that γ is the membership of G is as follows:

$$
\begin{array}{cl}
 & \gamma_C\backslash R \\
= & \{\quad \text{calculation above}\quad\} \\
 & \cap(\mathsf{mem}_{(C,A)}\backslash(R, \top\!\!\top_{A,\Delta_{k-1}B})) \\
= & \{\quad \mathsf{mem} \text{ membership of } \mathsf{F}, \text{(4.17)}: \\
 & \quad\ (R, \top\!\!\top_{A,\Delta_{k-1}B}) = (R, \mathsf{id}_A) \cdot (\mathsf{id}_B, \top\!\!\top_{A,\Delta_{k-1}B})\quad\} \\
 & \mathsf{F}(R, \mathsf{id}_A) \cdot \cap(\mathsf{mem}_{(B,A)}\backslash(\mathsf{id}_B, \top\!\!\top_{A,\Delta_{k-1}B}))
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \text{definition } \mathsf{G}\text{, calculation above} \quad \} \\
& \mathsf{GR} \cdot \gamma_{\mathsf{B}} \backslash \mathsf{id}_{\mathsf{B}}
\end{array}
$$

In particular, we have for binary relator $\otimes$ with membership relation $\mathsf{mem} = (\mathsf{meml}, \mathsf{memr})$ that $\mathsf{meml}_{\_,A}$ is a membership relation of the sectioning $(\otimes \mathsf{id}_A)$. That is to say, we have for $\mathsf{R} : \mathsf{B} \leftarrow \mathsf{C}$,

$$\mathsf{R} \otimes \mathsf{id}_A \cdot \mathsf{meml}_{C,A} \backslash \mathsf{id}_C = \mathsf{meml}_{B,A} \backslash \mathsf{R} \ .$$

Multi-valued relators

So far we have only considered single-valued relators, i.e. relators of arity $1 \leftarrow k$. Next we consider the membership relation for relator F of arity $l \leftarrow k$. Recall that for $\mathsf{F} : 1 \leftarrow k$, $\mathsf{mem}.\mathsf{F}$ is a vector of membership relations for each of the k arguments of F. Generalizing this property to a relator of arity $l \leftarrow k$, we require that $\mathsf{mem}.\mathsf{F}$ is a vector of length k, each component corresponding to one of the arguments of F and being itself a vector of length l. Thus we require $\mathsf{mem}.\mathsf{F}$ to have arity $l{*}k \leftarrow k$. This leads to the following definition,

Definition 4.19 For relator F of arity $l \leftarrow k$, the *membership* relation, $\mathsf{mem}.\mathsf{F} : l{*}k \leftarrow k$, is defined by,

$$\mathsf{mem}.\mathsf{F} \triangleq \tau\Delta_l(\mathsf{mem}.\mathsf{F}_l) \ .$$

□

For $\mathsf{mem}.\mathsf{F}$ we have the following property.

Lemma 4.20 For $\mathsf{F} : l \leftarrow k$, we have for each R, $\mathsf{R} \in \mathcal{C}^k$ and $\mathsf{R} : \mathsf{B} \leftarrow \Delta_k A$,

$$\mathsf{FR} \cdot \cap(\mathsf{mem}.\mathsf{F} \backslash (\Delta_l)^k \mathsf{id})_{\Delta_k A} = \cap((\mathsf{mem}.\mathsf{F})_{\mathsf{B}} \backslash (\Delta_l)^k \mathsf{R}) \ . \tag{4.21}$$

Proof We first calculate that

$$
\begin{array}{ll}
& \cap((\mathsf{mem}.\mathsf{F})_{\mathsf{B}} \backslash (\Delta_l)^k \mathsf{R}) \\
= & \{ \quad \text{definition } \mathsf{mem}.\mathsf{F},\ (\Delta_l)^k = \tau\Delta_l \text{ where } \tau : l{*}k \leftarrow k{*}l \quad \} \\
& \cap(\tau\Delta_l(\mathsf{mem}.\mathsf{F}_l)_{\mathsf{B}} \backslash \tau\Delta_l \mathsf{R}) \\
= & \{ \quad \tau \text{ and } \Delta_l \text{ distribute over } \backslash \quad \} \\
& \cap\tau\Delta_l((\mathsf{mem}.\mathsf{F}_l)_{\mathsf{B}} \backslash \mathsf{R}) \\
= & \{ \quad \cap\tau = \cap^l,\ (\cap^l)\Delta_l = \Delta_l\cap \quad \} \\
& \Delta_l\cap((\mathsf{mem}.\mathsf{F}_l)_{\mathsf{B}} \backslash \mathsf{R})
\end{array}
$$

Using the above calculation the lemma is straightforward to prove:

$$
\begin{array}{ll}
& \mathsf{FR} \cdot \cap(\mathsf{mem}.\mathsf{F} \backslash (\Delta_l)^k \mathsf{id})_{\Delta_k A} = \cap((\mathsf{mem}.\mathsf{F})_{\mathsf{B}} \backslash (\Delta_l)^k \mathsf{R}) \\
\equiv & \{ \quad \mathsf{F} = \Delta_l \mathsf{F}_l\text{, calculation above} \quad \} \\
& \Delta_l \mathsf{F}_l \mathsf{R} \ \cdot \ \Delta_l\cap(\mathsf{mem}.\mathsf{F}_l \backslash \mathsf{id})_{\Delta_k A} \ = \ \Delta_l\cap((\mathsf{mem}.\mathsf{F}_l)_{\mathsf{B}} \backslash \mathsf{R})
\end{array}
$$

$$
\begin{array}{ll}
\equiv & \{ \quad \Delta_l \text{ functor } \} \\
& \Delta_l(\mathsf{F}_l R \cdot \cap(\mathsf{mem}.\mathsf{F}_l \backslash \mathsf{id})_{\Delta_k A}) = \Delta_l \cap((\mathsf{mem}.\mathsf{F}_l)_B \backslash R) \\
\equiv & \{ \quad \text{for each } l \ \} \\
& \mathsf{F}_l R \cdot \cap(\mathsf{mem}.\mathsf{F}_l \backslash \mathsf{id})_{\Delta_k A} = \cap((\mathsf{mem}.\mathsf{F}_l)_B \backslash R)
\end{array}
$$

Hence, the lemma follows from the definition of membership (4.15) for each $\mathsf{mem}.\mathsf{F}_l$.
□

As a matter of fact, one can use property (4.21) as the defining equation of membership for an arbitrary relator, and show that the choice $\mathsf{mem}.\mathsf{F} = \tau\Delta_l(\mathsf{mem}.\mathsf{F}_l)$ satisfies equation (4.21).

4.1.3 Uniqueness of membership (arbitrary relators)

In subsection 4.1.1 we showed that an endorelator has at most one membership relation. In this section we generalise this result for arbitrary relators of arity $l \leftarrow k$. We start with single-valued relators. That is to say, we prove that equation (4.16) has at most one solution.

Lemma 4.22 If mem is a membership relation of single-valued relator F, then mem is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \Delta\mathsf{F}$.

Proof First we prove that mem is a natural transformation:

$$
\begin{array}{ll}
& \mathsf{mem} \cdot \Delta\mathsf{F}R \subseteq R \cdot \mathsf{mem} \\
\equiv & \{ \quad \text{factors } \} \\
& \Delta\mathsf{F}R \subseteq \mathsf{mem} \backslash (R \cdot \mathsf{mem}) \\
\equiv & \{ \quad \Delta \leftrightarrow \cap \ \} \\
& \mathsf{F}R \subseteq \cap(\mathsf{mem} \backslash (R \cdot \mathsf{mem})) \\
\equiv & \{ \quad \text{mem membership (4.17) } \} \\
& \mathsf{F}R \subseteq \mathsf{F}R \cdot \cap(\mathsf{mem} \backslash \mathsf{mem}) \\
\equiv & \{ \quad \mathsf{id} \subseteq \mathsf{mem} \backslash \mathsf{mem}, \cap\mathsf{id} = \mathsf{id} \ \} \\
& \textit{true}
\end{array}
$$

Next we show that mem is the largest natural transformation of its type. Let α be a natural transformation of type $\mathrm{Id} \hookleftarrow \Delta\mathsf{F}$. Then we have to show that $\alpha \subseteq \mathsf{mem}$:

$$
\begin{array}{ll}
& \alpha \\
\subseteq & \{ \quad \text{factors, intersection: } \mathsf{id}_{\Delta\mathsf{F}} \subseteq \Delta\cap(\mathsf{mem} \backslash \mathsf{mem}) \ \} \\
& \alpha \cdot \Delta\cap(\mathsf{mem} \backslash \mathsf{mem}) \\
= & \{ \quad \text{mem membership (4.17), } \Delta \text{ functor } \} \\
& \alpha \cdot \Delta\mathsf{F}\mathsf{mem} \cdot \Delta\cap(\mathsf{mem} \backslash \mathsf{id})_{\Delta\mathsf{F}} \\
\subseteq & \{ \quad \alpha : \mathrm{Id} \hookleftarrow \Delta\mathsf{F} \ \}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathsf{mem} \cdot \alpha_{\Delta\mathsf{F}} \cdot \Delta\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta\mathsf{F}} \\
= & \qquad \{ \quad \text{lifting} \quad \} \\
 & \mathsf{mem} \cdot (\alpha_{\Delta} \cdot \Delta\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta})_{\mathsf{F}} \\
\subseteq & \qquad \{ \quad \text{claim: } \alpha_{\Delta} \cdot \Delta\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta} \subseteq \Delta\mathsf{id} \quad \} \\
 & \mathsf{mem}
\end{array}
$$

Next we verify the claim. It is easy to prove that

$$\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta} \;:\; \mathsf{F}\Delta \hookleftarrow \mathrm{Id} \;, \tag{4.23}$$

since for $\mathsf{R} : \mathsf{B} \leftarrow \mathsf{A}$,

$$\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta\mathsf{B}} \cdot \mathsf{R} \subseteq \cap((\mathsf{mem}\backslash\mathsf{id})_{\Delta\mathsf{B}} \cdot \Delta\mathsf{R}) \subseteq \cap(\mathsf{mem}_{\Delta\mathsf{B}} \backslash \Delta\mathsf{R}) = \mathsf{F}\Delta\mathsf{R} \cdot \cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta\mathsf{A}}$$

Furthermore, notice that it follows from the identification axiom that $\Delta\mathsf{id}$ is the largest natural transformation of type $\Delta \hookleftarrow \Delta$. Using both facts we calculate

$$
\begin{array}{ll}
 & \alpha_{\Delta} \cdot \Delta\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta} \subseteq \Delta\mathsf{id} \\
\Leftarrow & \qquad \{ \quad \text{identification axiom} \quad \} \\
 & \alpha_{\Delta} \cdot \Delta\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta} \;:\; \Delta \hookleftarrow \Delta \\
\Leftarrow & \qquad \{ \quad \text{typing rule} \quad \} \\
 & \alpha_{\Delta} \;:\; \Delta \hookleftarrow \Delta\mathsf{F}\Delta \;\wedge\; \Delta\cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta} \;:\; \Delta\mathsf{F}\Delta \hookleftarrow \Delta \\
\Leftarrow & \qquad \{ \quad \text{typing rules} \quad \} \\
 & \alpha \;:\; \mathrm{Id} \hookleftarrow \Delta\mathsf{F} \;\wedge\; \cap(\mathsf{mem}\backslash\mathsf{id})_{\Delta} \;:\; \mathsf{F}\Delta \hookleftarrow \mathrm{Id} \\
\equiv & \qquad \{ \quad \text{assumption } \alpha\text{, property (4.23)} \quad \} \\
 & \mathit{true}
\end{array}
$$

□

Hence, we have shown that a relator of arity $1 \leftarrow \mathsf{k}$ has at most one membership relation. Recall that the membership relation for a relator F of arity $\mathsf{l} \leftarrow \mathsf{k}$ is defined by $\mathsf{mem.F} = \tau\Delta_{\mathsf{l}}(\mathsf{mem.F}_{\mathsf{l}})$. Since $\mathsf{mem.F}_{\mathsf{l}} \;:\; \mathrm{Id} \hookleftarrow \Delta_{\mathsf{k}}\mathsf{F}_{\mathsf{l}}$, we have

$$\mathsf{mem.F} \;\;:\;\; \tau\Delta_{\mathsf{l}} \;\hookleftarrow\; \tau\Delta_{\mathsf{l}}\Delta_{\mathsf{k}}\mathsf{F}_{\mathsf{l}} \;=\; (\Delta_{\mathsf{l}})^{\mathsf{k}} \hookleftarrow \Delta_{\mathsf{k}}\Delta_{\mathsf{l}}\mathsf{F}_{\mathsf{l}} \;=\; (\Delta_{\mathsf{l}})^{\mathsf{k}} \hookleftarrow \Delta_{\mathsf{k}}\mathsf{F}$$

We claim that $\mathsf{mem.F}$ is the largest natural transformation of its type too. First, because Δ_{k} is an order-isomorphism it follows that $\Delta_{\mathsf{l}}(\mathsf{mem.F}_{\mathsf{l}})$, where each $\mathsf{mem.F}_{\mathsf{l}}$ the largest natural transformation of type $\mathrm{Id} \hookleftarrow \Delta_{\mathsf{k}}\mathsf{F}_{\mathsf{l}}$, is the largest natural transformation of type $\Delta_{\mathsf{l}} \hookleftarrow \Delta_{\mathsf{l}}\Delta_{\mathsf{k}}\mathsf{F}_{\mathsf{l}}$. Now, since τ is an isomorphism it follows that $\mathsf{mem.F}$ is the largest natural transformation of its type. Hence, every relator of arity $\mathsf{l} \leftarrow \mathsf{k}$ has at most one membership relation.

4.2 Membership for regular relators

All regular relators have membership. In this section we give a definition of the membership relation for the regular relators. The construction of the membership relations is inductive over the structure of the regular relators. We start with the basic relators:

Lemma 4.24

$$\begin{aligned} \mathsf{mem}.\mathrm{Id} &= \mathsf{id} \\ \mathsf{mem}.\mathrm{K}_A &= \perp\!\!\perp_{\_,A} \\ \mathsf{mem}.\times &= \langle \mathsf{outl}\, , \mathsf{outr} \rangle \\ \mathsf{mem}.+ &= \langle \mathsf{inl}^\circ\, , \mathsf{inr}^\circ \rangle \end{aligned}$$

Proof Note that for $R : B \leftarrow C$, we have $\mathsf{id}_B \backslash R = R$ and $\perp\!\!\perp_{B,A} \backslash R = \top\!\!\top_{A,C}$. Using this, the verification of membership for Id and K_A is straightforward. Verification of membership for product follows from property (2.64) and from the product-split fusion rule. Similarly, verification of membership for coproduct follows from the converse of property (2.47) and the converse of the coproduct-junc fusion rule.
□

4.2.1 Membership of projections

In this section we give the membership relation for a projection relator $\mathtt{Proj}_i : 1 \leftarrow k$ for $0 \leq i < k$. We have to construct a natural transformation $\mathsf{mem}.\mathtt{Proj}_i$ of arity $k \leftarrow k$. Recall that the interpretation of $\mathsf{mem}.\mathtt{Proj}_i$ is a vector of length k of membership relations for each of the k arguments of F. Since it is only the ith argument of $\mathtt{Proj}_i$ which contributes to the data that is stored in a $\mathtt{Proj}_i$-structure, we suspect that $\mathsf{mem}.\mathtt{Proj}_i$ is vector of length k with id at position i and $\perp\!\!\perp$ elsewhere. Indeed, we have,

Lemma 4.25 For a projection relator $\mathtt{Proj}_i : 1 \leftarrow k$, $0 \leq i < k$ its membership relation is defined by

$$\mathsf{mem}.\mathtt{Proj}_i \ \triangleq\ \Delta_k(i = k \rightarrow \mathsf{id}_{\mathtt{Proj}_i}\, , \perp\!\!\perp_{\mathtt{Proj}_k\, ,\, \mathtt{Proj}_i}) \ .$$

Where $f(j) = (i = j \rightarrow g\, , h)$ denotes a so-called conditional such that $f(j) = g$ if $i = j$ and $f(j) = h$ if $i \neq j$.

Proof We first calculate for $R : B \leftarrow \Delta_k A$,

$$\begin{aligned} & (\Delta_k(i = k \rightarrow \mathsf{id}_{\mathtt{Proj}_i}\, , \perp\!\!\perp_{\mathtt{Proj}_k\, ,\, \mathtt{Proj}_i}))_B \backslash R \\ = \quad & \{ \quad B = \Delta_k B_k, \text{ conditionals; } R = \Delta_k R_k \quad \} \\ & \Delta_k(i = k \rightarrow \mathsf{id}_{B_i}\, , \perp\!\!\perp_{B_k\, ,\, B_i}) \backslash \Delta_k R_k \\ = \quad & \{ \quad \Delta_k \text{ distributes over } \backslash \quad \} \\ & \Delta_k((i = k \rightarrow \mathsf{id}_{B_i}\, , \perp\!\!\perp_{B_k\, ,\, B_i}) \backslash R_k) \\ = \quad & \{ \quad \text{conditionals, } \mathsf{id}_{B_i} \backslash R_i = R_i,\ \perp\!\!\perp_{B_k\, ,\, B_i} \backslash R_k = \top\!\!\top_{B_i\, ,\, A} \quad \} \\ & \Delta_k(i = k \rightarrow R_i\, , \top\!\!\top_{B_i\, ,\, A}) \end{aligned}$$

So, $\cap((\Delta_k(i = k \rightarrow \mathsf{id}_{\mathsf{Proj}_i}, \perp\!\!\perp_{\mathsf{Proj}_k,\mathsf{Proj}_i}))_B \backslash R) = \cap(\Delta_k(i = k \rightarrow R_i, \top\!\!\top_{B_i,A})) = R_i$. Using this, equation (4.16) is trivially verified.
□

4.2.2 Membership of composition

In this section we construct the membership relation for the composition of two relators. For a composition of endorelators, the membership relation is trivial. That is to say,

$$\mathsf{mem.FG} = \mathsf{mem.G} \cdot (\mathsf{mem.F})_G$$

It is easy to verify that $\mathsf{mem.FG}$ satisfies equation (4.8).

Before constructing the membership relation for the general case, we first look at a concrete example. We take $F := \times$ and $G := \langle H, I \rangle$, then $FG = (\times)\langle H, I\rangle = H \times I$. A value stored in a $HA \times IA$ structure is stored either in the H-structure or in the I-structure. This suggest that

$$\mathsf{mem.}(H \times I) = \mathsf{mem.}H \cdot \mathsf{outl}_{\langle H, I\rangle} \quad \cup \quad \mathsf{mem.}I \cdot \mathsf{outr}_{\langle H, I\rangle}$$

More generally, if $F : 1 \leftarrow k$ and $G : k \leftarrow 1$, we expect that,

$$\mathsf{mem.FG} = \cup_k(\mathsf{mem.G}_k \cdot ((\mathsf{mem.F})_k)_G)$$

(where $\cup_k$ is a shorthand for $\cup\Delta_k$), since a value stored in an FGA structure is stored in one of the G_k-structures at the kth argument of F. Recall that $(\mathsf{mem.F})_k$ is the membership relation of the kth argument of F. Now, since $\mathsf{mem.G}_k = (\mathsf{mem.G})_k$ and $((\mathsf{mem.F})_k)_G = ((\mathsf{mem.F})_G)_k$, we have

$$\begin{aligned} &\cup_k(\mathsf{mem.G}_k \cdot ((\mathsf{mem.F})_k)_G) = \cup_k((\mathsf{mem.G})_k \cdot ((\mathsf{mem.F})_G)_k) \\ &= \cup(\mathsf{mem.G} \cdot (\mathsf{mem.F})_G) \end{aligned}$$

Hence, we define for $F : 1 \leftarrow k$ and $G : k \leftarrow 1$,

$$\mathsf{mem.FG} = \cup(\mathsf{mem.G} \cdot (\mathsf{mem.F})_G)$$

Next we consider the most general case that $F : 1 \leftarrow k$ and $G : k \leftarrow l$. Since FG has arity $1 \leftarrow l$, membership $(\mathsf{mem.FG})_A$ is an element of $\mathcal{C}^l$, i.e. a vector of l membership relations, one for each of the l arguments of FG. Now, because the lth argument of FG is F applied to the lth argument of G, we expect,

$$\mathsf{mem.FG} = \Delta_l \cup ((\mathsf{mem.G})_l \cdot (\mathsf{mem.F})_G) \tag{4.26}$$

We try to rewrite this definition of $\mathsf{mem.FG}$ into a form without quantification:

$$\begin{aligned} &\Delta_l \cup ((\mathsf{mem.G})_l \cdot (\mathsf{mem.F})_G) \\ = &\qquad \{ \quad \Delta_l(\cup) = (\cup^l)\Delta_l,\ \Delta_l \text{ functor} \quad \} \\ &\cup^l(\Delta_l(\mathsf{mem.G})_l \cdot \Delta_l(\mathsf{mem.F})_G) \end{aligned}$$

$$
\begin{array}{ll}
= & \{ \quad \Delta_l(\mathsf{mem.G})_l = \mathsf{mem.G},\ \cup^l = \cup\tau \text{ where } \tau : l{*}k \leftarrow k{*}l,\ \tau \text{ functor} \quad \} \\
& \cup(\tau(\mathsf{mem.G}) \cdot \tau\Delta_l(\mathsf{mem.F})_G) \\
= & \{ \quad \tau\Delta_l = (\Delta_l)^k \quad \} \\
& \cup(\tau(\mathsf{mem.G}) \cdot ((\Delta_l)^k)(\mathsf{mem.F})_G)
\end{array}
$$

Hence we define,

Definition 4.27 For relator $F : 1 \leftarrow k$ and relator $G : k \leftarrow l$,

$$\mathsf{mem.FG} \triangleq \cup(\tau(\mathsf{mem.G}) \cdot ((\Delta_l)^k)(\mathsf{mem.F})_G) \ . \tag{4.28}$$

□

In order to verify that $\mathsf{mem.FG}$ is indeed the membership relation of FG, we first prove the following lemma,

Lemma 4.29 For each $R : B \leftarrow A$,

$$\cap((\mathsf{mem.FG})_B \backslash R) \ = \ \cap((\mathsf{mem.F})_{GB} \backslash \cap((\mathsf{mem.G})_B \backslash (\Delta_k)^l R)) \ .$$

Proof

$$
\begin{array}{ll}
& \cap((\mathsf{mem.FG})_B \backslash R) \\
= & \{ \quad \text{definition } \mathsf{mem.FG} \ (4.28) \quad \} \\
& \cap((\cup(\tau(\mathsf{mem.G})_B \cdot ((\Delta_l)^k)(\mathsf{mem.F})_{GB}))\backslash R) \\
= & \{ \quad \text{factors: } (\cup X)\backslash Y = \cap(X \backslash \Delta_k Y) \text{ for } X \in \mathcal{C}^k \text{ and } Y \in \mathcal{C} \quad \} \\
& \cap\cap((\tau(\mathsf{mem.G})_B \cdot ((\Delta_l)^k)(\mathsf{mem.F})_{GB}) \backslash \Delta_k R) \\
= & \{ \quad \text{factors: } (X \cdot Y)\backslash Z = Y\backslash(X\backslash Z) \quad \} \\
& \cap\cap(((\Delta_l)^k)(\mathsf{mem.F})_{GB} \backslash (\tau(\mathsf{mem.G})_B \backslash \Delta_k R)) \\
= & \{ \quad (\Delta_l)^k = \tau\Delta_l,\ \Delta_k = \tau(\Delta_k)^l \text{ where } \tau : l{*}k \leftarrow k{*}l, \\
& \qquad \tau \text{ distributes over } \backslash \text{ (twice)} \quad \} \\
& \cap\cap\tau(\Delta_l(\mathsf{mem.F})_{GB} \backslash ((\mathsf{mem.G})_B \backslash (\Delta_k)^l R)) \\
= & \{ \quad \cap\cap\tau = \cap\cap,\ \cap(\Delta_l X \backslash Y) = X \backslash \cap Y \text{ for } Y \in \mathcal{C}^l \text{ and } X \in \mathcal{C} \quad \} \\
& \cap((\mathsf{mem.F})_{GB} \backslash \cap((\mathsf{mem.G})_B \backslash (\Delta_k)^l R))
\end{array}
$$

□

Using lemma (4.29) it is straightforward to verify that equation (4.16) holds for $\mathsf{mem.FG}$. We assume that $\mathsf{mem.F}$ and $\mathsf{mem.G}$ are memberships for F and G. Then for each $R : A \leftarrow B$,

$$
\begin{array}{ll}
& FGR \cdot \cap((\mathsf{mem.FG})_A \backslash \mathsf{id}_A) \\
= & \{ \quad \text{lemma } (4.29) \quad \}
\end{array}
$$

$$
\begin{array}{ll}
& \mathsf{FGR} \cdot \cap((\mathsf{mem.F})_{\mathsf{GA}} \backslash \cap((\mathsf{mem.G})_{\mathsf{A}} \backslash (\Delta_{\mathsf{k}})^{\iota}\mathsf{id}_{\mathsf{A}})) \\
= & \quad \{ \quad \mathsf{mem.F} \text{ membership, (4.17)} \quad \} \\
& \cap((\mathsf{mem.F})_{\mathsf{GB}} \backslash (\mathsf{GR} \cdot \cap((\mathsf{mem.G})_{\mathsf{A}} \backslash (\Delta_{\mathsf{k}})^{\iota}\mathsf{id}_{\mathsf{A}}))) \\
= & \quad \{ \quad \mathsf{mem.G} \text{ membership, (4.21)} \quad \} \\
& \cap((\mathsf{mem.F})_{\mathsf{GB}} \backslash \cap((\mathsf{mem.G})_{\mathsf{B}} \backslash (\Delta_{\mathsf{k}})^{\iota}\mathsf{R})) \\
= & \quad \{ \quad \text{lemma (4.29)} \quad \} \\
& \cap((\mathsf{mem.FG})_{\mathsf{B}} \backslash \mathsf{R})
\end{array}
$$

Hence, $\mathsf{mem.FG}$ is membership of FG.

For binary relator $\otimes$ and endorelators F and G it follows that

$$\mathsf{mem.(F{\otimes}G)} = \mathsf{mem.F} \cdot ((\mathsf{mem.}{\otimes})_0)_{\langle \mathsf{F},\mathsf{G}\rangle} \cup \mathsf{mem.G} \cdot ((\mathsf{mem.}{\otimes})_1)_{\langle \mathsf{F},\mathsf{G}\rangle} \ .$$

For instance, we have for membership of $\mathsf{F} \times \mathsf{G}$ and $\mathsf{F} + \mathsf{G}$ that,

$$
\begin{array}{lcl}
\mathsf{mem.(F \times G)} & = & \mathsf{mem.F} \cdot \mathsf{outl}_{\langle \mathsf{F},\mathsf{G}\rangle} \cup \mathsf{mem.G} \cdot \mathsf{outr}_{\langle \mathsf{F},\mathsf{G}\rangle} \\
\mathsf{mem.(F + G)} & = & \mathsf{mem.F} \cdot \mathsf{inl}^{\circ}_{\langle \mathsf{F},\mathsf{G}\rangle} \cup \mathsf{mem.G} \cdot \mathsf{inr}^{\circ}_{\langle \mathsf{F},\mathsf{G}\rangle}
\end{array}
$$

4.2.3 Membership of the power relator

As one would expect, membership of the power relator P is $\in$. We have to show that

$$\mathsf{PR} \cdot \in\backslash \mathsf{id} = \in\backslash \mathsf{R}$$

One inclusion follows from naturality of $\in$, $\in : \mathrm{Id} \leftarrow \mathrm{E}$, or equivalently, $\in : \mathrm{Id} \hookleftarrow \mathsf{P}$:

$$
\begin{array}{ll}
& \mathsf{PR} \cdot \in\backslash\mathsf{id} \subseteq \in\backslash\mathsf{R} \\
\equiv & \quad \{ \quad \text{factors} \quad \} \\
& \in \cdot\, \mathsf{PR} \cdot \in\backslash\mathsf{id} \subseteq \mathsf{R} \\
\Leftarrow & \quad \{ \quad \in : \mathrm{Id} \hookleftarrow \mathsf{P} \quad \} \\
& \mathsf{R} \cdot \in \cdot \in\backslash\mathsf{id} \subseteq \mathsf{R} \\
\equiv & \quad \{ \quad \text{factors, monotonicity} \quad \} \\
& \mathit{true}
\end{array}
$$

For the other inclusion, let (f,g) be a tabulation of R. Then,

$$
\begin{array}{ll}
& \mathsf{PR} \cdot \in\backslash\mathsf{id} \\
= & \quad \{ \quad (\mathsf{f},\mathsf{g}) \text{ tabulation of } \mathsf{R}\text{, relator } \mathsf{P} \text{ extension of } \mathrm{E} \quad \} \\
& \mathrm{E}\mathsf{f} \cdot (\mathrm{E}\mathsf{g})^{\circ} \cdot \in\backslash\mathsf{id} \\
= & \quad \{ \quad \mathrm{E}\mathsf{g} \text{ function, factors} \quad \} \\
& \mathrm{E}\mathsf{f} \cdot (\in \cdot\, \mathrm{E}\mathsf{g})\backslash\mathsf{id}
\end{array}
$$

$$
\begin{array}{ll}
= & \{ \quad \in : \mathsf{Id} \leftarrow \mathsf{E} \quad \} \\
& \mathsf{Ef} \cdot (\mathsf{g} \cdot \in)\backslash \mathsf{id} \\
= & \{ \quad \mathsf{g} \text{ function, factors} \quad \} \\
& \mathsf{Ef} \cdot \in\backslash \mathsf{g}^{\circ} \\
\supseteq & \{ \quad \text{claim: } \mathsf{Ef} \cdot \in\backslash \mathsf{R} \supseteq \in\backslash(\mathsf{f} \cdot \mathsf{R}) \quad \} \\
& \in\backslash(\mathsf{f} \cdot \mathsf{g}^{\circ}) \\
= & \{ \quad (\mathsf{f},\mathsf{g}) \text{ tabulation of } \mathsf{R} \quad \} \\
& \in\backslash \mathsf{R}
\end{array}
$$

The claim in the penultimate step we prove by a pseudo-pointwise argument. Let (x,z) be a tabulation of $\in\backslash(\mathsf{f} \cdot \mathsf{R})$. That is:

$$x \cdot z^{\circ} = \in\backslash(\mathsf{f} \cdot \mathsf{R}) \ . \tag{4.30}$$

We aim to construct a function y such that

$$x \cdot y^{\circ} \subseteq \mathsf{Ef} \quad \wedge \quad y \cdot z^{\circ} \subseteq \in\backslash \mathsf{R} \ , \tag{4.31}$$

then it follows that

$$\in\backslash(\mathsf{f} \cdot \mathsf{R}) \ = \ x \cdot z^{\circ} \ \subseteq \ x \cdot y^{\circ} \cdot y \cdot z^{\circ} \ \subseteq \ \mathsf{Ef} \cdot \in\backslash \mathsf{R}$$

The first inclusion follows because y is a function and functions are total. The second inclusion follows from (4.31) and monotonicity of composition. Next, we construct function y such that equation (4.31) holds:

$$
\begin{array}{ll}
& x \cdot y^{\circ} \subseteq \mathsf{Ef} \quad \wedge \quad y \cdot z^{\circ} \subseteq \in\backslash \mathsf{R} \\
\equiv & \{ \quad \text{shunting of functions and factors} \quad \} \\
& x = \mathsf{Ef} \cdot y \quad \wedge \quad \in \cdot y \subseteq \mathsf{R} \cdot z \\
\equiv & \{ \quad y := \Lambda\alpha, \text{ fusion and cancellation} \quad \} \\
& x = \Lambda(\mathsf{f} \cdot \alpha) \quad \wedge \quad \alpha \subseteq \mathsf{R} \cdot z \\
\equiv & \{ \quad \alpha := \beta \cap \mathsf{R} \cdot z, \text{ intersection} \quad \} \\
& x = \Lambda(\mathsf{f} \cdot (\beta \cap \mathsf{R} \cdot z)) \\
\equiv & \{ \quad \text{universal property of } \Lambda \text{ (2.29)} \quad \} \\
& \in \cdot x = \mathsf{f} \cdot (\beta \cap \mathsf{R} \cdot z) \\
\equiv & \{ \quad \beta := \mathsf{f}^{\circ} \cdot \gamma, \text{ modular identity} \quad \} \\
& \in \cdot x = \gamma \cap \mathsf{f} \cdot \mathsf{R} \cdot z \\
\equiv & \{ \quad \in \cdot x \subseteq \mathsf{f} \cdot \mathsf{R} \cdot z \\
& \qquad \equiv \quad \{ \quad \text{shunting of functions and factors} \quad \} \\
& \qquad\qquad x \cdot z^{\circ} \subseteq \in\backslash(\mathsf{f} \cdot \mathsf{R})
\end{array}
$$

$$\begin{array}{ll} \equiv & \{ \quad \text{assumption (4.30)} \quad \} \\ & \mathit{true} \\ & \text{intersection} \quad \} \\ & \in \cdot\, x = \gamma \end{array}$$

Thus equation (4.31) is valid with y assigned to $\Lambda(f^\circ \cdot \in \cdot\, x \cap R \cdot z)$.

Surprisingly, the above proof is very long. It is very difficult, only using that P is an extension of E and $\in\, : \mathrm{Id} \leftarrow E$, to show that $\in$ satisfies the defining equation of the membership relation of P. However, in [40] the following property is proved using a similar proof technique:

$$\in\backslash\in\, : P \leftarrow E \tag{4.32}$$

Using this property, the verification that $\in$ is the membership relation of P is almost trivial. Since ΛR is a function it follows that $\in\backslash R = \in\backslash(\in \cdot \Lambda R) = \in\backslash\in \cdot \Lambda R$. Using this we prove:

$$\begin{array}{ll} & PR \cdot \in\backslash id \\ = & \{ \quad \text{property above} \quad \} \\ & PR \cdot \in\backslash\in \cdot \Lambda id \\ = & \{ \quad \in\backslash\in : P \leftarrow E \quad \} \\ & \in\backslash\in \cdot ER \cdot \Lambda id \\ = & \{ \quad \text{fusion rule: } ER \cdot \Lambda S = \Lambda(R \cdot S) \quad \} \\ & \in\backslash\in \cdot \Lambda R \\ = & \{ \quad \text{property above} \quad \} \\ & \in\backslash R \end{array}$$

The way we proved that $\in$ is the membership relation of P is an adaption of the proof in [40] of property (4.32). As a matter of fact, having the fact that $\in$ is membership of P, naturality of $\in\backslash\in$ follows trivially:

$$\begin{array}{ll} & PR \cdot \in\backslash\in \\ = & \{ \quad \in \text{ membership of } P \quad \} \\ & \in\backslash(R \cdot \in) \\ = & \{ \quad \in : \mathrm{Id} \leftarrow E \quad \} \\ & \in\backslash(\in \cdot ER) \\ = & \{ \quad ER \text{ function, factors} \quad \} \\ & \in\backslash\in \cdot ER \end{array}$$

4.3 Membership of tree types

In this section we construct the membership relations for tree types. Let $\otimes$ be a binary relator and let $(\mathsf{in}\,,\mathsf{T})$ be its tree type. We try to construct the membership of T. This construction is inductive in the sense that we assume the existence of membership of $\otimes$ and construct the membership of T in terms of it. Let $\mathsf{mem} = (\mathsf{meml}\,,\mathsf{memr})$ denote the membership relation of $\otimes$.

We know that relators T and $\mathrm{Id}\otimes\mathsf{T}$ are isomorphic: in is a natural isomorphism of type $\mathsf{T} \leftarrow \mathrm{Id}\otimes\mathsf{T}$. It is easy to verify that if F and G are isomorphic, that is, there exists a natural isomorphism $\alpha : \mathsf{F} \leftarrow \mathsf{G}$, then $\mathsf{mem.G}\cdot\alpha^\circ$ and $\mathsf{mem.F}\cdot\alpha$ are memberships of F and G, respectively. Hence, we have,

$$\mathsf{mem.T} = \mathsf{mem.}(\mathrm{Id}\otimes\mathsf{T})\cdot\mathsf{in}^\circ \tag{4.33}$$

From the definition of membership of composition, it follows that

$$\mathsf{mem.}(\mathrm{Id}\otimes\mathsf{T}) = \mathsf{meml}_{\langle\mathrm{Id}\,,\mathsf{T}\rangle} \cup \mathsf{mem.T}\cdot\mathsf{memr}_{\langle\mathrm{Id}\,,\mathsf{T}\rangle}$$

Instantiating this in equation (4.33) and distributing in° over the composition, yields the following recursive equation:

$$\mathsf{mem.T} = \mathsf{meml}_{\langle\mathrm{Id}\,,\mathsf{T}\rangle}\cdot\mathsf{in}^\circ \cup \mathsf{mem.T}\cdot\mathsf{memr}_{\langle\mathrm{Id}\,,\mathsf{T}\rangle}\cdot\mathsf{in}^\circ \tag{4.34}$$

Now, define

$$\mathsf{root} = \mathsf{meml}_{\langle\mathrm{Id}\,,\mathsf{T}\rangle}\cdot\mathsf{in}^\circ \ \ .$$

The natural transformation $\mathsf{root} : \mathrm{Id} \hookleftarrow \mathsf{T}$ returns an element that occurs at the root of a tree. Similarly, define

$$\mathsf{branch} = \mathsf{memr}_{\langle\mathrm{Id}\,,\mathsf{T}\rangle}\cdot\mathsf{in}^\circ \ \ .$$

The natural transformation $\mathsf{branch} : \mathsf{T} \hookleftarrow \mathsf{T}$ returns an immediate subtree of its argument. Using both definitions, equation (4.34) can be rewritten as,

$$\mathsf{mem.T} = \mathsf{root} \cup \mathsf{mem.T}\cdot\mathsf{branch} \tag{4.35}$$

In general, $\mathsf{R}\cdot\mathsf{S}^*$ is a solution of the equation:

$$\mathsf{X} \ :: \ \mathsf{X} = \mathsf{R} \cup \mathsf{X}\cdot\mathsf{S}$$

where S^* is the reflexive, transitive closure of S. That is, S^* is by definition the least solution of:

$$\mathsf{X} \ :: \ \mathsf{X} = \mathsf{id} \cup \mathsf{X}\cdot\mathsf{S} \ \ .$$

So, equation (4.35) suggests that we should define

$$\mathsf{mem.T} \triangleq \mathsf{root}\cdot\mathsf{branch}^* \ \ .$$

The interpretation of $\mathsf{root}\cdot\mathsf{branch}^*$ is that branch^* selects an arbitrary subtree and root takes the root element of it. That this definition indeed defines membership of a tree type follows from the following lemma:

Lemma 4.36 Let $\otimes$ be a binary relator with tree type (in,T). Then

$$(\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R} = [\![\mathsf{id}\otimes;\ \mathsf{meml}\backslash\mathsf{R}\cap\mathsf{memr}\backslash\mathsf{id}]\!]$$

Proof We aim to show

$$\mathsf{in}^\circ\cdot(\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R} = \mathsf{id}\otimes(\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R}\cdot(\mathsf{meml}\backslash\mathsf{R}\cap\mathsf{memr}\backslash\mathsf{id})$$

This may be done as follows

$$\begin{array}{ll}
 & \mathsf{in}^\circ\cdot(\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R} \\
= & \quad\{\quad \text{in function, factors} \quad\} \\
 & (\mathsf{root}\cdot\mathsf{branch}^*\cdot\mathsf{in})\backslash\mathsf{R} \\
= & \quad\{\quad \text{closure} \quad\} \\
 & ((\mathsf{root}\ \cup\ \mathsf{root}\cdot\mathsf{branch}^*\cdot\mathsf{branch})\cdot\mathsf{in})\backslash\mathsf{R} \\
= & \quad\{\quad \text{composition over union, defs } \mathsf{root} \text{ and } \mathsf{branch} \quad\} \\
 & (\mathsf{meml}\ \cup\ \mathsf{root}\cdot\mathsf{branch}^*\cdot\mathsf{memr})\backslash\mathsf{R} \\
= & \quad\{\quad \text{factors} \quad\} \\
 & \mathsf{meml}\backslash\mathsf{R}\cap\mathsf{memr}\backslash((\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R}) \\
= & \quad\{\quad \mathsf{mem} = (\mathsf{meml}\,,\mathsf{memr}) \text{ membership of binary relator } \otimes \quad\} \\
 & \mathsf{id}\otimes(\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R}\cdot(\mathsf{meml}\backslash\mathsf{R}\cap\mathsf{memr}\backslash\mathsf{id})
\end{array}$$

□

As a corollary to the above lemma, we have

Corollary 4.37 Let $\otimes$ be a binary relator with tree type (in,T). Then

$$\mathsf{mem}.\mathsf{T} = \mathsf{root}\cdot\mathsf{branch}^*$$

Proof We argue

$$\begin{array}{ll}
 & \mathsf{TR}\cdot(\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{id} \\
= & \quad\{\quad \text{lemma (4.36)} \quad\} \\
 & \mathsf{TR}\cdot[\![\mathsf{meml}\backslash\mathsf{id}\cap\mathsf{memr}\backslash\mathsf{id}]\!] \\
= & \quad\{\quad [\![_]\!]\leftrightarrow(\![_]\!),\ \text{tree type fusion} \quad\} \\
 & [\![\mathsf{R}\otimes\mathsf{id}\cdot(\mathsf{meml}\backslash\mathsf{id}\cap\mathsf{memr}\backslash\mathsf{id})]\!] \\
= & \quad\{\quad \mathsf{mem} = (\mathsf{meml},\mathsf{memr}) \text{ membership of binary relator } \otimes \quad\} \\
 & [\![\mathsf{meml}\backslash\mathsf{R}\cap\mathsf{memr}\backslash\mathsf{id}]\!] \\
= & \quad\{\quad \text{lemma (4.36)} \quad\} \\
 & (\mathsf{root}\cdot\mathsf{branch}^*)\backslash\mathsf{R}
\end{array}$$

□

4.4 A Counter-example

Finally, we mention the fact that not every relator has a membership relation. Consider the case that $\mathcal{C} = \mathcal{B}^2$ for some $\mathcal{B}$ and define the *swapping* relator on $\mathcal{C}$:

$$\mathsf{F}(\mathsf{R},\mathsf{S}) = (\mathsf{S},\mathsf{R})$$

Suppose there is a natural transformation (α_0,α_1) of type $\mathrm{Id} \hookleftarrow \mathsf{F}$. We argue

$$\begin{array}{ll}
 & (\alpha_0,\alpha_1) : \mathrm{Id} \hookleftarrow \mathsf{F}. \\
= & \quad \{ \quad \text{definitions} \quad \} \\
 & \forall(\mathsf{R},\mathsf{S} :: (\mathsf{R},\mathsf{S}) \cdot (\alpha_0,\alpha_1) \supseteq (\alpha_0,\alpha_1) \cdot (\mathsf{S},\mathsf{R})) \\
\Rightarrow & \quad \{ \quad \text{composition in } \mathcal{B}^2 \quad \} \\
 & \forall(\mathsf{R},\mathsf{S} :: \mathsf{R} \cdot \alpha_0 \supseteq \alpha_0 \cdot \mathsf{S}) \\
\Rightarrow & \quad \{ \quad \text{take } \mathsf{R} := \perp\!\!\perp \text{ and } \mathsf{S} := \mathsf{id} \quad \} \\
 & \perp\!\!\perp \supseteq \alpha_0
\end{array}$$

By symmetry, we also get $\alpha_1 = \perp\!\!\perp$, and therefore $(\perp\!\!\perp,\perp\!\!\perp)$ is the only transformation of type $\mathrm{Id} \hookleftarrow \mathsf{F}$. Since $(\perp\!\!\perp,\perp\!\!\perp)\backslash(\mathsf{R},\mathsf{S}) = (\top\!\!\top,\top\!\!\top)$ it follows that F does not have membership.

It is important to realize that F does not have a membership relation since we consider $\mathcal{C} = \mathcal{B}^2$ to be our base category and thus F to be an endorelator. If we consider $\mathcal{B}$ to be our base category, so F is a relator of arity $2 \leftarrow 2$ defined by $\langle \mathrm{Outr}, \mathrm{Outl} \rangle$, it follows from lemma's (4.19) and (4.25) that F does have a membership relation.

4.5 Fans

In section 4.1.1 we have proven that, for single-valued F, $\mathsf{mem.F}$ is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \Delta\mathsf{F}$. This fact can be generalized to any pair of single-valued relators that have membership:

Lemma 4.38 Let F and G be single-valued relators with membership $\mathsf{mem.F}$ and $\mathsf{mem.G}$ respectively. Then the largest natural transformation of type $\mathsf{F} \hookleftarrow \mathsf{G}$ is $\cap(\mathsf{mem.F}\backslash\mathsf{mem.G})$.

Proof From the definition of membership (4.16) it follows that

$$\cap(\mathsf{mem.F}\backslash\mathsf{mem.G}) = \mathsf{Fmem.G} \cdot \cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta\mathsf{G}}$$

Furthermore, $\mathsf{mem.G}$ has type $\mathrm{Id} \hookleftarrow \Delta\mathsf{G}$ and $\cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta}$ has type $\mathsf{F}\Delta \hookleftarrow \mathrm{Id}$ by equation (4.23). Hence, $\cap(\mathsf{mem.F}\backslash\mathsf{mem.G})$ has type $\mathsf{F} \hookleftarrow \mathsf{G}$. Now, let $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$, then

$$\begin{array}{ll}
 & \alpha \subseteq \cap(\mathsf{mem.F}\backslash\mathsf{mem.G}) \\
\equiv & \quad \{ \quad \Delta \leftrightarrow \cap \quad \} \\
 & \Delta\alpha \subseteq \mathsf{mem.F}\backslash\mathsf{mem.G} \\
\equiv & \quad \{ \quad \text{factors} \quad \}
\end{array}$$

$$
\begin{array}{ll}
 & \mathsf{mem.F} \cdot \Delta\alpha \subseteq \mathsf{mem.G} \\
\Leftarrow & \{ \quad \mathsf{mem.G} \text{ largest natural transformation of type } \mathrm{Id} \hookleftarrow \Delta\mathsf{G} \quad \} \\
 & \mathsf{mem.F} \cdot \Delta\alpha \, : \, \mathrm{Id} \hookleftarrow \Delta\mathsf{G} \\
\equiv & \{ \quad \mathsf{mem.F} \, : \, \mathrm{Id} \hookleftarrow \Delta\mathsf{F} \;,\; \Delta\alpha \, : \, \Delta\mathsf{F} \hookleftarrow \Delta\mathsf{G} \quad \} \\
 & \mathit{true}
\end{array}
$$

□

So, for endorelators F and G, we have that $\mathsf{mem.F}\backslash\mathsf{mem.G}$ is the largest natural transformation of type $\mathsf{F} \hookleftarrow \mathsf{G}$. It is illuminating to interpret this result in Rel. If $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$ then $\alpha \subseteq \mathsf{mem.F}\backslash\mathsf{mem.G}$. That is, $\mathsf{mem.F} \cdot \alpha \subseteq \mathsf{mem.G}$. This says that α can never invent new values: if $x \; \alpha \; y$, then every value stored in the F-structure x is a member of the set of values stored in the G-structure y.

An important special case occurs when G is the identity relator: for endorelator F, $\mathsf{mem.F}\backslash\mathsf{id}$ is the largest natural transformation of type $\mathsf{F} \hookleftarrow \mathsf{id}$. The interpretation of $\mathsf{mem.F}\backslash\mathsf{id}$ is a relation that holds between an F-structure and a value x such that all values stored in the F-structure are copies of the value x. Such a relation was called a *generator* in [4] (where it was first introduced) because it generates (or creates) F-structures from a given seed value x. We now prefer to use the term *fan* because "generator" is sometimes used with a different meaning and because "fan" is shorter. (The metaphor is a hand-held device for creating a current of air made from several copies of a single blade that can be fanned out or completely closed up.)

In this section we give a general definition of a fan (satisfied by $\mathsf{mem.F}\backslash\mathsf{id}$ in the case of an endorelator F) and then show that any relator with membership has a unique fan. The definition of fan is based on the following two observations: first, we have that $\mathsf{mem.F}\backslash\mathsf{id} : \mathsf{F} \hookleftarrow \mathsf{id}$ and second, because $\lambda R \,.\, (P\backslash R)$ preserves arbitrary intersections, it follows from the definition of membership that $\lambda R \,.\, (\mathsf{F}R \cdot \mathsf{mem.F}\backslash\mathsf{id})$ preserves arbitrary intersections. For the definition of a fan, we only require preservation of *finite* intersections. This leads to the following definition.

Definition 4.39 A *fan* of endorelator F is a natural transformation $\mathsf{fan} : \mathsf{F} \hookleftarrow \mathrm{Id}$ such that the mapping $\lambda(R : B \leftarrow A).(\mathsf{F}R \cdot \mathsf{fan}_A)$ preserves finite intersections:

$$
\begin{array}{rcl}
\mathsf{F}\top\!\!\top_{B,A} \cdot \mathsf{fan}_A & = & \top\!\!\top_{\mathsf{F}B,A} \\
\mathsf{F}(R \cap S) \cdot \mathsf{fan}_A & = & \mathsf{F}R \cdot \mathsf{fan}_A \;\cap\; \mathsf{F}S \cdot \mathsf{fan}_A \quad \text{for all } R\,,S : B \leftarrow A.
\end{array}
$$

□

Corollary 4.40 $\mathsf{mem.F}\backslash\mathsf{id}$ is a fan of endorelator F.

□

We start with a useful lemma:

Lemma 4.41 If for endorelator F and a collection of arrows α, $\alpha_A : FA \leftarrow A$ the mapping $\lambda(X : B \leftarrow A) . (FX \cdot \alpha_A)$ preserves binary intersections then

$$FR \cap FS \cdot \alpha \cdot \alpha^\circ \cdot FT \subseteq F(R \cap S \cdot T)$$

Proof Since id is total it follows from equation (2.53) that for each $R : A \leftarrow B$,

$$\mathsf{outl}_{A,B} \cdot R \vartriangle \mathsf{id}_B = R$$

Note that $\mathsf{outl}_{A,B}$ is simple, and $R \vartriangle \mathsf{id}_B$ is cosimple since $R \vartriangle \mathsf{id}_B \subseteq \mathsf{outr}^\circ_{A,B}$. So, although we do not assume tabularity, it follows that if we assume products then for every relation R there exists a pair of *simple* relations (f, g) such that $R = f \cdot g^\circ$. Now, using that $R = f \cdot g^\circ$ for some simple f and g, we calculate:

$$\begin{array}{cl}
 & F(f \cdot g^\circ) \;\cap\; FS \cdot \alpha \cdot \alpha^\circ \cdot FT \\
= & \{ \quad \text{modular identity, } f \text{ and } g \text{ simple} \quad \} \\
 & Ff \;\cdot\; (\mathsf{id} \;\cap\; F(f^\circ \cdot S) \cdot \alpha \cdot \alpha^\circ \cdot F(T \cdot g)) \;\cdot\; Fg^\circ \\
= & \{ \quad \text{domains: } \mathsf{id} \cap P \cdot Q^\circ = (P \cap Q)< \quad \} \\
 & Ff \;\cdot\; (F(f^\circ \cdot S) \cdot \alpha \;\cap\; F(T \cdot g)^\circ \cdot \alpha)< \;\cdot\; Fg^\circ \\
= & \{ \quad \lambda R . (FR \cdot \alpha) \text{ preserves binary intersections} \quad \} \\
 & Ff \;\cdot\; (F(f^\circ \cdot S \;\cap\; (T \cdot g)^\circ) \;\cdot\; \alpha)< \;\cdot\; Fg^\circ \\
\subseteq & \{ \quad \text{domains: } (P \cdot Q)< \subseteq P<,\ (FP)< = F(P<) \quad \} \\
 & Ff \;\cdot\; F((f^\circ \cdot S \;\cap\; (T \cdot g)^\circ)<) \;\cdot\; Fg^\circ \\
= & \{ \quad \text{domains: } (P \cap Q)< \;=\; \mathsf{id} \;\cap\; P \cdot Q^\circ \quad \} \\
 & Ff \;\cdot\; F(\mathsf{id} \;\cap\; f^\circ \cdot S \cdot T \cdot g) \;\cdot\; Fg^\circ \\
= & \{ \quad \text{modular identity, } f \text{ and } g \text{ simple} \quad \} \\
 & F(f \cdot g^\circ \;\cap\; S \cdot T)
\end{array}$$

□

As a direct corollary, we have for fans:

Corollary 4.42 Let fan be a fan of endorelator F, then

$$F\top\!\!\top_{A,A} \;\cap\; \mathsf{fan}_A \cdot \mathsf{fan}^\circ_A \;\subseteq\; \mathsf{id}_{FA}$$

Proof Follows from (4.41) by taking $R = \top\!\!\top_{A,A}$, $S = \mathsf{id}_A$ and $T = \mathsf{id}_A$.
□

The interpretation of corollary (4.42) is that if we have two F-structures x and y with the same shape, i.e. $x\, F\top\!\!\top\, y$, and x and y are filled with the same value a, i.e. $x\, \mathsf{fan}\, a$ and $y\, \mathsf{fan}\, a$ then x and y are equal. In other words, for every value a there exists for every F-shape at most one F-structure of that shape containing only copies of a.

Not only is $\mathsf{mem}.F \backslash \mathsf{id}$ a fan of relator F, it is also the unique one. Corollary (4.42) is the key-argument in the proof of this fact.

Lemma 4.43 If endorelator F has membership $\mathsf{mem.F}$, then $\mathsf{mem.F}\backslash\mathsf{id}$ is the unique fan of F.

Proof For uniqueness we argue as follows: suppose α and β are both fans of F such that $\alpha \subseteq \beta$, then

$$\begin{array}{ll}
 & \beta \\
= & \{ \quad \alpha \text{ fan of } \mathsf{F}\text{: } \top\!\top = \mathsf{F}\top\!\top \cdot \alpha\text{, intersection} \quad \} \\
 & \beta \cap \mathsf{F}\top\!\top \cdot \alpha \\
\subseteq & \{ \quad \text{modular law} \quad \} \\
 & (\beta \cdot \alpha^{\circ} \cap \mathsf{F}\top\!\top) \cdot \alpha \\
\subseteq & \{ \quad \alpha \subseteq \beta \quad \} \\
 & (\beta \cdot \beta^{\circ} \cap \mathsf{F}\top\!\top) \cdot \alpha \\
\subseteq & \{ \quad \beta \text{ fan of } \mathsf{F}\text{, lemma (4.42)} \quad \} \\
 & \alpha
\end{array}$$

Hence, $\alpha = \beta$. So if there are two fans then they are incomparable via $\subseteq$. But from theorem (4.38) it follows that $\mathsf{mem.F}\backslash\mathsf{id}$ is the largest transformation of type $\mathsf{F} \hookleftarrow \mathrm{Id}$, hence, $\mathsf{mem.F}\backslash\mathsf{id}$ is the unique fan of relator F.
□

We call $\mathsf{mem.F}\backslash\mathsf{id}$ the *canonical* fan of relator F.

4.5.1 Fans of non-endo relators

Until now we have only considered fans of endorelators. In this section we generalize the definition of a fan to arbitrary relators. We start with single-valued relators. For endorelator F, $\mathsf{mem.F}\backslash\mathsf{id}$ is its canonical fan. For a single-valued relator $\mathsf{F} : 1 \leftarrow \mathsf{k}$ we want $\cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta_k}$ to be its canonical fan. Equation (4.23) states that $\cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta_k}$ is a natural transformation of type $\mathsf{F}\Delta_k \hookleftarrow \mathrm{Id}$. Furthermore, we have that the (partial) mapping

$$\lambda R \;:\; B \leftarrow \Delta_k A \;.\; (\mathsf{F}R \cdot \cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta_k A})$$

preserve finite intersections since by definition (4.15),

$$\mathsf{F}R \cdot \cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta_k A} = \cap(\mathsf{mem.F}\backslash R)_B$$

and $\lambda R \;:\; B \leftarrow \Delta_k A \;.\; \cap(\mathsf{mem.F}\backslash R)_B$ preserves finite intersections. This leads to the following definition:

Definition 4.44 A *fan* of single-valued relator $\mathsf{F} : 1 \leftarrow \mathsf{k}$ is a natural transformation $\mathsf{fan} \;:\; \mathsf{F}\Delta_k \hookleftarrow \mathrm{Id}$ such that the mapping $\lambda(R \;:\; B \leftarrow \Delta_k A).(\mathsf{F}R \cdot \mathsf{fan}_A)$ preserves finite intersections:

$$\begin{array}{rcll}
\mathsf{F}\top\!\top_{B,\Delta A} \cdot \mathsf{fan}_A & = & \top\!\top_{\mathsf{F}B,A} & \\
\mathsf{F}(R \cap S) \cdot \mathsf{fan}_A & = & \mathsf{F}R \cdot \mathsf{fan}_A \;\cap\; \mathsf{F}S \cdot \mathsf{fan}_A & \text{for all } R\,, S \;:\; B \leftarrow \Delta_k A.
\end{array}$$

□

Corollary 4.45 $\cap(\mathsf{mem.F}\backslash\mathsf{id})_{\Delta_k}$ is a fan of endorelator F.
□

Next we define the fan for arbitrary relators F of arity $m \leftarrow k$. Just as for membership, we define a fan of a tuple of relators as a tuple of the corresponding fans:

Definition 4.46 For relator F of arity $l \leftarrow k$, the fan, $\mathsf{fan.F} : l \leftarrow 1$, is defined by

$$\mathsf{fan.F} \;\triangleq\; \Delta_l \mathsf{fan.F}_l \;\;.$$

□

Note that fan.F is a natural transformation of type $\mathsf{F}\Delta_k \hookleftarrow \Delta_l$ since

$$\Delta_l \mathsf{fan.F}_l \;\;:\;\; \Delta_l \mathsf{F}_l \Delta_k \hookleftarrow \Delta_l \;=\; \mathsf{F}\Delta_k \hookleftarrow \Delta_l \;\;.$$

4.5.2 Uniqueness of fan (non-endo) relator

Just as for endorelators it is the case that the existence of a membership relation implies that the canonical fan of a non-endo relator is the largest natural transformation of its type. We consider single-valued relators first. From lemma (4.29) and $\mathsf{mem}.\Delta = \Delta(\mathsf{id})$ it follows that $\cap(\mathsf{mem.F}\backslash\mathsf{id})_\Delta = \mathsf{mem.F}\Delta\backslash\mathsf{id}$. So, the canonical fan of a single-valued relator F is the canonical fan of the endo (!) relator $\mathsf{F}\Delta$. Hence, from lemma (4.43) it follows that $\cap(\mathsf{mem.F}\backslash\mathsf{id})_\Delta$ is the unique fan of relator F. The canonical fan of an arbitrary relator F of arity $l \leftarrow k$ is defined as $\Delta_l \mathsf{fan.F}_l$. Each of the components $\mathsf{fan.F}_l$ is the largest natural transformation of its type, hence, $\Delta_l \mathsf{fan.F}_l$ is the largest natural transformation of type $\mathsf{F}\Delta_k \hookleftarrow \Delta_l$.

4.5.3 Fans of regular relators

The definition of a fan as presented here arose more or less as a counterpart of membership, i.e. a membership has type $\mathsf{Id} \hookleftarrow \mathsf{F}$ whereas a fan has type $\mathsf{F} \hookleftarrow \mathsf{Id}$. Directly connected with fans we have the notion of the so-called *fan-function*. The fan-function $\hat{\mathsf{F}}$ of relator $\mathsf{F} : l \leftarrow k$ is defined by

$$\hat{\mathsf{F}}\mathsf{R} = \mathsf{FR} \cdot (\mathsf{fan.F})_\mathsf{A} \quad \text{for all } \mathsf{R} \;:\; \mathsf{B} \leftarrow \Delta_k \mathsf{A}$$

Note that the fan-function is a partial function: it is only defined on a vector of relations with the same source. We call the fan-function *canonical* if it is constructed using the canonical fan.

Note that for an arbitrary relator we have

$$\hat{\mathsf{F}} = \Delta_l \widehat{\mathsf{F}_l} \;\;,$$

since

$$\mathsf{FR} \cdot (\mathsf{fan.F})_\mathsf{A} = \Delta_l \mathsf{F}_l \mathsf{R} \cdot \Delta_l (\mathsf{fan.F}_l)_\mathsf{A} = \Delta_l (\mathsf{F}_l \mathsf{R} \cdot (\mathsf{fan.F}_l)_\mathsf{A}) \;\;.$$

For this reason we may restrict our attention to single-valued relators without loss of generality.

Having the notion of a (canonical) fan-function, the definition of membership (for a single-valued relator) can be expressed as the requirement that

$$\hat{F}R = \cap((\mathsf{mem}.F)_B \backslash R) \quad \text{for all } R : B \leftarrow \Delta_k A$$

since

$$\hat{F}R = FR \cdot (\mathsf{fan}.F)_A = FR \cdot (\mathsf{mem}.F\Delta \backslash \mathsf{id})_A = FR \cdot \cap(\mathsf{mem}.F \backslash \mathsf{id})_{\Delta A}$$

The definition of membership, which is stated as a fusion property, is for product and coproduct the fusion property with split and cojunc. So, membership may be a new concept for relators whereas the fan-function is already an existing notion for product and coproduct.

The fan-functions of the regular relators can be constructed inductively.

Lemma 4.47 The canonical fan-functions of the regular relators are given by the following clauses:

$$\begin{array}{rcl}
\widehat{\mathrm{Id}} & = & \mathrm{Id} \\
\widehat{\mathrm{Proj}} & = & \mathrm{Proj} \\
\widehat{K_A} & = & \lambda(R : C \leftarrow B) \,.\, \top\!\top_{A,B} \\
\hat{+} & = & \blacktriangledown \\
\hat{\times} & = & \vartriangle \\
\widehat{FG} & = & \hat{F}\hat{G} \\
\hat{T} & = & \lambda R : A \leftarrow B \;.\; [\![\mathsf{id}_A \otimes ;\; R \hat{\otimes} \mathsf{id}_B]\!]
\end{array}$$

where "$\blacktriangledown$" denotes the cojunc operator defined by $R \blacktriangledown S = (R^\circ \triangledown S^\circ)^\circ$ and T is the tree type induced by binary relator $\otimes$.

Proof The verification of the case K_A is straightforward. The verification of the case Proj — which includes Id — follows from lemma (4.25) where we have proved that

$$\cap(\mathsf{mem}.\mathrm{Proj} \backslash R) = \mathrm{Proj} R$$

The verification of the definition of the fan-function of "$+$" and "$\times$" is a direct corollary of properties (2.47) and (2.64). Verification of the case FG follows from lemma (4.29). And finally, the verification of the fan-function of tree type T is a directly corollary of lemma (4.36). □

Since we have the identity $\hat{F}\mathsf{id}_{\Delta_k A} = (\mathsf{fan}.F)_A$, we have as a corollary,

Corollary 4.48 The (canonical) fan of a regular relator is given by the following clauses:

$$\begin{array}{rcl}
\mathsf{fan}.\mathrm{Id} & = & \mathsf{id} \\
\mathsf{fan}.\mathrm{Proj} & = & \mathsf{id} \\
\mathsf{fan}.K_A & = & \top\!\top_{A,_}
\end{array}$$

$$\begin{aligned}
\mathsf{fan}.+ &= \mathsf{id} \blacktriangledown \mathsf{id}\\
\mathsf{fan}.\times &= \mathsf{id} \vartriangle \mathsf{id}\\
\mathsf{fan}.\mathsf{FG} &= \mathsf{F}(\mathsf{fan}.\mathsf{G}) \cdot \mathsf{fan}.\mathsf{F}\\
\mathsf{fan}.\mathsf{T} &= [\![\mathsf{id}\otimes;\ \mathsf{fan}.\otimes]\!]
\end{aligned}$$

□

Just as for cojunc, cosplit and cocatamorphism, we define the notion of a *cofan-function*, denoted by $\check{\mathsf{F}}$, as the converse-conjugate of the fan-function. That is, we define $\check{\mathsf{F}}$ by

$\check{\mathsf{F}}R = (\hat{\mathsf{F}}(R^\circ))^\circ$ for each $R : \Delta_k B \leftarrow A$.

4.5.4 Recoverable relators

In this section we show that coproduct, product and the projection relators can be recovered from their fan-functions with their membership relations. This property we later exploit in Chapter 5 and it enables us to use the slok-theory as introduced in Chapter 3.

We have the properties,

$$\mathsf{inl}^\circ_{A,B} \blacktriangledown \mathsf{inr}^\circ_{A,B} = \mathsf{id}_A + \mathsf{id}_B \quad ,$$

$$\mathsf{outl}_{A,B} \vartriangle \mathsf{outr}_{A,B} = \mathsf{id}_A \times \mathsf{id}_B \quad ,$$

and

$$\mathsf{Proj}_i\Delta_k(i = k \rightarrow \mathsf{id}_{\mathsf{Proj}_i}\, ,\, \perp\!\!\perp_{\mathsf{Proj}_k\,,\,\mathsf{Proj}_i}) = \mathsf{id}_{\mathsf{Proj}_i} \quad .$$

All three properties are easily to verify. Hence, in the case of coproduct, product and projection relators, the relator can be recovered from its fan-function with its membership relation.

Chapter 5

A Class of Commuting Relators

5.1 Introduction

The zip function is well known to functional programmers [9]. Informally, zip maps two lists of the same length into a single list of pairs whereby

$$([a_1\,,a_2\,,\ldots]\,,[b_1\,,b_2\,,\ldots]) \mapsto [(a_1\,,b_1)\,,(a_2\,,b_2)\,,\ldots]$$

Zip is just one of a whole family of operations that *commute* the order of two data structures. Whilst zip commutes a pair of lists into a list of pairs, it is not difficult to imagine generalising zip to a function that commutes m lists each of length n into n lists each of length m. Indeed, this latter function is also well known under the name *matrix transposition*. With a little bit more imagination several other members of the family suggest themselves. For example, there is a function that commutes a tree of lists all of the same length into a list of trees all of the same shape. There is also a function that "broadcasts" a value to all elements of a list —thus

$$(a\,,[b_1\,,b_2\,,\ldots]) \mapsto [(a\,,b_1)\,,(a\,,b_2)\,,\ldots]$$

— . That is, the datatype an element of type A paired with (a list of elements of type B) is "commuted" to a list of (element of type A paired with an element of type B). This list broadcast is itself an instance of a subfamily of the operations that we discuss in this chapter. In general, a broadcast operation copies a given value to all locations in a given data structure.

The construction of individual members of this class of operations (for example the zip function on pairs of lists or matrix transposition) is a straightforward programming exercise. Constructing, in one go, a *class* of such functions, having an infinite number of elements, is much more exacting, and demands a good theory of datatypes that facilitates the formulation of a compact and workable generic specification of the members of the class of functions.

In this chapter we consider the construction of such functions. We specify the class of so-called "zippable" data-structures and consider in detail a large and significant subclass of this class. For this subclass we show how to construct "zips" and identify several consequences of their being "zippable".

A substantial portion of the chapter is devoted simply to formulating requirements for what it is for something to be a generic "zip" of two datatypes. (We use the terminology "requirement" because, at this stage in our work, we are unable to prove that our "requirements" completely characterise zips. That is, we have no generally valid uniqueness property of solutions to our requirements.) The techniques we use are, to the best of our knowledge, original and, so, perhaps not easy to grasp on a first encounter. To ease the understanding we build up to the requirement for the most general case in successive stages. We also present several non-trivial consequences of the requirement giving greater insight into their nature.

The simplification we make in the first instance is that all datatypes are endorelators. That is, we do not consider datatypes having more than one type parameter. This initial formulation is progressively refined to the case that the two datatypes to be "zipped" have arbitrary, not necessarily matching, arities. Another simplification that we make is to base our requirements on the notion of a "half-zip". The latter simplification is of a different nature to the former, it being a generalisation of the notion of a zip rather than a specialisation.

Having constructed our requirement we then proceed to show that it is met by the regular datatypes.

5.2 The requirement (endorelators)

The first problem we have to tackle is that of formulating a specification of the family of "zip" functions. As stated before, we begin by restricting our attention to endorelators like list and tree. Of course, this is inadequate for our ultimate goal —even the standard example of zipping a pair of lists is excluded by this restriction since product (pairing) is not endo— but we shall see that our requirements in their final form reduce directly to the form given in this subsection if all information on the arity of the relators is hidden from view.

Looking again at the examples above, the first step towards an abstract problem specification is clear enough. Replacing "list", "tree" etc. by "relator F" the problem is to construct an operation $\mathsf{zip.F.G}$ for given relators F and G that maps F-G-structures into G-F-structures.

The first step may be clear enough, subsequent steps are less clear. One complication is that "zips" are typically partial, being defined only on F–structures of (G–structures *of the same shape*) and not on all F-G-structures. Because of the level of generality that we are seeking it is not immediately evident how to formulate a precise specification.

Rather than falling into the trap of basing our specification on operational considerations — such as the effect on the shape of the given structures— we base our specification on abstract, algebraic properties.

The nature of our requirements is influenced by Reynolds' insights into the relationship between parametric polymorphism and naturality properties [43]. Reynolds' idea as popularised by Wadler [52] is that if a function is parametrically polymorphic in a type then it is possible to derive from its type a property of the function (a "theorem for free" as Wadler called it). For example, any parametrically polymorphic function α such that α_{A} has type $\mathit{List}\mathsf{A} \leftarrow \mathit{Tree}\mathsf{A}$ for all A must satisfy the naturality property $\mathit{List}\mathsf{R} \cdot \alpha_{\mathsf{B}} \supseteq \alpha_{\mathsf{A}} \cdot \mathit{Tree}\mathsf{R}$ for all relations R of type $\mathsf{A} \leftarrow \mathsf{B}$. This, according to Reynolds, expresses the requirement that α_{A} should be independent of the representation of elements A, or that there is a "logical relationship" between the different instances of α.

Our exploitation of Reynolds' insight takes place at a higher level. We consider the relator F to be fixed and specify a collection of operations $\mathsf{zip.F.G}$ indexed by the relator G. (The fact that the index is a relator rather than a type is what we mean by "at a higher level".) Such a family forms what we call a collection of "half-zips". The requirement is that the collection be "parametric" in G. That is, the elements of the family $\mathsf{zip.F}$ should be "logically related" to each other.

The precise formulation of this idea leads us to three requirements on "half-zips". The symmetry between F and G, lost in the process of fixing F and varying G, is then restored by the simple requirement that a zip is both a half-zip and the converse of a half-zip.

The division of our requirements into "half-zips" and "zips" corresponds to the way that we construct zips later in this chapter. We construct $\mathsf{zip.F.G}$ for each relator F in the class of the regular relators and an arbitrary relator G. (This construction is inductive over the structure of the regular relators.) That is to say, for each regular relator F we construct the mapping $\mathsf{zip.F}$ on relators, which for an arbitrary relator G gives the corresponding zip operation $\mathsf{zip.F.G}$. The mapping is constructed to meet the requirement that it define a collection of half-zips; subsequently we show that if the collection is restricted to regular relators G then each half-zip is in fact a zip.

5.2.1 Naturality requirements

Our first requirement is that $\mathsf{zip.F.G}$ be natural. That is to say, its application to an F-G-structure should not in any way depend on the values in that structure. So, we demand

$$\mathsf{zip.F.G} \;:\; \mathsf{GF} \leftarrow \mathsf{FG} \tag{5.1}$$

Note that we have chosen $\mathsf{zip.F.G}$ to be a natural transformation with *equality* since for a zip operation on a structure no loss or duplication of values should occur.

Demanding naturality is not enough. Somehow we want to express that the members of the family $\mathsf{zip.F}$ of zip operations for different datatypes G and H are related. For instance, if we have a natural transformation $\alpha : \mathsf{G} \leftarrow \mathsf{H}$ then $\mathsf{zip.F.G}$ and $\mathsf{zip.F.H}$ should be coherent with the transformation α. That is to say, having both zips and α, there are two ways of transforming F-H-structures into G-F-structures; these should effectively be the same.

One way is first transforming an F-H-structure into an F-G-structure using $\mathsf{F}\alpha$, (i.e. applying the transformation α to each H-structure inside the F-structure) and then zipping the F-G-structure into a G-F-structure using $\mathsf{zip.F.G}$.

Another way is first zipping an F-H-structure into a H-F-structure with $\mathsf{zip.F.H}$ and then transforming this H-structure into a G-structure (both containing F-structures) using α_{F}. So, we have the following diagram.

$$\begin{tikzcd}
\mathsf{FGA} \arrow[d, "\mathsf{zip.F.G_A}"'] & \mathsf{FHA} \arrow[l, "\mathsf{F}\alpha_{\mathsf{A}}"'] \arrow[d, "\mathsf{zip.F.H_A}"] \\
\mathsf{GFA} & \mathsf{HFA} \arrow[l, "\alpha_{\mathsf{FA}}"]
\end{tikzcd}$$

One might suppose that an equality is required, i.e.

$$\alpha_F \cdot \mathsf{zip}.F.H \quad = \quad \mathsf{zip}.F.G \cdot F\alpha \tag{5.2}$$

for all natural transformations $\alpha : G \leftarrow H$. But this requirement is too severe for two reasons.

The first reason is that if α is not functional, i.e. a non-deterministic transformation, the rhs of equation (5.2) may be more non-deterministic than the lhs because of the possible multiple occurrences of α. Take for instance $F := \mathit{List}$ and $G = H := \times$, i.e. $\mathsf{zip}.F.G$ and $\mathsf{zip}.F.H$ are both the inverse of the zip function on a pair of lists, and take $\alpha := \mathsf{id} \cup \mathsf{swap}$, i.e. α non-deterministically swaps the elements of a pair or not. Then $\alpha_F \cdot \mathsf{zip}.F.H$ unzips a list of pairs into a pair of lists and swaps the lists or not. On the other hand, $\mathsf{zip}.F.G \cdot F\alpha$ first swaps some of the elements of a list of pairs and then unzips it into a pair of lists.

The second reason is that, due to the partiality of zips, the domain of the left side of (5.2) may be smaller than that of the right.

As a concrete example, suppose listify is a polymorphic function that constructs a list of the elements stored in a tree. The way that the tree is traversed (inorder, preorder etc.) is immaterial; what is important is that listify is a natural transformation of type $\mathit{List} \leftarrow \mathit{Tree}$. Now suppose we are given a list of trees. Then it can be transformed to a list of lists by “listify”ing each tree in the list, i.e. by applying the (appropriate instance of the) function $\mathit{List}(\mathsf{listify})$. If all the trees in the list have the same shape, a list of lists can also be obtained by first “zipping” the list of trees to a tree of lists (all of the same length) and then “listify”ing the tree structure. That is we apply the (appropriate instance of the) function $\mathsf{listify}_{\mathit{List}} \cdot \mathsf{zip}.\mathit{List}.\mathit{Tree}$. The two lists of lists will not be the same: if the size of the original list is m and the size of each tree in the list is n then the first method will construct m lists each of length n whilst the second method will construct n lists each of length m. However the two lists of lists are “zips” of each other (“transposes” would be the more conventional terminology). This is expressed by the commutativity of the following diagram in the case that the input type $\mathit{List}(\mathit{Tree}A)$ is restricted to lists of trees of the same shape.

$$\begin{tikzcd}
\mathit{List}(\mathit{List}A) \arrow[d, "(\mathsf{zip}.\mathit{List}.\mathit{List})_A"'] & \mathit{List}(\mathit{Tree}A) \arrow[l, "\mathit{List}(\mathsf{listify})_A"'] \arrow[d, "(\mathsf{zip}.\mathit{List}.\mathit{Tree})_A"] \\
\mathit{List}(\mathit{List}A) & \mathit{Tree}(\mathit{List}A) \arrow[l, "\mathsf{listify}_{\mathit{List}A}"]
\end{tikzcd}$$

Note however that if we view both paths through the diagram as partial relations of type $\mathit{List}(\mathit{List}A) \leftarrow \mathit{List}(\mathit{Tree}A)$ then the upper path (via $\mathit{List}(\mathit{List}A)$) includes the lower path (via $\mathit{Tree}(\mathit{List}A)$). This is because the function $\mathit{List}(\mathsf{listify})_A$ may construct a list of lists all of the same length (as required by the subsequent zip operation) even though all the trees in the given list of trees may not all have the same shape. The requirement on the trees is that they all have the same size, which is weaker than their all having the same shape.

Both examples show that we have to relax requirement (5.2) using an inclusion instead of equality. Having this inclusion, the requirement for α can be relaxed as well. So, the requirement becomes

$$\alpha_F \cdot \mathsf{zip}.F.H \subseteq \mathsf{zip}.F.G \cdot F\alpha \quad \text{for all } \alpha : G \hookleftarrow H \tag{5.3}$$

5.2.2 Monoid homomorphism

For our next requirement we consider the monoid structure of functors under composition. Fix functor F and consider the collection of zips, $\mathsf{zip}.F.G$, indexed by (endo)functor G. Since the (endo)functors form a monoid it is required that the mapping $\mathsf{zip}.F$ is a monoid homomorphism.

In order to formulate this requirement precisely we let ourselves be driven by type considerations. The requirement is that $\mathsf{zip}.F.GH$ be some composition of $\mathsf{zip}.F.G$ and $\mathsf{zip}.F.H$ of which $\mathsf{zip}.F.\mathrm{Id}$ is the identity. But the type of $\mathsf{zip}.F.GH$,

$$\mathsf{zip}.F.GH \;:\; GHF \leftarrow FGH \;\;,$$

demands that the relator F has to be "pushed" through GH leaving the order of G and H unchanged. With $\mathsf{zip}.F.G$ we can swap the order of F and G, with $\mathsf{zip}.F.H$ the order of F and H. Thus transforming FGH to GHF can be achieved as shown below.

$$GHF \xleftarrow{G(\mathsf{zip}.F.H)} GFH \xleftarrow{(\mathsf{zip}.F.G)_H} FGH$$

So, we demand

$$\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_H$$

Formally, define (for fixed relator F) a monoid $\mathcal{M}$ as follows. The elements are pairs consisting of a natural transformation, α , and a functor, G, where

$$(\alpha, G) \in \mathcal{M} \quad \equiv \quad \alpha : GF \xleftarrow{\mathsf{Fun}} FG$$

Define composition in the following way:

$$(\alpha, G)\circ(\beta, H) \triangleq (G\beta \cdot \alpha_H \;,\; GH)$$

Note that $(G\beta \cdot \alpha_H \;, GH)$ is an element of $\mathcal{C}$ since $G\beta \cdot \alpha_H : GHF \leftarrow FGH$ follows from $\beta : HF \leftarrow FH$ and $\alpha : GF \leftarrow FG$. It is clear that "$\circ$" has unit $(\mathrm{id}_F, \mathrm{Id})$ and is associative, i.e.

$$((\alpha, G)\circ(\beta, H))\circ(\gamma, I) = (\alpha, G)\circ((\beta, H)\circ(\gamma, I))$$

since $GH\gamma \cdot (G\beta \cdot \alpha_H)_I = G(H\gamma \cdot \beta_I) \cdot \alpha_{HI}$. Thus $\mathcal{M}$ is indeed a monoid.

Now, define $f(G) = (\mathsf{zip}.F.G\,, G)$. Then the mapping $\mathsf{zip}.F$ is a monoid homomorphism if $f(GH) = f(G)\circ f(H)$ and $f(\mathrm{Id}) = (\mathrm{id}_F\,, \mathrm{Id})$. Expanding the definition of f, we thus demand

$$\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_H \tag{5.4}$$

and

$$\mathsf{zip}.F.\mathrm{Id} = \mathrm{id}_F \tag{5.5}$$

(Note that $\mathrm{id}_F : \mathrm{Id}F \leftarrow F\mathrm{Id}$.)

5.2.3 Half zips and commuting relators

Apart from the very first of our requirements ((5.1), the requirement that $\mathsf{zip.F.G}$ be natural), all the other requirements have been requirements on the nature of the mapping $\mathsf{zip.F}$. Roughly speaking, (5.3) demands that it be parametric, and (5.4) and (5.5) that it be functorial. We find it useful to bundle these requirements together into the definition of something that we call a "half-zip".

Definition 5.6 (Half Zip (Endorelators)) Consider a fixed relator F and a class of relators $\mathcal{G}$ that includes at least the identity relator and is closed under composition. Then the members of the collection $\mathsf{zip.F.G}$, where G ranges over $\mathcal{G}$, are called *half-zips* iff,

(a) $\mathsf{zip.F.G} \ : \ \mathsf{GF} \leftarrow \mathsf{FG}$,

(b) $\alpha_{\mathsf{F}} \cdot \mathsf{zip.F.H} \subseteq \mathsf{zip.F.G} \cdot \mathsf{F}\alpha$ for each $\alpha : \mathsf{G} \hookleftarrow \mathsf{H}$,

(c) $\mathsf{zip.F.GH} = \mathsf{G}(\mathsf{zip.F.H}) \cdot (\mathsf{zip.F.G})_{\mathsf{H}}$,

(d) $\mathsf{zip.F.Id} = \mathsf{id}_{\mathsf{F}}$.

□

Given the notion of a half-zip, we can recover the natural symmetry between the parameters F and G by defining a zip to be something that is such that both it and its converse are half-zips.

Definition 5.7 (Commuting Relators (Endo)) The half-zip $\mathsf{zip.F.G}$ is said to be a *zip* of (F, G) if there exist collections of half-zips, $\mathsf{zip.F}$ and $\mathsf{zip.G}$, such that

$$\mathsf{zip.F.G} = (\mathsf{zip.G.F})^{\circ}$$

We say that relators F and G *commute* if there exists a zip for (F, G).
□

This completes our list of requirements for endorelators.

5.3 Analysis of the requirement

Before generalising the requirements on zips to non-endo relators we interpose an analysis of the requirements. At first sight there is little connection between the formal and informal requirements given in the introduction to this chapter. For instance, the formal requirement makes no explicit mention of any shape preservation properties. The purpose of this section is to show that such considerations are indeed captured by the formal requirement.

The discussion begins at an abstract level and then progresses to concrete examples of the properties of zips. We begin with a more detailed analysis of the parametricity requirement (subsection 5.3.1) , and then discuss how the requirements capture the shape preservation of zips (subsection 5.3.2). This is followed by a technical discussion of some of the consequences of relators being commuting (subsection 5.3.3) in preparation for two concrete examples (sections 5.3.4 and 5.3.5). The message in these last two sections is that zips crop up in many unexpected places; indeed they are everywhere!

5.3.1 Higher-order parametricity

In section 5.2 we derived part of the definition of a half-zip from the requirement that $\mathsf{zip.F.G}$ and $\mathsf{zip.F.H}$ should be coherent with respect to natural transformations of type $\mathsf{G} \leftarrow \mathsf{H}$. We referred to Reynolds' work on parametric polymorphism and logical relations as the inspiration for the requirement but claimed that our use of the notion was at a higher level. In this section we present requirement (5.3) in a more abstract way in order to provide support for our claim. In particular, we show that (5.3) follows from the requirement that the construction of the class of zip operations $\mathsf{zip.F.G}$ should be natural (parametric) in G.

In order to see in what sense $\mathsf{zip.F.G}$ is natural in G let us denote it according to our convention for denoting natural transformations. Specifically, define $\beta_\mathsf{G} = \mathsf{zip.F.G}$. Thus β is a collection of arrows indexed by a relator. Now observe that the naturality property of β_G (property (5.1)) takes the form

$$\beta_\mathsf{G} \; : \; (\circ\mathsf{F})\mathsf{G} \leftarrow (\mathsf{F}\circ)\mathsf{G} \;\; . \tag{5.8}$$

Recall that $(\circ\mathsf{F})$ and $(\mathsf{F}\circ)$ denote pre- and post-composition with relator F. That is, $(\circ\mathsf{F})\mathsf{G} = \mathsf{GF}$ and $(\mathsf{F}\circ)\mathsf{G} = \mathsf{FG}$.

Property (5.8) states that each individual element β_G of the collection of arrows β is a natural transformation. If the collection of arrows is *itself* a natural transformation then the mappings $(\circ\mathsf{F})$ and $(\mathsf{F}\circ)$ are the object maps of the functors[1] between which β is a natural transformation. In subsection 2.2.3, we showed that $(\circ\mathsf{F})$ and $(\mathsf{F}\circ)$ are functors on natural transformations. The arrows maps of these functors are $(\circ\mathsf{F})\alpha$ $(= \alpha_\mathsf{F})$ and $(\mathsf{F}\circ)\alpha$ $(= \mathsf{F}\alpha)$, respectively. So, from $\beta_\mathsf{G} \; : \; (\circ\mathsf{F})\mathsf{G} \leftarrow (\mathsf{F}\circ)\mathsf{G}$ we derive the typing of β as a (higher order) natural transformation:

$$\beta \; : \; (\circ\mathsf{F}) \leftarrow (\mathsf{F}\circ)$$

Expanding the definition of a natural transformation gives us:

$$(\circ\mathsf{F})\alpha \cdot \beta_\mathsf{H} \; = \; \beta_\mathsf{G} \cdot (\mathsf{F}\circ)\alpha \quad \text{for each } \alpha : \mathsf{G} \leftarrow \mathsf{H}$$

Rewriting this and expanding $\beta = \mathsf{zip.F}$ gives:

$$\alpha_\mathsf{F} \cdot \mathsf{zip.F.H} \; = \; \mathsf{zip.F.G} \cdot \mathsf{F}\alpha \quad \text{for each } \alpha : \mathsf{G} \leftarrow \mathsf{H} \tag{5.9}$$

Equation (5.9), motivated here by type considerations, is not precisely the requirement on half-zips derived in section 5.2. As explained there, the conventional categorical notation of naturality is inadequate. In an allegory there are three kinds of natural transformation and care needs to be taken in deciding which is relevant. However, if all components involved are functions (5.9) is satisfied since inclusion of functions is equality of functions. The restriction to functions is the framework in which Reynolds' ideas are usually formulated and thus (a specialisation of) our requirement does express the fact that $\mathsf{zip.F.G}$ is polymorphic in the (higher order) parameter G.

[1] That is to say, functors on the functor category.

5.3.2 Shape preservation

Zips are partial operations: $\mathsf{zip.F.G}$ should map F-structures of (G-structures of the same shape) into G-structures of (F-structures of the same shape). This requirement is, however, not explicitly stated in our formalisation of being a zip. In this subsection we show that it is nevertheless a consequence of the formal requirement. In particular we show that a half-zip always constructs G-structures of (F-structures of the same shape).

Let us first recall how shape considerations are expressed. Recall that $(\mathsf{F}!_{\mathsf{A}})x$ is the shape of the F-structure x, and $\mathsf{F}!_{\mathsf{A}} \cdot f$ is the shape of the result of applying function f.

Now, for a natural transformation α of type $\mathsf{F} \hookleftarrow \mathsf{G}$, the shape characteristics of α in general are determined by α_1, since

$$\mathsf{F}!_{\mathsf{A}} \cdot \alpha_{\mathsf{A}} = \alpha_1 \cdot \mathsf{G}!_{\mathsf{A}}$$

That is, the shape of the result of applying α_{A} is completely determined by the behaviour of α_1. The shape characteristics of $\mathsf{zip.F.G}$, in particular, are determined by $(\mathsf{zip.F.G})_1$ since

$$\mathsf{GF}!_{\mathsf{A}} \cdot (\mathsf{zip.F.G})_{\mathsf{A}} = (\mathsf{zip.F.G})_1 \cdot \mathsf{FG}!_{\mathsf{A}}$$

Our shape requirement is that a half-zip maps an F-G-shape into a G-F-shape in which all F-shapes equal the original F-shape. This we can express by a single equation relating the behaviour of $(\mathsf{zip.F.G})_1$ to that of $\mathsf{fan.G}$. Specifically, we note that $(\mathsf{fan.G})_{\mathsf{F1}}$ generates from a given F-shape, x, a G-structure with an arbitrary shape in which all elements equal x, and thus have the same F-shape. On the other hand, $\mathsf{F}(\mathsf{fan.G})_1$, when applied to x, generates F-structures with shape x containing arbitrary G-shapes. The shape requirement is thus satisfied if we can establish the property

$$(\mathsf{fan.G})_{\mathsf{F1}} \;=\; (\mathsf{zip.F.G})_1 \cdot \mathsf{F}(\mathsf{fan.G})_1 \tag{5.10}$$

This property is an immediate consequence of the following lemma.

Lemma 5.11 If $\mathsf{fan.G}$ is the canonical fan of relator G then,

$$(\mathsf{fan.G})_{\mathsf{F}} \;=\; \mathsf{zip.F.G} \cdot \mathsf{F}(\mathsf{fan.G})$$

Proof The inclusion $\subseteq$ follows from the definition of half-zips (definition (5.6)) since $\mathsf{fan.G} : \mathsf{G} \hookleftarrow \mathsf{Id}$ and $\mathsf{zip.F.Id} = \mathsf{Fid}$. For $\supseteq$ we calculate:

$$\begin{array}{ll}
& (\mathsf{fan.G})_{\mathsf{F}} \supseteq \mathsf{zip.F.G} \cdot \mathsf{F}(\mathsf{fan.G}) \\
\equiv & \{ \quad \mathsf{fan.G} = \mathsf{mem.G}\backslash\mathsf{id} \quad \} \\
& (\mathsf{mem.G}\backslash\mathsf{id})_{\mathsf{F}} \supseteq \mathsf{zip.F.G} \cdot \mathsf{F}(\mathsf{fan.G}) \\
\equiv & \{ \quad \text{factors} \quad \} \\
& \mathsf{id}_{\mathsf{F}} \supseteq (\mathsf{mem.G})_{\mathsf{F}} \cdot \mathsf{zip.F.G} \cdot \mathsf{F}(\mathsf{fan.G}) \\
\Leftarrow & \{ \quad \mathsf{mem.G} : \mathsf{id} \hookleftarrow \mathsf{G}, \text{ definition half-zips } \mathsf{zip.F} \text{ (5.6)} \quad \} \\
& \mathsf{id}_{\mathsf{F}} \supseteq \mathsf{zip.F.id} \cdot \mathsf{F}(\mathsf{mem.G} \cdot \mathsf{fan.G}) \\
\Leftarrow & \{ \quad \mathsf{zip.F.id} = \mathsf{id}_{\mathsf{F}}, \text{ monotonicity} \quad \}
\end{array}$$

$$\begin{array}{ll} & \mathsf{id} \supseteq \mathsf{mem.G} \cdot \mathsf{fan.G} \\ \Leftarrow & \{ \quad \mathsf{fan.G} = \mathsf{mem.G} \backslash \mathsf{id}, \text{ factors } \} \\ & \textit{true} \end{array}$$

□

From equation (5.10) it also follows that the range of $(\mathsf{zip.F.G})_1$ is the range of $(\mathsf{fan.G})_{F1}$, i.e. arbitrary G-structures of which all elements are the same, but arbitrary, F-shape. Formally,

$$\begin{array}{ll} & ((\mathsf{zip.F.G})_1)_< \\ = & \{ \quad \text{definition fans (4.39): } (\mathsf{fan.G})_1 = \top\!\!\top_{G1,1}, \\ & \quad\ \text{hence, } ((\mathsf{fan.G})_1)_< = \mathsf{Gid}_1 \ \} \\ & ((\mathsf{zip.F.G})_1 \cdot \mathsf{F}(\mathsf{fan.G})_{1<})_< \\ = & \{ \quad \text{domains, (5.10)} \ \} \\ & ((\mathsf{fan.G})_{F1})_< \end{array}$$

Again, it follows from naturality of zip.F.G that the above interpretation of the shape of the range holds for $\mathsf{zip.F.G}_A$ too:

$$(\mathsf{GF!}_A \cdot (\mathsf{zip.F.G})_A)_< \;=\; ((\mathsf{zip.F.G})_1 \cdot \mathsf{FG!}_A)_< \;=\; ((\mathsf{zip.F.G})_1)_<$$

Note that nothing can be deduced about the shape of the G-structure from (5.10). As mentioned in section 5.2, a zip operation on an F-G-structure is a *partial* operation which is only defined on F-structures of (G-structures of the same shape). The requirement that a zip be the converse of a half-zip is a simple way of meeting this requirement: because zip.G.F is a half-zip, the domain of zip.F.G (which is the range of zip.G.F) consists of F-structures of (G-structures of the same shape) and zip.F.G respects the shape of the G-structures. Thus a zip of (F , G) is a partial operation which maps an F-structure of (G-structures of the same shape) into a G-structure of (F-structures of the same shape). Furthermore, the G-shape of the result is the same as all the original G-structures, and the shape of all the F-structures contained in the G-structure is the same as the original F-structure.

Before leaving lemma (5.11) let us note that it, together with naturality of zip.F.G, implies that zip.F.G has a slok property. Recall that $\hat{\mathsf{G}}$ denotes the fan-function of relator G defined by $\hat{\mathsf{G}}\mathsf{R} = \mathsf{GR} \cdot \mathsf{fan.G}_B$ for $\mathsf{R} : A \leftarrow B$. Then we have,

$$\hat{\mathsf{G}}\mathsf{FR} = \mathsf{zip.F.G}_A \cdot \mathsf{F}\hat{\mathsf{G}}\mathsf{R} \quad \text{for each } \mathsf{R} : A \leftarrow B \tag{5.12}$$

or, equivalently,

$$\mathsf{zip.F.G} : \hat{\mathsf{G}}\mathsf{F} \xleftarrow{\mathsf{Slok}} \mathsf{F}\hat{\mathsf{G}} \quad .$$

The proof proceeds as follows

$$
\begin{array}{ll}
 & \mathsf{zip.F.G} \cdot \mathsf{F\hat{G}R} \\
= & \quad \{ \quad \text{definition } \hat{\mathsf{G}} \quad \} \\
 & \mathsf{zip.F.G} \cdot \mathsf{F(GR} \cdot \mathsf{fan.G)} \\
= & \quad \{ \quad \text{naturality } \mathsf{zip.F.G} \quad \} \\
 & \mathsf{GFR} \cdot \mathsf{zip.F.G} \cdot \mathsf{F(fan.G)} \\
= & \quad \{ \quad \text{lemma (5.11)} \quad \} \\
 & \mathsf{GFR} \cdot (\mathsf{fan.G})_{\mathsf{F}} \\
= & \quad \{ \quad \text{definition } \hat{\mathsf{G}} \quad \} \\
 & \hat{\mathsf{G}}\mathsf{FR}
\end{array}
$$

5.3.3 Commuting relators

We say that a class of relators is *commuting* if the class contains Id, is closed under composition and all its elements are pair-wise commuting. The main concern of this subsection is to explore the naturality properties of zips defined on a commuting class of relators.

As a preliminary observation we remark that, although requirement (5.6b) for half-zips is stated with an inclusion, we have an equality in some cases:

Lemma 5.13
(a) For cofunction α and $\alpha : \mathsf{G} \hookrightarrow \mathsf{H}$,

$$\alpha_{\mathsf{F}} \cdot \mathsf{zip.F.H} \supseteq \mathsf{zip.F.G} \cdot \mathsf{F}\alpha \ ,$$

(b) For cofunction α and $\alpha : \mathsf{G} \leftarrow \mathsf{H}$,

$$\alpha_{\mathsf{F}} \cdot \mathsf{zip.F.H} = \mathsf{zip.F.G} \cdot \mathsf{F}\alpha \ .$$

Proof Part (b) follows from combining part (a) with (5.6b). For (a) we calculate:

$$
\begin{array}{ll}
 & \alpha_{\mathsf{F}} \cdot \mathsf{zip.F.H} \supseteq \mathsf{zip.F.G} \cdot \mathsf{F}\alpha \\
\equiv & \quad \{ \quad \alpha \text{ cofunction, shunting} \quad \} \\
 & \mathsf{zip.F.H} \cdot \mathsf{F}\alpha^{\circ} \supseteq \alpha_{\mathsf{F}}^{\circ} \cdot \mathsf{zip.F.G} \\
\equiv & \quad \{ \quad \alpha^{\circ} : \mathsf{H} \hookleftarrow \mathsf{G}, \text{ definition half-zips (5.6b)} \quad \} \\
 & \textit{true}
\end{array}
$$

□

Combining the properties of both half-zips gives

Lemma 5.14 Let ξ denote a class of commuting relators . Then for all relators F, G and H in ξ,

(a) $\mathsf{zip.F.G} : \mathsf{GF} \leftarrow \mathsf{FG}$,

(b) $\alpha_F \cdot \mathsf{zip}.F.H \subseteq \mathsf{zip}.F.G \cdot F\alpha$ for each $\alpha : G \hookleftarrow H$,

(c) $F\alpha \cdot \mathsf{zip}.H.F \supseteq \mathsf{zip}.G.F \cdot \alpha_F$ for each $\alpha : G \hookrightarrow H$,

(d) $\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_H$,

(e) $\mathsf{zip}.FG.H = (\mathsf{zip}.F.H)_G \cdot F(\mathsf{zip}.G.H)$,

(f) $\mathsf{zip}.F.Id = \mathsf{id}_F = \mathsf{zip}.Id.F$,

(g) $\mathsf{zip}.F.G = (\mathsf{zip}.G.F)^\circ$.

□

Note that property (5.14c) implies the dual properties of lemma (5.13) and lemma (5.11). That is to say, we have

Lemma 5.15 Let ξ denote a class of commuting relators. Then for all relators F, G and H in ξ,

(a) for function α and $\alpha : G \hookleftarrow H$,

$$F\alpha \cdot \mathsf{zip}.H.F \subseteq \mathsf{zip}.G.F \cdot \alpha_F$$

(b) for function α and $\alpha : G \leftarrow H$,

$$F\alpha \cdot \mathsf{zip}.H.F = \mathsf{zip}.G.F \cdot \alpha_F$$

(c) if $\mathsf{fan}.F$ is the canonical fan of relator F then

$$G(\mathsf{fan}.F)^\circ \cdot \mathsf{zip}.F.G = (\mathsf{fan}.F)^\circ_G$$

□

Note that property (5.15c) together with naturality of $\mathsf{zip}.F.G$ implies the dual slok property of (5.12). Recall that the cofan function $\check{F}$ is defined by $\check{F}R = (\mathsf{fan}.F)^\circ_A \cdot FR$ for $R : A \leftarrow B$, so $\check{F}R = (\hat{F}(R^\circ))^\circ$. So, we have for commuting relators (F, G)

$$G\check{F}R \cdot (\mathsf{zip}.F.G)_B = \check{F}GR \quad \text{for each } R : A \leftarrow B,$$

or, equivalently, $\mathsf{zip}.F.G$ is an arrow having a coslok property,

$$\mathsf{zip}.F.G : G\check{F} \xleftarrow{\mathsf{CoSlok}} \check{F}G$$

5.3.4 Strength

In this section and the next we consider some more specific cases of "zips".

Several scientists have argued that the notion of functor is too general to capture the notion of a datatype as understood by programmers. Moggi [39] claims that the notion of "strength" is fundamental to computation, "strength" being defined as follows.

Definition 5.16 (Strength) A natural transformation $\mathsf{str}_{A,B} : F(A \times B) \hookleftarrow FA \times B$ is said to be a *strength* of relator F iff $\mathsf{str}_{A\,,\,B}$ is a function that behaves coherently with respect to product in the following sense. First, the diagram

$$\begin{tikzcd} F(A \times 1) \arrow[dr, "\mathsf{Frid}_A"'] & & FA \times 1 \arrow[ll, "\mathsf{str}_{A,1}"'] \arrow[dl, "\mathsf{rid}_{FA}"] \\ & FA & \end{tikzcd}$$

(where $\mathsf{rid}_A : A \leftarrow A \times 1$ is the obvious natural isomorphism) commutes. Second, the diagram

$$\begin{tikzcd} FA \times (B \times C) \arrow[dd, "\mathsf{str}_{A\,,\,B\times C}"'] & (FA \times B) \times C \arrow[l, "\mathsf{ass}_{FA,B,C}"'] \arrow[d, "\mathsf{str}_{A,B} \times \mathsf{id}_C"] \\ & F(A \times B) \times C \arrow[d, "\mathsf{str}_{A\times B\,,\,C}"] \\ F(A \times (B \times C)) & F((A \times B) \times C) \arrow[l, "\mathsf{Fass}_{A,B,C}"] \end{tikzcd}$$

(where $\mathsf{ass}_{A,B,C} : A \times (B \times C) \leftarrow (A \times B) \times C$ is the obvious natural isomorphism) commutes as well. A relator that has at least one strength is said to be *strong*.
□

We have no idea why the term "strength" has been chosen to name this property of a relator. The idea behind it is however very simple. A relator F is "strong" if, for each pair of types A and B, it is possible to broadcast a given value of type B to every element in an F-structure of A's. The broadcasting operation is what Moggi calls the "strength" of the relator.

Note that the requirement $\mathsf{Frid}_A \cdot \mathsf{str}_{A\,,1} = \mathsf{rid}_{FA}$ is equivalent to

$$\mathsf{Foutl}_{A\,,\,B} \cdot \mathsf{str}_{A\,,\,B} = \mathsf{outl}_{FA\,,\,B} \tag{5.17}$$

since using $\mathsf{rid}_A = \mathsf{outl}_{A\,,1}$ and $\mathsf{outl}_{A\,,\,B} = \mathsf{outl}_{A\,,1} \cdot \mathsf{id}_A \times !_B$

$$\begin{aligned} &\mathsf{Foutl}_{A\,,\,B} \cdot \mathsf{str}_{A\,,\,B} = \mathsf{Foutl}_{A\,,1} \cdot F(\mathsf{id}_A \times !_B) \cdot \mathsf{str}_{A\,,\,B} = \mathsf{Foutl}_{A\,,1} \cdot \mathsf{str}_{A\,,1} \cdot \mathsf{id}_{FA} \times !_B \\ &= \mathsf{outl}_{FA\,,1} \cdot \mathsf{id}_{FA} \times !_B = \mathsf{outl}_{FA\,,\,B} \end{aligned}$$

The type of the strength $\mathsf{str}_{A,B}$ of relator F is the same as the type of $(\mathsf{zip}.(\times B).F)_A$, namely $F(A \times B) \hookleftarrow FA \times B$. We shall argue in this section that, if F and the family of relators $(\times B)$

are included in a class of commuting relators, then any relation satisfying the requirements of $(\mathsf{zip}.(\times \mathsf{B}).\mathsf{F})_{\mathsf{A}}$ also satisfies the definition of $\mathsf{str}_{\mathsf{A},\mathsf{B}}$. Furthermore, we show in chapter 6 that a relator with membership has a unique strength. In other words, a strength of relator F with membership *is* the half-zip $(\mathsf{zip}.(\times \mathsf{B}).\mathsf{F})_{\mathsf{A}}$.

Let us begin with an informal scrutiny of the definition of strength. In the introduction to this chapter we remarked that a broadcast operation (a "strength") is an example of a (half-)zip. Specifically, a broadcast operation is a zip of the form $(\mathsf{zip}.(\times \mathsf{B}).\mathsf{F})_{\mathsf{A}}$. Paying due attention to the fact that the relator F is a parameter of the definition, we observe that all the natural transformations involved in the definition of strength are special cases of a broadcast operation and thus of zips.

In the first diagram there are two occurrences of the canonical isomorphism rid. In general, we recognise a projection of type $\mathsf{A} \leftarrow \mathsf{A} \times \mathsf{B}$ as a broadcast where the parameter F is instantiated to the constant relator K_{A}. Thus $\mathsf{rid}_{\mathsf{A}}$ is $(\mathsf{zip}.(\times 1).\mathsf{K}_{\mathsf{A}})_{\mathsf{B}}$ for some arbitrary B. In words, $\mathsf{rid}_{\mathsf{A}}$ commutes the relators $(\times 1)$ and K_{A}. Redrawing the first diagram above, using that all the arrows are broadcasts and thus zips, we get the following diagram[2].

$$\begin{tikzcd}
\mathsf{F}(\mathsf{A} \times 1) \arrow[dr, "\mathsf{Fzip}.(\times 1).(\mathsf{K}_{\mathsf{A}})"'] & & \mathsf{FA} \times 1 \arrow[ll, "(\mathsf{zip}.(\times 1).\mathsf{F})_{\mathsf{K}_{\mathsf{A}}}"'] \arrow[dl, "\mathsf{zip}.(\times 1).(\mathsf{K}_{\mathsf{F}.\mathsf{A}})"] \\
& \mathsf{FA} &
\end{tikzcd}$$

Comparing with (5.4) we see that the first coherence property of strengths is just an instance of the general requirement on half-zips that they respect composition of functors.

Now we turn to the second diagram in the definition of strength. Just as we observed that rid is an instance of a broadcast and thus a zip, we also observe that ass is a broadcast and thus a zip. Specifically, $\mathsf{ass}_{\mathsf{A},\mathsf{B},\mathsf{C}}$ is $(\mathsf{zip}.(\times \mathsf{C}).(\mathsf{A}\times))_{\mathsf{B}}$. Once again, every edge in the diagram involves a zip operation! That is not all. Yet more zips can be added to the diagram. For our purposes it is crucial to observe that the bottom left and middle right nodes —the nodes la-

[2]To be perfectly correct we should instantiate each of the transformations at some arbitrary B. We haven't done so because the choice of which B in this case is truly irrelevant.

belled $\mathsf{F}(A \times (B \times C))$ and $\mathsf{F}(A \times B) \times C$— are connected by the edge $(\mathsf{zip}.(\times C).\mathsf{F}(A\times))_B$.

$$\begin{tikzcd}
\mathsf{F}A \times (B \times C) \arrow[dd, "(\mathsf{zip}.(\times(B\times C)).\mathsf{F})_A"'] & & (\mathsf{F}A \times B) \times C \arrow[ll, "(\mathsf{zip}.(\times C).(\mathsf{F}A\times))_B"'] \arrow[d, "(\mathsf{zip}.(\times B).\mathsf{F})_A \times \mathsf{id}_C"] \\
 & & \mathsf{F}(A \times B) \times C \arrow[dll, "(\mathsf{zip}.(\times C).\mathsf{F}(A\times))_B"'] \arrow[d, "(\mathsf{zip}.(\times C).\mathsf{F})_{A\times B}"] \\
\mathsf{F}(A \times (B \times C)) & & \mathsf{F}((A \times B) \times C) \arrow[ll, "\mathsf{F}(\mathsf{zip}.(\times C).(A\times))_B"]
\end{tikzcd}$$

This means that we can decompose the original coherence property into a combination of two properties of zips.

The first property, represented by the bottom triangle, is just the requirement that $\mathsf{zip}.(\times C)$ respect composition of functors (requirement (5.4) once more).

$$\mathsf{zip}.(\times C).\mathsf{F}(A\times) \;=\; \mathsf{F}\mathsf{zip}.(\times C).(A\times) \cdot (\mathsf{zip}.(\times C).\mathsf{F})_{(A\times)}$$

The second property is less easy to match with the requirements on zips. Reading off from the diagram (the inner path from the top right node to the bottom left node) we require that

$$\begin{array}{ll}
 & (\mathsf{zip}.(\times(B\times C)).\mathsf{F})_A \cdot (\mathsf{zip}.(\times C).((\mathsf{F}A)\times))_B \\
= & \\
 & (\mathsf{zip}.(\times C).\mathsf{F}(A\times))_B \cdot (\mathsf{zip}.(\times B).\mathsf{F})_A \times \mathsf{id}_C
\end{array}$$

Careful inspection reveals that it is an instance of requirement (5.3). To see this define α by $\alpha_B = (\mathsf{zip}.(\times B).\mathsf{F})_A$ and instantiate functors F, G and H to $(\times C)$, $((\mathsf{F}A)\times)$ and $\mathsf{F}(A\times)$, respectively. In order to apply (5.3) we require that α_B be natural in the parameter B. In section 5.9 we show that this is indeed the case. Now, applying (5.3) we obtain the inclusion

$$\begin{array}{ll}
 & (\mathsf{zip}.(\times(B\times C)).\mathsf{F})_A \cdot (\mathsf{zip}.(\times C).((\mathsf{F}A)\times))_B \\
\subseteq & \\
 & (\mathsf{zip}.(\times C).\mathsf{F}(A\times))_B \cdot (\mathsf{zip}.(\times B).\mathsf{F})_A \times \mathsf{id}_C
\end{array}$$

(for all B) rather than equality. In section 5.9 we show that $(\mathsf{zip}.(\times B).\mathsf{F})_A$ is a function. Hence, equality follows because an inclusion between functions is equivalent to their equality.

Let us summarise this section with a formal statement of what we have observed. The focus of our discussion has been a family of broadcast operations $\mathsf{zip}.(\times C)$ indexed by C and some class of relators. We have observed that if this family is natural in the parameter C and, for each C, $\mathsf{zip}.(\times C)$ defines a class of functional half-zips then each element $\mathsf{zip}.(\times C).\mathsf{F}$ is a "strength" of the relator F.

In section 5.6 we show how to construct such a family of half-zips. By focusing on the parameter F in the definition of strength we have obtained a beautiful characterisation of the notion.

5.3.5 Structure multiplication

Another example[3] of the beauty of the "zip" generalisation is afforded by what we shall call "structure multiplication". A simple, concrete example of structure multiplication is the following. Given two lists $[a_1\,,a_2\,,\ldots]$ and $[b_1\,,b_2\,,\ldots]$, form a matrix in which the $(i\,,j)$th element is the pair $(a_i\,,b_j)$. We call this "structure multiplication" because the input type is the product $\mathit{List}\mathsf{A}\times\mathit{List}\mathsf{B}$ for some types A and B.

Given certain basic functions, this task may be completed in one of two ways. The first way has two steps. First, the list of a's is broadcast over the list of b's to form the list

$$[([a_1\,,a_2\,,\ldots]\,,b_1)\,,([a_1\,,a_2\,,\ldots]\,,b_2)\,,\ldots]$$

Then each b is broadcast over the list of a's. The second way is identical but for an interchange of "a" and "b".

Both methods return a list of lists, but the results are not identical. The connection between the two results is that one is the transpose of the other. The two methods and the connection between them are summarised in the following diagram.

$$\begin{tikzcd}
 & \mathit{List}\mathsf{A}\times\mathit{List}\mathsf{B} \arrow[dl] \arrow[dr] & \\
\mathit{List}(\mathit{List}\mathsf{A}\times\mathsf{B}) \arrow[d] & & \mathit{List}(\mathsf{A}\times\mathit{List}\mathsf{B}) \arrow[d] \\
\mathit{ListList}(\mathsf{A}\times\mathsf{B}) \arrow[rr, leftrightarrow] & & \mathit{ListList}(\mathsf{A}\times\mathsf{B})
\end{tikzcd}$$

The generalisation of this procedure is obvious: replace $\mathit{List}\mathsf{A}$ by FA and $\mathit{List}\mathsf{B}$ by GB for some arbitrary relators F and G. Doing so leads to the realisation that every step involves a "zip" operation. This is made explicit in the diagram below.

[3] This example was communicated to Roland Backhouse by D.J. Lillie in December,1994.

$$\begin{tikzcd}
 & \mathsf{F}\mathsf{A} \times \mathsf{G}\mathsf{B} \arrow[dl, "(\mathsf{zip}.(\mathsf{FA}\times).\mathsf{G})_\mathsf{B}"'] \arrow[dr, "(\mathsf{zip}.(\times\mathsf{GB}).\mathsf{F})_\mathsf{A}"] & \\
\mathsf{G}(\mathsf{FA} \times \mathsf{B}) \arrow[d, "\mathsf{G}(\mathsf{zip}.(\times\mathsf{B}).\mathsf{F})_\mathsf{A}"'] & & \mathsf{F}(\mathsf{A} \times \mathsf{GB}) \arrow[d, "\mathsf{F}(\mathsf{zip}.(\mathsf{A}\times).\mathsf{G})_\mathsf{B}"] \arrow[dll, "(\mathsf{zip}.\mathsf{F}(\mathsf{A}\times).\mathsf{G})_\mathsf{B}"'] \\
\mathsf{GF}(\mathsf{A} \times \mathsf{B}) & & \mathsf{FG}(\mathsf{A} \times \mathsf{B}) \arrow[ll, "(\mathsf{zip}.\mathsf{F}.\mathsf{G})_{\mathsf{A}\times\mathsf{B}}"]
\end{tikzcd}$$

An extra edge has been added to the diagram in order to show how the commutativity of the diagram can be decomposed into smaller parts[4]. Thus we have reduced our proof obligation into two separate conditions.

The first condition, the commutativity of the triangle in the bottom right of the diagram, is:

$$(\mathsf{zip}.\mathsf{F}(\mathsf{A}\times).\mathsf{G})_\mathsf{B} \;=\; (\mathsf{zip}.\mathsf{F}.\mathsf{G})_{\mathsf{A}\times\mathsf{B}} \cdot \mathsf{F}(\mathsf{zip}.(\mathsf{A}\times).\mathsf{G})_\mathsf{B}$$

This is clearly property (5.14e). (take $\mathsf{G} = \mathsf{F}$ and $\mathsf{H} = (\mathsf{A}\times)$).

The second condition, the commutativity of the remainder of the diagram, is:

$$\mathsf{G}(\mathsf{zip}.(\times\mathsf{B}).\mathsf{F})_\mathsf{A} \cdot (\mathsf{zip}.(\mathsf{FA}\times).\mathsf{G})_\mathsf{B} \;=\; (\mathsf{zip}.\mathsf{F}(\mathsf{A}\times).\mathsf{G})_\mathsf{B} \cdot (\mathsf{zip}.(\times\mathsf{GB}).\mathsf{F})_\mathsf{A}$$

This is an instance of (5.15b) (take $\alpha_\mathsf{B} = (\mathsf{zip}.(\times\mathsf{B}).\mathsf{F})_\mathsf{A}$, $\mathsf{F} = \mathsf{G}$, $\mathsf{G} = \mathsf{F}(\mathsf{A}\times)$ and $\mathsf{H} = (\mathsf{FA}\times)$). (Note that it is required that the broadcast operation $(\mathsf{zip}.(\times\mathsf{B}).\mathsf{F})_\mathsf{A}$ is a function and natural in the parameter B. In section 5.9 we verify that this is indeed the case.)

Formally the claim is that the diagram headed by the node $\mathsf{FA} \times \mathsf{GB}$ commutes provided that F and G are elements of a class of commuting relators that includes in addition all relators of the form $(\mathsf{A}\times)$ and $(\times\mathsf{B})$ and such that the broadcast operations $(\mathsf{zip}.(\times\mathsf{B}).\mathsf{F})_\mathsf{A}$ are functions and natural in B. (Commutativity of the symmetric diagram requires that the broadcast operations $(\mathsf{zip}.(\times\mathsf{B}).\mathsf{G})_\mathsf{A}$ are also functions.)

5.4 The requirement (arbitrary relators)

Until now, we have said nothing about zips for non-endo relators. We assumed that both compositions FG and GF existed, hence F and G are endorelators of the same type. For the regular relators we have to include non-endo relators as well. For example, the standard zip function on a pair of lists, i.e. $\mathsf{zip}.\times.\mathit{List}$, mentioned in the introduction has type

$$(\mathsf{zip}.\times.\mathit{List})_{\mathsf{A}\,,\,\mathsf{B}} \;:\; \mathit{List}(\mathsf{A}\times\mathsf{B}) \leftarrow \mathit{List}\mathsf{A} \times \mathit{List}\mathsf{B}$$

[4]The additional edge together with the removal of the right-pointing edge in the bottom line seem to make the diagram asymmetric. But, of course, there are symmetric edges. Corresponding to the added diagonal edge there is an edge connecting $\mathsf{G}(\mathsf{FA}\times\mathsf{B})$ and $\mathsf{FG}(\mathsf{A}\times\mathsf{B})$ but only one of these edges is needed in the argument that follows.

So, $\mathsf{zip}.\times.\mathit{List}$ is a natural transformation of type $\mathit{List}(\times) \leftarrow (\times)\mathit{List}^2$. Recall that "$\_^2$" denotes the doubling functor on relators, i.e. $F^2(R\,,S) = (FR\,,FS)$.

For the definition of $\mathsf{zip}.F.G$ we want to include the case that F and G are single-valued. For the remainder of this section, we assume that F and G are single-valued.

Furthermore, we assume for the moment that G is an endorelator as is the case for the example. The definition of $\mathsf{zip}.F.G$ for a relator F of type $1 \leftarrow k$ is then derived from the original definition by replacing the composition FG by $F(G^k)$ and doing the same for related compositions of relators and natural transformations. So, for a single-valued relator F the definition of a half-zip becomes:

Definition 5.18 For relator F of type $1 \leftarrow k$ and G and H endorelators,
(a) $\mathsf{zip}.F.G \;:\; GF \leftarrow F(G^k)$,

(b) $\alpha_F \cdot \mathsf{zip}.F.H \subseteq \mathsf{zip}.F.G \,\cdot\, F(\alpha^k)$ for each $\alpha : G \hookleftarrow H$,

(c) $\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \,\cdot\, (\mathsf{zip}.F.G)_{H^k}$,

(d) $\mathsf{zip}.F.\mathrm{Id} = F(\mathsf{id}^k)$.
$\square$

For instance, take the case of a binary relator $\otimes$. Filling in the objects, the definition reads

(a) $(\mathsf{zip}.\otimes.G)_{A\,,B} \;:\; G(A{\otimes}B) \leftarrow GA \otimes GB$,

(b) $\alpha_{A\otimes B} \cdot (\mathsf{zip}.\otimes.H)_{A\,,B} \subseteq (\mathsf{zip}.\otimes.G)_{A\,,B} \,\cdot\, \alpha_A{\otimes}\alpha_B$ for each $\alpha : G \hookleftarrow H$,

(c) $(\mathsf{zip}.\otimes.GH)_{A\,,B} = G(\mathsf{zip}.\otimes.H)_{A\,,B} \cdot (\mathsf{zip}.\otimes.G)_{HA\,,HB}$,

(d) $(\mathsf{zip}.\otimes.\mathrm{Id})_{A\,,B} = \mathsf{id}_A{\otimes}\mathsf{id}_B$.

Similarly, if we take for F an endorelator and we allow non-endo relators of type $1 \leftarrow k$ for the second component of the zip function we have to replace the composition GF by $G(F^k)$ and do the same for related compositions.

Furthermore, since we have included the projection relators as well, we add the requirement that $\mathsf{zip}.F$ is also coherent with projections. For a projection relator $\mathtt{Proj} : 1 \leftarrow k$, the projection on a element of a vector of length k, we demand:

$$\mathsf{zip}.F.\mathtt{Proj} = \mathsf{id}_{F(\mathtt{Proj})} \tag{5.19}$$

The identity relator is a projection relator on a vector of length 1. Hence, requirement (5.19) is a generalisation of the requirement that $\mathsf{zip}.F$ respects identities. So, we replace condition (d) by requirement (5.19).

The extended definition of $\mathsf{zip}.F$ becomes:

Definition 5.20 For endorelator F,
(a) $\mathsf{zip}.F.G \;:\; G(F^k) \leftarrow FG$, for $G : 1 \leftarrow k$,

(b) $\alpha_{F^k} \cdot \mathsf{zip}.F.H \subseteq \mathsf{zip}.F.G \cdot F\alpha$ for each $\alpha : G \hookleftarrow H$ and $G, H : 1 \leftarrow k$.

(c) $\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_H$ for endo G and $H : 1 \leftarrow k$,

(d) $\mathsf{zip}.F.\mathsf{Proj} = \mathsf{id}_{F(\mathsf{Proj})}$ for each $\mathsf{Proj} : 1 \leftarrow k$.
□

Again, instantiating the second argument of zip with a binary relator and filling in the objects gives:

(a) $(\mathsf{zip}.F.\otimes)_{A,B} \ : \ FA \otimes FB \leftarrow F(A \otimes B)$

(b) $\alpha_{FA,FB} \cdot (\mathsf{zip}.F.\oplus)_{A,B} \subseteq (\mathsf{zip}.F.\otimes)_{A,B} \cdot F\alpha$ for each $\alpha : \otimes \hookleftarrow \oplus$.

(d) $(\mathsf{zip}.F.\mathsf{Outl})_{A,B} = \mathsf{id}_{FA}$ and $(\mathsf{zip}.F.\mathsf{Outr})_{A,B} = \mathsf{id}_{FB}$

Next we give the definition of zip for both arguments being not necessarily endorelators. A first attempt would be to take the combination of both definitions but this is too simple. For instance for $F : 1 \leftarrow k$ and $G : 1 \leftarrow l$ the naturality requirement becomes:

$$\mathsf{zip}.F.G \ : \ G(F^l) \leftarrow F(G^k)$$

but $G(F^l)$ and $F(G^k)$ do not have the same type: the source of the former is $k{*}l$ whereas the source of the latter is $l{*}k$. Of course, the relationship between both types is matrix transposition τ. In other words, we take $G(F^l)\tau$ instead of $G(F^l)$. Note that for single-valued F, we have $G(F^l)\tau = G({}^lF)$.

Now, the definition of zips for single valued relators is derived from the original one by replacing FG by $F(G^k)$, and the same for related compositions of relators and natural transformations., and replacing GF by $G({}^lF)$, and the same for related compositions.

Definition 5.21 For relator $F : 1 \leftarrow k$, the members of the collection $\mathsf{zip}.F.G$, for each single-valued relator G, are called *half-zips* iff,
(a) $\mathsf{zip}.F.G \ : \ G({}^lF) \leftarrow F(G^k)$, for each $G : 1 \leftarrow l$

(b) $\alpha_{({}^lF)} \cdot \mathsf{zip}.F.H \subseteq \mathsf{zip}.F.G \cdot F(\alpha^k)$ for each $\alpha : G \hookleftarrow H$ and $G, H : 1 \leftarrow l$,

(c) $\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_{H^k}$ for $G : 1 \leftarrow 1$ and $H : 1 \leftarrow l$,

(d) $\mathsf{zip}.F.\mathsf{Proj} = \mathsf{id}_{F(\mathsf{Proj}^k)}$ for each $\mathsf{Proj} : 1 \leftarrow l$.
□

Since the definition of $\mathsf{zip}.F.G$ for non-endo relators is derived from the original one by replacing relator composition by adding "$\_^k$" and "${}^l\_$" at the appropriate places, results for the non-endo case can be derived from results for the unary one by making the same replacement. These adjustments follow from arity considerations.

Commuting relators Since we have generalized the definition of half-zips, we have to generalize the definition of commuting relators. Note that for $F : 1 \leftarrow k$ and $G : 1 \leftarrow l$, we have $\mathsf{zip}.F.G \ : \ G({}^{l}F) \leftarrow F(G^{k})$ and $(\mathsf{zip}.G.F)^{\circ} \ : \ G(F^{l}) \leftarrow F({}^{k}G)$ so for non-endo F and G they do not have the same type, hence, we can not demand $\mathsf{zip}.F.G = (\mathsf{zip}.G.F)^{\circ}$. However, the relationship between $G({}^{l}F)$ and $G(F^{l})$, and between $F(G^{k})$ and $F({}^{k}G)$ is matrix transposition, i.e. the relator τ, since

$$G({}^{l}F) = G(F^{l})\tau \quad \text{and} \quad F(G^{k}) = F({}^{k}G)\tau$$

Hence, we have $(\mathsf{zip}.G.F)^{\circ}_{\tau} \ : \ G(F^{l})\tau \leftarrow F({}^{k}G)\tau = G({}^{l}F) \leftarrow F(G^{k})$. So, the general definition becomes:

Definition 5.22 The half-zip $\mathsf{zip}.F.G$ for single-valued relators F and G, is said to be a *zip* of (F, G) if there exists a half-zip $\mathsf{zip}.G.F$ such that

$$\mathsf{zip}.F.G = (\mathsf{zip}.G.F)^{\circ}_{\tau}$$

□

5.4.1 Multi-valued relators

Until now we have assumed that both relators F and G are single valued, i.e. relators of type $1 \leftarrow k$, for some k. Next we define $\mathsf{zip}.F.G$ for $F : m \leftarrow k$ and $G : n \leftarrow l$. We try to extend the definition (5.21). From arity consideration, it follows that we should generalize $F(G^{k})$ to $({}^{n}F)(G^{k})$. Similarly, we take $(G^{m})({}^{l}F)$ instead of $G({}^{l}F)$. Making the same adjustments for related compositions of relators and natural transformations yields the following definition.

Definition 5.23 For relator $F : m \leftarrow k$, the members of the collection $\mathsf{zip}.F.G$, for each relator $G : n \leftarrow l$, are called *half-zips* iff,
(a) $\mathsf{zip}.F.G \ : \ (G^{m})({}^{l}F) \leftarrow ({}^{n}F)(G^{k})$, for each $G : n \leftarrow l$

(b) $(\alpha^{m})_{({}^{l}F)} \cdot \mathsf{zip}.F.H \subseteq \mathsf{zip}.F.G \cdot ({}^{n}F)(\alpha^{k})$ for each $\alpha : G \hookleftarrow H$ and $G , H : n \leftarrow l$,

(c) $\mathsf{zip}.F.GH = (G^{m})(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_{H^{k}}$ for $G : n \leftarrow l$ and $H : l \leftarrow o$,

(d) $\mathsf{zip}.F.\mathsf{Proj} = \mathsf{id}_{F(\mathsf{Proj}^{k})}$ for each $\mathsf{Proj} : 1 \leftarrow l$.
□

From property (5.23c) and (5.23d) it follows that the mapping $\mathsf{zip}.F$ is coherent with tupling in the following sense.

Lemma 5.24 For relators $F : m \leftarrow k$ and $G : n \leftarrow l$, we have,

$$(\mathsf{zip}.F.G =)\ \mathsf{zip}.F.\Delta_{n}G_{n} = \tau\Delta_{n}\mathsf{zip}.F.G_{n}$$

Proof We aim to use the universal property for arbitrary products,

$$\mathsf{zip}.F.G = \tau\Delta_{n}\mathsf{zip}.F.G_{n}$$

$$
\begin{array}{ll}
\equiv & \{ \quad \tau \text{ isomorphism } \} \\
& \tau \mathsf{zip.F.G} = \Delta_n \mathsf{zip.F.G}_n \\
\equiv & \{ \quad \text{universal property } \Delta_n \ \} \\
& \mathsf{Proj}\tau\mathsf{zip.F.G} = \mathsf{zip.F.G}_n \\
\equiv & \{ \quad \mathsf{Proj}\tau = \mathsf{Proj}^m;\ (\mathsf{Proj}^m)({}^n\mathsf{F}) = \mathsf{F}(\mathsf{Proj}^k)\text{, property (5.23d) } \} \\
& (\mathsf{Proj}^m)\mathsf{zip.F.G} \cdot (\mathsf{zip.F.Proj})_{\mathsf{G}^k} = \mathsf{zip.F.G}_n \\
\equiv & \{ \quad \text{compositionality (5.23c) } \} \\
& \mathsf{zip.F.ProjG} = \mathsf{zip.F.G}_n \\
\equiv & \{ \quad \mathsf{ProjG} = \mathsf{G}_n \ \} \\
& \textit{true}
\end{array}
$$

□

So, from lemma (5.24) it follows that $\mathsf{zip.F.G}$ can be expressed in terms of the collection of half-zips $\mathsf{zip.F.G}_n$, half-zips for which the second argument is a single-valued relator. It is even the case that we can define $\mathsf{zip.F.G}$ in this way:

Lemma 5.25 For fixed relators $\mathsf{F} : m \leftarrow k$ and for each relator $\mathsf{G} : n \leftarrow l$,

$$\mathsf{zip.F.G} \triangleq \tau\Delta_n \mathsf{zip.F.G}_n$$

defines a collection of half-zips given that $\mathsf{zip.F.H}$ is a half-zip for each single-valued relator H.

Proof Since τ and Δ_n are functors it follows from $\mathsf{zip.F.G}_n : (\mathsf{G}_n{}^m)({}^l\mathsf{F}) \leftarrow \mathsf{F}(\mathsf{G}_n{}^k)$ that

$$\tau\Delta_n \mathsf{zip.F.G}_n \ : \ \tau\Delta_n(\mathsf{G}_n{}^m)({}^l\mathsf{F}) \ \leftarrow \ \tau\Delta_n\mathsf{F}(\mathsf{G}_n{}^k) \ ,$$

since $\tau\Delta_n(\mathsf{G}_n{}^m) = \mathsf{G}^m$ and $\tau\Delta_n\mathsf{F}(\mathsf{G}_n{}^k) = ({}^n\mathsf{F})(\mathsf{G}^k)$, we have,

$$\tau\Delta_n \mathsf{zip.F.G}_n : (\mathsf{G}^m)({}^l\mathsf{F}) \leftarrow ({}^n\mathsf{F})(\mathsf{G}^k) \ ,$$

as required by (5.23a). For requirement (5.23b) we calculate, for each $\mathsf{H} : n \leftarrow l$ and $\alpha : \mathsf{G} \hookleftarrow \mathsf{H}$,

$$(\alpha^m)_{({}^l\mathsf{F})} \cdot \mathsf{zip.F.H}_n \subseteq \mathsf{zip.F.G}_n \cdot \mathsf{F}(\alpha^k)$$

we calculate

$$
\begin{array}{ll}
& (\alpha^m)_{({}^l\mathsf{F})} \cdot \mathsf{zip.F.H} \subseteq \mathsf{zip.F.G} \cdot ({}^n\mathsf{F})(\alpha^k) \\
\equiv & \{ \quad \text{definition } \mathsf{zip.F},\ \alpha^m = \tau\Delta_n(\alpha_n{}^m),\ ({}^n\mathsf{F})(\alpha^k) = \tau\Delta_n\mathsf{F}(\alpha_n{}^k) \ \} \\
& \tau\Delta_n(\alpha_n{}^m)_{({}^l\mathsf{F})} \cdot \tau\Delta_n\mathsf{zip.F.H}_n \subseteq \tau\Delta_n\mathsf{zip.F.G}_n \cdot \tau\Delta_n\mathsf{F}(\alpha_n{}^k) \\
\equiv & \{ \quad \tau\Delta_n \text{ relator } \} \\
& \tau\Delta_n((\alpha_n{}^m)_{({}^l\mathsf{F})} \cdot \mathsf{zip.F.H}_n) \subseteq \tau\Delta_n(\mathsf{zip.F.G}_n \cdot \mathsf{F}(\alpha_n{}^k))
\end{array}
$$

$$\equiv \quad \{ \quad \tau \text{ isomorphism; product } \}$$

$$(\alpha_n{}^m)_{(\iota F)} \cdot \mathsf{zip}.F.H_n \subseteq \mathsf{zip}.F.G_n \cdot F(\alpha_n{}^k)$$

$$\equiv \quad \{ \quad H_n \text{ and } G_n \text{ single-valued, assumption } \}$$

true

Requirement (5.23c) we verify by, for $G : n \leftarrow l$ and $H : l \leftarrow o$,

$$\mathsf{zip}.F.GH$$

$$= \quad \{ \quad \text{definition } \mathsf{zip}.F,\ (GH)_n = G_n H \ \}$$

$$\tau\Delta_n \mathsf{zip}.F.G_n H$$

$$= \quad \{ \quad G_n H \text{ single-valued, assumption: (5.23c) } \}$$

$$\tau\Delta_n((G_n{}^m)(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G_n)_{H^k})$$

$$= \quad \{ \quad \tau\Delta_n \text{ functor } \}$$

$$\tau\Delta_n((G_n{}^m)(\mathsf{zip}.F.H)) \cdot \tau\Delta_n(\mathsf{zip}.F.G_n)_{H^k}$$

$$= \quad \{ \quad \tau\Delta_n(G_n{}^m) = G^m, \text{ definition } \mathsf{zip}.F \ \}$$

$$(G^m)(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_{H^k}$$

Finally, requirement (5.23d) holds by assumption since Proj is a single-valued relator.
□

The importance of lemma (5.25) is that for the construction of $\mathsf{zip}.F.G$ we can restrict our attention to the cases that G is single-valued. In the next section we show that $\mathsf{zip}.F.G$ is also coherent with tupling in the first argument. Hence, combining both coherence properties it follows that an arbitrary $\mathsf{zip}.F.G$ can be defined in terms of the collection of half-zips $\mathsf{zip}.F_m.G_n$, half-zips both arguments of which are single-valued relators.

Commuting relators Just as for the generalisation for single-valued relators, we have to extend the definition of commuting relators. From arity considerations, we derive that $\tau(\mathsf{zip}.G.F)_\tau^\circ$ has the same arity as $\mathsf{zip}.F.G$. Hence, we define,

Definition 5.26 The half-zip $\mathsf{zip}.F.G$ for arbitrary relators F and G, is said to be a *zip* of (F, G) if there exists a half-zip $\mathsf{zip}.G.F$ such that

$$\mathsf{zip}.F.G = \tau(\mathsf{zip}.G.F)_\tau^\circ$$

□

Fans of single-valued relators Although we claimed that results for arbitrary relators can be derived from the endo case, one has to be careful when translating results which mention the fan. The fan of an endorelator F is a natural transformation of type $F \hookleftarrow \mathsf{Id}$. The fan of an arbitrary relator $F : 1 \leftarrow k$ has type $F\Delta_k \hookleftarrow \mathsf{Id}$ since the fan of a non-endo relator is by definition the fan of the endorelator $F\Delta_k$. For instance, from the generalized version of lemma (5.11) it follows that for $F : 1 \leftarrow k$ and $G : 1 \leftarrow l$,

$$(\mathsf{fan}.G)_F = \mathsf{zip}.F.G\Delta_l \cdot F(\mathsf{fan}.G)^k$$

Equivalently, since from (5.23c) and (5.24) it follows that

$$\mathsf{zip.F.G}\Delta_l \;=\; \mathsf{G}\Delta_l(\mathsf{zip.F.Id}) \;\cdot\; (\mathsf{zip.F.G})_{(\Delta_l)^k} \;=\; (\mathsf{zip.F.G})_{(\Delta_l)^k} \quad ,$$

we have,

$$(\mathsf{fan.G})_{\mathsf{F}} = (\mathsf{zip.F.G})_{(\Delta_l)^k} \cdot \mathsf{F}(\mathsf{fan.G})^k \tag{5.27}$$

Fortunately, the results for single-valued relators which mention the fan-function are still derived by adding "$\_^k$" or "$^l\_$" at the right place since a fan-function of a relator has the same type as the relator. However, one has to remember that a fan-function is a partial function: the fan-function $\hat{\mathsf{F}}$ for relator F of type $1 \leftarrow k$ is only defined for a vector of relations which all have the same source. For instance, for arbitrary relators the translation of property (5.12), i.e.

$$\hat{\mathsf{G}}\mathsf{F}R \;=\; \mathsf{zip.F.G} \;\cdot\; \mathsf{F}\hat{\mathsf{G}}R \quad \text{for each } R : A \leftarrow B$$

and its proof, reads for $\mathsf{F} : 1 \leftarrow k$ and $\mathsf{G} : 1 \leftarrow l$,

$$\hat{\mathsf{G}}(^l\mathsf{F})R \;=\; (\mathsf{zip.F.G})_A \;\cdot\; \mathsf{F}(\hat{\mathsf{G}}^k)R \quad \text{for each } R \;:\; A \leftarrow ((\Delta_l)^k)B$$

since,

$$\begin{array}{cl}
 & (\mathsf{zip.F.G})_A \;\cdot\; \mathsf{F}(\hat{\mathsf{G}}^k)R \\
= & \{ \quad \text{definition } \hat{\mathsf{G}},\ _^k \text{ functor} \quad \} \\
 & (\mathsf{zip.F.G})_A \cdot \mathsf{F}((\mathsf{G}^k)R \cdot (\mathsf{fan.G})^k{}_B) \\
= & \{ \quad \text{naturality } \mathsf{zip.F.G} \quad \} \\
 & \mathsf{G}(^l\mathsf{F})R \;\cdot\; (\mathsf{zip.F.G})_{((\Delta_l)^k)B} \;\cdot\; \mathsf{F}(\mathsf{fan.G})^k{}_B \\
= & \{ \quad \text{property (5.27)} \quad \} \\
 & \mathsf{G}(^l\mathsf{F})R \;\cdot\; (\mathsf{fan.G})_{\mathsf{F}B} \\
= & \{ \quad \text{definition } \hat{\mathsf{G}} \quad \} \\
 & \hat{\mathsf{G}}(^l\mathsf{F})R
\end{array}$$

5.5 The zip-a-dee-doo-dah theorem

We are now ready to formulate a general theorem about the existence of zips. We call the theorem the zip-a-dee-doo-dah theorem after the song "Zip-a-dee-doo-dah, zip-a-dee-ay" because it seemed such a wonderful day when the theorem was first discovered (at least to us anyway)! The inspiration for the theorem came from Fokkinga's Ph.D. thesis [16, chapter 4, p 90 onwards]. Fokkinga considers only one particular case — the construction of a function that transposes a stream of lists into a list of streams — but it was clear from his calculations that they could be made much more general if the step were taken to a higher level of abstraction.

Theorem 5.28 (Zip-a-dee-doo-dah) For all regular relators F, there are $\mathsf{zip.F.G}$, for each relator G with membership, such that the collection $\mathsf{zip.F.G}$ are half-zips. Furthermore, the regular relators are commuting, i.e. if G is also a regular relator then $\mathsf{zip.F.G} = \tau(\mathsf{zip.G.F})^{\circ}_{\tau}$.
□

As we mentioned before, we show how to construct $\mathsf{zip}.F.G$ for each regular relator F and an arbitrary relator G. This construction is inductive over the structure of the regular relators. We first construct "candidate" half-zips. The construction of "candidate" half-zips is dictated most of the time by the desire to meet one of the properties which followed from the fact that $(F\,,G)$ should commute. For instance, for the construction of the candidates zip for the identity and projections relators, product and coproduct, we use the derived coslok property for commuting relators $(F\,,G)$, for $F : 1 \leftarrow k$ and $G : 1 \leftarrow l$,

$$G({}^{l}\check{F})R \cdot \mathsf{zip}.F.G_B = \check{F}(G^k)R \quad \text{for each } R : A \leftarrow B. \tag{5.29}$$

For the construction of a zip of a composition, we aim to meet the requirement that $(FG\,,H)$ should commute. For the construction of a candidate zip for a tree type, we use property (5.15b). Subsequently, we verify that the construction we give satisfies the clauses in the definition of half-zips.

After the construction of the half-zips $\mathsf{zip}.F.G$ for all regular relators F, we verify that the regular relators are commuting.

Recoverable relator If a relator $F : 1 \leftarrow k$ is corecoverable from its cofan-function with β, i.e. $\beta_A \;:\; \Delta_k FA \leftarrow A$ and $\check{F}\beta = \mathsf{id}_F$, many proofs for the verification of the Zip-a-dee-doo-dah theorem with respect to $\mathsf{zip}.F$ follow for free. Recall the uniqueness property (3.21), that is to say, if $\alpha : F \xleftarrow{\mathsf{CoSlok}} G$ for mappings F and G, and the functor F' is corecoverable from F with β then $\alpha = G\beta$. Hence, it follows that the only possible candidate for $\mathsf{zip}.F.G$, for $F : 1 \leftarrow k$ and $G : 1 \leftarrow l$, which can satisfy equation (5.29), i.e. has type $G({}^{l}\check{F}) \xleftarrow{\mathsf{CoSlok}} \check{F}(G^k)$, is $\check{F}(G^k)({}^{l}\beta)$ since $G({}^{l}F)$ is corecoverable from $G({}^{l}\check{F})$ with ${}^{l}\beta$:

$$G({}^{l}\check{F})({}^{l}\beta) = G({}^{l}(\check{F}\beta)) = G({}^{l}(\mathsf{id}_F)) = \mathsf{id}_{G({}^{l}F)} \;.$$

Note that $({}^{l}\beta)_A$ has type $\Delta_k({}^{l}F)A \leftarrow A$, for each A, since ${}^{l}(\Delta_k F) = \Delta_k({}^{l}F)$ for single-valued F.

If the candidate $\check{F}(G^k)({}^{l}\beta)$ has indeed the coslok property (5.29) then the naturality of $\mathsf{zip}.F.G$ follows directly from lemma (3.22) since relator $F(G^k)$ is cofuseable into $\check{F}(G^k)$ since a relator is always cofuseable into its cofan function.

Furthermore, from type considerations it follows that $\mathsf{zip}.F$ is compositional. Since the category CoSlok obeys the same typing rules as the functor category Fun, it follows that for $G : 1 \leftarrow l$ and $H : l \leftarrow m$

$$G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_{H^k} \;:\; GH({}^{m}\check{F}) \xleftarrow{\mathsf{CoSlok}} \check{F}(GH)^k$$

but $\mathsf{zip}.F.GH$ is the unique arrow with the coslok property $GH({}^{m}\check{F}) \leftarrow \check{F}(GH)^k$, hence,

$$\mathsf{zip}.F.GH = G(\mathsf{zip}.F.H) \cdot (\mathsf{zip}.F.G)_{H^k} \;.$$

Similarly, we have $\mathsf{zip}.F.\mathsf{Proj} = \mathsf{id}_{F(\mathsf{Proj}^k)}$ since for $\mathsf{Proj} : 1 \leftarrow l$

$$\mathsf{zip}.F.\mathsf{Proj} = \check{F}(\mathsf{Proj}^k)({}^{l}\beta) = \check{F}\beta(\mathsf{Proj}^k) = \mathsf{id}_{F(\mathsf{Proj}^k)} \;.$$

From the naturality of α it follows that the candidate $\check{F}(G^k)({}^l\beta)$ is higher-order natural, i.e.

$$\alpha_{({}^lF)} \cdot \check{F}(H^k)({}^l\beta) \subseteq \check{F}(G^k)({}^l\beta) \cdot F(\alpha^k) \quad \text{for each } \alpha : G \hookleftarrow H \text{ and } G\,, H : 1 \leftarrow l.$$

For the cofan-function we have that $R \cdot \check{F}S \subseteq \check{F}(\Delta_k(R) \cdot S)$ since $(\mathsf{fan}.F)^\circ$ is a natural transformation of type $Id \hookrightarrow F\Delta_k$. Hence,

$$\begin{array}{cl} & \alpha_{({}^lF)} \cdot \check{F}(H^k)({}^l\beta) \\ = & \{ \quad \text{remark above, } \Delta_k(\alpha({}^lF)) = (\alpha^k)\Delta_k({}^lF) \quad \} \\ & \check{F}({\alpha^k}_{\Delta_k({}^lF)} \cdot (H^k)({}^l\beta)) \\ \subseteq & \{ \quad \alpha^k : G^k \hookleftarrow H^k,\ ({}^l\beta)_A : \Delta_k({}^lF)A \leftarrow A \quad \} \\ & \check{F}((G^k)({}^l\beta) \cdot \alpha^k) \\ = & \{ \quad \check{F}\text{–}F \text{ cofusion} \quad \} \\ & \check{F}(G^k)({}^l\beta) \cdot F(\alpha^k) \end{array}$$

Finally, it follows that $(F\,, G)$ for single-valued relator G commute. From the slok property of half-zips (5.12) it follows that $(\mathsf{zip}.G.F)^\circ_\tau$ has the coslok property $G({}^m\check{F}) \xleftarrow{\mathsf{CoSlok}} \check{F}(G^k)$ too. Hence, from the uniqueness property (3.21) it follows that

$$\mathsf{zip}.F.G = (\mathsf{zip}.G.F)^\circ_\tau \ .$$

Collecting all the results together, we have the following lemma

Lemma 5.30 If relator F is corecoverable from $\check{F}$ with β the only possible candidate for $\mathsf{zip}.F.G$, for each single-valued relator G, is $\check{F}(G^k)({}^l\beta)$. If the candidate half-zips $\check{F}(G^k)({}^l\beta)$ satisfy the coslok properties $G({}^m\check{F}) \xleftarrow{\mathsf{CoSlok}} \check{F}(G^k)$, for each G, then the collection of natural transformations $\check{F}(G^k)({}^l\beta)$ are half-zips of F. Furthermore, if the half-zip $\mathsf{zip}.G.F$ exists then F and G commute, i.e. $\mathsf{zip}.F.G = (\mathsf{zip}.G.F)^\circ_\tau$.
□

5.6 Constructing candidate zips

We are ready to embark on the task of constructing $\mathsf{zip}.F.G$ for regular relator F and arbitrary relator G. Note that we can restrict ourselves to single-valued relators G; from lemma (5.24) it follows that the $\mathsf{zip}.F.G$ for an arbitrary relator $G : m \leftarrow l$ can be constructed in terms of the collection $\mathsf{zip}.F.G_m$. Hence, in the remainder of this section we only consider single-valued relators G.

The zips we construct are, at this point in time, *candidate* zips. In the next section we vet them against the clauses in the specification of a zip.

The construction is by induction on the structure of the regular relators. That is, we show how to construct $\mathsf{zip}.Id.G$, $\mathsf{zip}.Proj.G$, $\mathsf{zip}.K_A.G$, $\mathsf{zip}.{+}.G$ and $\mathsf{zip}.{\times}.G$ for single-valued relator G. We show how to construct $\mathsf{zip}.FG.H$ by composing the zips $\mathsf{zip}.F.H$ and $\mathsf{zip}.G.H$. And we show how to construct $\mathsf{zip}.\Delta_m F_m.G$ in terms of the collection of zips $\mathsf{zip}.F_m.G$. Finally, we construct $\mathsf{zip}.T.G$ for relator T from a tree type induced by the binary relator $\otimes$ given that $\mathsf{zip}.{\otimes}.G$ has already been constructed.

5.6.1 Zipping identity, projections, coproduct and product

Recall that the fan-functions of the projections — which includes the identity relator —, coproduct and product are recoverable from their corresponding membership relation. Or, dually, for relator F being a projection, coproduct or product, we have that F is corecoverable from the cofan function $\check{\mathsf{F}}$ with $\beta \triangleq (\mathsf{mem.F})^\circ$ since $\check{\mathsf{F}}\beta = \mathsf{id}_\mathsf{F}$. This means that we can use lemma (5.30) to construct the candidate zips for the projections, coproduct and product. Recall that $\mathtt{Proj}_i : 1 \leftarrow k$ has membership relation $\Delta_k(i = k \rightarrow \mathsf{id}_{\mathtt{Proj}_i}, \perp\!\!\perp_{\mathtt{Proj}_k, \mathtt{Proj}_i})$, and coproduct and product have membership relations $\langle \mathsf{inl}^\circ, \mathsf{inr}^\circ \rangle$ and $\langle \mathsf{outl}, \mathsf{outr} \rangle$, respectively. So, lemma (5.30) gives us the following candidate zips, for $\mathsf{G} : 1 \leftarrow l$,

$$
\begin{aligned}
\mathsf{zip}.\mathtt{Proj}_i.\mathsf{G} &= \mathtt{Proj}_i(\mathsf{G}^k)({}^l(\Delta_k(i = k \rightarrow \mathsf{id}_{\mathtt{Proj}_i}, \perp\!\!\perp_{\mathtt{Proj}_k, \mathtt{Proj}_i})))\\
\mathsf{zip}.+.\mathsf{G} &= (\triangledown)(\mathsf{G}^2)({}^l\langle \mathsf{inl}, \mathsf{inr} \rangle)\\
\mathsf{zip}.\times.\mathsf{G} &= (\blacktriangle)(\mathsf{G}^2)({}^l\langle \mathsf{outl}^\circ, \mathsf{outr}^\circ \rangle)
\end{aligned}
$$

The candidate for $\mathsf{zip}.\mathtt{Proj}_i.\mathsf{G}$ can be simplified,

$$
\begin{aligned}
& \mathtt{Proj}_i(\mathsf{G}^k)({}^l(\Delta_k(i = k \rightarrow \mathsf{id}_{\mathtt{Proj}_i}, \perp\!\!\perp_{\mathtt{Proj}_k, \mathtt{Proj}_i})))\\
= & \qquad \{ \quad \mathtt{Proj}_i(\mathsf{G}^k) = \mathsf{G}({}^l\mathtt{Proj}_i), \text{“}{}^l\_\text{” functor} \quad \}\\
& \mathsf{G}({}^l(\mathtt{Proj}_i\Delta_k(i = k \rightarrow \mathsf{id}_{\mathtt{Proj}_i}, \perp\!\!\perp_{\mathtt{Proj}_k, \mathtt{Proj}_i})))\\
= & \qquad \{ \quad \mathtt{Proj}_i\Delta_k R_k = R_i \quad \}\\
& \mathsf{G}({}^l(\mathsf{id}_{\mathtt{Proj}_i}))\\
= & \qquad \{ \quad \text{identities, } \mathsf{G}({}^l\mathtt{Proj}) = \mathtt{Proj}(\mathsf{G}^k) \quad \}\\
& \mathsf{id}_{\mathtt{Proj}_i(\mathsf{G}^k)}
\end{aligned}
$$

The candidates zips for coproduct and product, we can simplify too,

$$
(\triangledown)(\mathsf{G}^2)({}^l\langle \mathsf{inl}, \mathsf{inr} \rangle) = (\triangledown)(\mathsf{G}^2)\langle {}^l\mathsf{inl}, {}^l\mathsf{inr} \rangle = \mathsf{G}({}^l\mathsf{inl}) \triangledown \mathsf{G}({}^l\mathsf{inr})
$$

and similarly

$$
(\blacktriangle)(\mathsf{G}^2)({}^l\langle \mathsf{outl}^\circ, \mathsf{outr}^\circ \rangle) = \mathsf{G}({}^l\mathsf{outl}^\circ) \blacktriangle \mathsf{G}({}^l\mathsf{outr}^\circ) \ .
$$

Note that ${}^l\mathsf{inl}, {}^l\mathsf{inr} : \mathcal{C}^l \leftarrow \mathcal{C}^l \times \mathcal{C}^l$ are injections in the allegory $\mathcal{C}^l$. We take $({}^n\mathsf{inl}, {}^n\mathsf{inr})$ as the canonical injections for allegory $\mathcal{C}^n$, for all n. That is to say, we define coproduct, and similarly product, component wise in $\mathcal{C}^n$. So, we define $\mathsf{inl} \triangleq {}^n\mathsf{inl} : \mathcal{C}^n \leftarrow \mathcal{C}^n \times \mathcal{C}^n$, $\mathsf{inr} \triangleq {}^n\mathsf{inr} : \mathcal{C}^n \leftarrow \mathcal{C}^n \times \mathcal{C}^n$ and $\triangledown \triangleq {}^n(\triangledown) : \mathcal{C}^n \leftarrow \mathcal{C}^n \times \mathcal{C}^n$, and similarly for product.

Hence, we have

$$
\begin{aligned}
\mathsf{zip}.+.\mathsf{G} &= \mathsf{Ginl} \vartriangle \mathsf{Ginr} && (5.31)\\
\mathsf{zip}.\times.\mathsf{G} &= \mathsf{Goutl}^\circ \blacktriangle \mathsf{Goutr}^\circ && (5.32)
\end{aligned}
$$

where $(\mathsf{inl}, \mathsf{inr})$ and $(\mathsf{outl}, \mathsf{outr})$ denote the injections and projections, respectively, in the allegory of the source of relator G.

For the above candidates we have to verify the coslok properties, for each $R : \Delta_k B \leftarrow A$,

$$G({}^l\mathsf{Proj}_i)R \cdot \mathsf{id}_{\mathsf{Proj}_i(G^k)A} = \mathsf{Proj}_i(G^k)R \tag{5.33}$$

and, for each $R : C \leftarrow A$, $S : C \leftarrow B$,

$$G(R \triangledown S) \cdot \mathsf{Ginl}_{A,B} \triangledown \mathsf{Ginr}_{A,B} \;=\; GR \triangledown GS \tag{5.34}$$

$$G(R \blacktriangle S) \cdot \mathsf{Goutl}_{A,B} \blacktriangle \mathsf{Goutr}_{A,B} \;=\; GR \blacktriangle GS \tag{5.35}$$

Property (5.33) follows from $G({}^l\mathsf{Proj}_i) = \mathsf{Proj}_i(G^k)$. Verification of (5.34) is straightforward since relational coproduct is the real categorical coproduct. Instead of (5.35) we verify the equivalent for $R : A \leftarrow C$ and $S : B \leftarrow C$,

$$\mathsf{Goutl}_{A,B} \vartriangle \mathsf{Goutr}_{A,B} \cdot G(R \vartriangle S) \;=\; GR \vartriangle GS \tag{5.36}$$

Note that (5.36) is the dual of (5.34) if we restrict R and S to functions since product is the real categorical product in the sub-category Map. For arbitrary relations R and S we calculate:

$$\begin{array}{ll}
 & \mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(R \vartriangle S) \\
= & \quad \{ \quad \text{product} \quad \} \\
 & \mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(R \times \mathsf{id}) \cdot G(\mathsf{id} \times S) \cdot G(\mathsf{id} \vartriangle \mathsf{id}) \\
= & \quad \{ \quad \text{claim: } \mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(R \times \mathsf{id}) = GR \times G\mathsf{id} \cdot \mathsf{Goutl} \vartriangle \mathsf{Goutr}, \\
 & \quad\quad\;\; \text{and } \mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(\mathsf{id} \times S) = G\mathsf{id} \times GS \cdot \mathsf{Goutl} \vartriangle \mathsf{Goutr} \quad \} \\
 & GR \times G\mathsf{id} \cdot G\mathsf{id} \times GS \cdot \mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(\mathsf{id} \vartriangle \mathsf{id}) \\
= & \quad \{ \quad \text{see remark above, id function, (5.36)} \quad \} \\
 & GR \times G\mathsf{id} \cdot G\mathsf{id} \times GS \cdot G\mathsf{id} \vartriangle G\mathsf{id} \\
= & \quad \{ \quad \text{product} \quad \} \\
 & GR \vartriangle GS
\end{array}$$

Next we verify the claim. It is straightforward to prove that

$$\mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(R \times \mathsf{id}) \subseteq GR \times G\mathsf{id} \cdot \mathsf{Goutl} \vartriangle \mathsf{Goutr}$$

For the other inclusion we calculate

$$\begin{array}{ll}
 & GR \times G\mathsf{id} \cdot \mathsf{Goutl} \vartriangle \mathsf{Goutr} \\
= & \quad \{ \quad \text{product-split fusion} \quad \} \\
 & (GR \cdot \mathsf{Goutl}) \vartriangle \mathsf{Goutr} \\
= & \quad \{ \quad \text{G functor, computation } \mathsf{outl} \quad \} \\
 & (\mathsf{Goutl} \cdot G(R \times \mathsf{id})) \vartriangle \mathsf{Goutr} \\
\subseteq & \quad \{ \quad \text{modulaw law} \quad \} \\
 & \mathsf{Goutl} \vartriangle (\mathsf{Goutr} \cdot G(R^\circ \times \mathsf{id})) \cdot G(R \times \mathsf{id}) \\
\subseteq & \quad \{ \quad \text{computation outr: } \mathsf{Goutr} \cdot G(R^\circ \times \mathsf{id}) \subseteq \mathsf{Goutr} \quad \} \\
 & \mathsf{Goutl} \vartriangle \mathsf{Goutr} \cdot G(R \times \mathsf{id})
\end{array}$$

Dually follows,

$$\mathsf{Goutl} \vartriangle \mathsf{Goutr} \;\cdot\; \mathsf{G}(\mathsf{id} \times \mathsf{S}) \;=\; \mathsf{Gid} \times \mathsf{GS} \;\cdot\; \mathsf{Goutl} \vartriangle \mathsf{Goutr}$$

So, indeed the candidates suggested by the uniqueness property (3.11) have the corresponding coslok properties. Hence, from lemma (5.30) it follows that the constructed candidates are half-zips.

Next we construct candidate zips for the constant relator, composition of relators and the relator of a tree type.

5.6.2 Zipping constants

For the construction of the candidate zip $\mathsf{zip}.\mathsf{K}_\mathsf{A}.\mathsf{G}$ for the constant relator $\mathsf{K}_\mathsf{A} : 1 \leftarrow k$, for some A, we concentrate on the derived slok property (5.27), for $\mathsf{G} : 1 \leftarrow l$

$$(\mathsf{fan}.\mathsf{G})_{\mathsf{K}_\mathsf{A}} = (\mathsf{zip}.\mathsf{K}_\mathsf{A}.\mathsf{G})_{(\Delta_l)^k} \cdot \mathsf{K}_\mathsf{A}(\mathsf{fan}.\mathsf{G})^k$$

But since $\mathsf{K}_\mathsf{A}(\mathsf{fan}.\mathsf{G})^k = \mathsf{id}_{\mathsf{K}_\mathsf{A}}$ it follows that

$$(\mathsf{fan}.\mathsf{G})_{\mathsf{K}_\mathsf{A}} = (\mathsf{zip}.\mathsf{K}_\mathsf{A}.\mathsf{G})_{(\Delta_l)^k}$$

This suggests that we should take

$$\mathsf{zip}.\mathsf{K}_\mathsf{A}.\mathsf{G} = (\mathsf{fan}.\mathsf{G})_{\mathsf{K}_\mathsf{A}} \tag{5.37}$$

as the candidate for $\mathsf{zip}.\mathsf{K}_\mathsf{A}.\mathsf{G}$. Note that this choice depends on the fact that there exists a fan for relator G.

5.6.3 Zipping compositions

For the construction of the candidate $\mathsf{zip}.\mathsf{FG}.\mathsf{H}$, for $\mathsf{F} : m \leftarrow k$, $\mathsf{G} : k \leftarrow l$, $\mathsf{H} : n \leftarrow o$, we let ourselves be guide by the requirement that $(\mathsf{FG}\,,\mathsf{H})$ should commute, i.e.

$$\mathsf{zip}.\mathsf{FG}.\mathsf{H} = \tau(\mathsf{zip}.\mathsf{H}.\mathsf{FG})^\circ_\tau$$

Assuming compositionality of $\mathsf{zip}.\mathsf{H}$ and commutativity of $(\mathsf{F}\,,\mathsf{H})$ and $(\mathsf{G}\,,\mathsf{H})$, we calculate

$$\begin{array}{ll}
 & \tau(\mathsf{zip}.\mathsf{H}.\mathsf{FG})^\circ_\tau \\
\equiv & \{\quad \mathsf{zip}.\mathsf{H} \text{ compositional, converse, } \tau \text{ functor} \quad\} \\
 & \tau(\mathsf{zip}.\mathsf{H}.\mathsf{F})^\circ_{(\mathsf{G}^\circ)\tau} \cdot \tau(\mathsf{F}^n)(\mathsf{zip}.\mathsf{H}.\mathsf{G})^\circ_\tau \\
\equiv & \{\quad (\mathsf{F}\,,\mathsf{H}) \text{ and } (\mathsf{G}\,,\mathsf{H}) \text{ are commuting, } \tau \text{ isomorphism} \quad\} \\
 & (\mathsf{zip}.\mathsf{F}.\mathsf{H})_{\tau(\mathsf{G}^\circ)\tau} \cdot \tau(\mathsf{F}^n)\tau(\mathsf{zip}.\mathsf{G}.\mathsf{H}) \\
\equiv & \{\quad \tau(\mathsf{G}^\circ)\tau = {}^\circ\mathsf{G}, \text{ same for } \mathsf{F} \quad\} \\
 & (\mathsf{zip}.\mathsf{F}.\mathsf{H})_{{}^\circ\mathsf{G}} \cdot ({}^n\mathsf{F})(\mathsf{zip}.\mathsf{G}.\mathsf{H})
\end{array}$$

Hence, we take as candidate for the half-zip of FG:

$$\mathsf{zip}.\mathsf{FG}.\mathsf{H} = (\mathsf{zip}.\mathsf{F}.\mathsf{H})_{{}^\circ\mathsf{G}} \cdot ({}^n\mathsf{F})(\mathsf{zip}.\mathsf{G}.\mathsf{H}) \tag{5.38}$$

Later on we exploit the fact that this derivation was based on the requirement of commutativity of $(\mathsf{FG}\,,\mathsf{H})$.

5.6.4 Zipping tuples

Just as we proved that from the properties (5.23c) and (5.23d) of the definition of a half-zip that $\mathsf{zip}.\mathsf{F}.\mathsf{G}$ is coherent with tupling in its second argument (lemma (5.24)), it follows from the definitions of the candidates zips $\mathsf{zip}.\mathsf{FG}$ and $\mathsf{zip}.\mathsf{Proj}$ that $\mathsf{zip}.\mathsf{F}.\mathsf{G}$ is coherent with tupling in the first argument. That is, we have for $\mathsf{F} : \mathfrak{m} \leftarrow \mathfrak{k}$ and $\mathsf{G} : \mathfrak{n} \leftarrow \mathfrak{l}$,

$$(\mathsf{zip}.\mathsf{F}.\mathsf{G} =)\ \mathsf{zip}.\Delta_{\mathfrak{m}}\mathsf{F}_{\mathfrak{m}}.\mathsf{G} = \Delta_{\mathfrak{m}}(\mathsf{zip}.\mathsf{F}_{\mathfrak{m}}.\mathsf{G})\ . \tag{5.39}$$

5.6.5 Zipping tree types

The final stage in this analysis of zips is their construction for relators of tree types. Let $\otimes$ be a binary relator and let $(\mathsf{in}\,,\mathsf{T})$ be its tree type. We consider the construction of $\mathsf{zip}.\mathsf{T}.\mathsf{G}$ for arbitrary relator G. The construction is inductive in the sense that we assume the existence of $\mathsf{zip}.\otimes.\mathsf{G}$ and construct $\mathsf{zip}.\mathsf{T}.\mathsf{G}$ in terms of it.

This time we let ourselves be guided by the generalized version of the derived property (5.15b) for commuting relators, for $\mathsf{F} : 1 \leftarrow \mathfrak{k}$ and $\mathsf{G}\,,\mathsf{H} : 1 \leftarrow \mathfrak{l}$,

$$\mathsf{F}({}^{\mathfrak{k}}\alpha) \cdot \mathsf{zip}.\mathsf{H}.\mathsf{F} = \mathsf{zip}.\mathsf{G}.\mathsf{F} \cdot \alpha_{\mathsf{F}^{\mathfrak{l}}} \quad \text{for function } \alpha,\ \alpha : \mathsf{G} \leftarrow \mathsf{H}.$$

Instantiating α with $\mathsf{in}_{\mathsf{A}}\ :\ \mathsf{TA} \leftarrow \mathsf{A} \otimes \mathsf{TA}$ and $\mathsf{F} := \mathsf{G}$ gives, for $\mathsf{G} : 1 \leftarrow \mathfrak{k}$

$$
\begin{array}{ll}
 & \mathsf{G}({}^{\mathfrak{k}}\mathsf{in}) \cdot \mathsf{zip}.\mathrm{Id}{\otimes}\mathsf{T}.\mathsf{G} = \mathsf{zip}.\mathsf{T}.\mathsf{G} \cdot \mathsf{in}_{\mathsf{G}} \\
\equiv & \qquad \{ \qquad \text{composition (5.38): } \mathrm{Id}{\otimes}\mathsf{T} = (\otimes)\langle \mathrm{Id}\,,\mathsf{T}\rangle \quad \} \\
 & \mathsf{G}({}^{\mathfrak{k}}\mathsf{in}) \ \cdot\ (\mathsf{zip}.\otimes.\mathsf{G})_{{}^{\mathfrak{k}}\langle \mathrm{Id}\,,\mathsf{T}\rangle} \cdot (\otimes)(\mathsf{zip}.\langle \mathrm{Id}\,,\mathsf{T}\rangle.\mathsf{G}) \ =\ \mathsf{zip}.\mathsf{T}.\mathsf{G} \cdot \mathsf{in}_{\mathsf{G}} \\
\equiv & \qquad \{ \qquad \text{definition (5.24), } \mathsf{zip}.\mathrm{Id}.\mathsf{G} = \mathsf{id}_{\mathsf{G}} \quad \} \\
 & \mathsf{G}({}^{\mathfrak{k}}\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{{}^{\mathfrak{k}}\langle \mathrm{Id}\,,\mathsf{T}\rangle} \cdot \mathsf{id}_{\mathsf{G}}{\otimes}\mathsf{zip}.\mathsf{T}.\mathsf{G} = \mathsf{zip}.\mathsf{T}.\mathsf{G} \cdot \mathsf{in}_{\mathsf{G}} \\
\equiv & \qquad \{ \qquad \text{unique extension} \quad \} \\
 & \mathsf{zip}.\mathsf{T}.\mathsf{G} = (\!|\mathsf{id}_{\mathsf{G}}\otimes;\ \mathsf{G}({}^{\mathfrak{k}}\mathsf{in}) \ \cdot\ (\mathsf{zip}.\otimes.\mathsf{G})_{{}^{\mathfrak{k}}\langle \mathrm{Id}\,,\mathsf{T}\rangle}|\!)
\end{array}
$$

So we define,

$$\mathsf{zip}.\mathsf{T}.\mathsf{G} = (\!|\mathsf{id}_{\mathsf{G}}\otimes;\ \mathsf{G}({}^{\mathfrak{k}}\mathsf{in}) \ \cdot\ (\mathsf{zip}.\otimes.\mathsf{G})_{{}^{\mathfrak{k}}\langle \mathrm{Id}\,,\mathsf{T}\rangle}|\!)\quad .$$

Later on we exploit the fact that this candidate was forced upon us. To be more precise, we use that for a given relator $\mathsf{G} : 1 \leftarrow \mathfrak{k}$ and tree type $(\mathsf{in}\,,\mathsf{T})$ induced by the binary relator $\otimes$ if we have a mapping f_{G} mapping relators to natural transformations such that

$$\mathsf{G}({}^{\mathfrak{k}}\mathsf{in}) \cdot \mathsf{f}_{\mathsf{G}}(\mathrm{Id}{\otimes}\mathsf{T}) = \mathsf{f}_{\mathsf{G}}(\mathsf{T}) \cdot \mathsf{in}_{\mathsf{G}} \tag{5.40}$$

and for $\mathsf{H} : 1 \leftarrow \mathfrak{m}$, $\mathrm{I} : \mathfrak{m} \leftarrow \mathfrak{n}$,

$$\mathsf{f}_{\mathsf{G}}(\mathsf{H}\mathrm{I}) = \mathsf{f}_{\mathsf{G}}(\mathsf{H})_{{}^{\mathfrak{k}}\mathrm{I}} \cdot \mathsf{H}(\mathsf{f}_{\mathsf{G}}(\mathrm{I}))\ \wedge\ \mathsf{f}_{\mathsf{G}}(\mathrm{I}) = \Delta_{\mathfrak{m}}(\mathsf{f}_{\mathsf{G}}(\mathrm{I}_{\mathfrak{m}}))\ \wedge\ \mathsf{f}_{\mathsf{G}}(\mathrm{Id}) = \mathsf{id}_{\mathsf{G}} \tag{5.41}$$

and

$$\mathsf{f}_{\mathsf{G}}(\otimes) = \mathsf{zip}.\otimes.\mathsf{G} \tag{5.42}$$

it follows that

$$\mathsf{f_G(T)} = (\!|\mathsf{id_G}\otimes;\ \mathsf{G}({}^{k}\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle}|\!)$$

that is to say, $\mathsf{f_G(T)} = \mathsf{zip.T.G}$. This fact follows from a generalisation of the above calculation. We start with property (5.40):

$$\begin{array}{ll}
& \mathsf{G}({}^{k}\mathsf{in}) \cdot \mathsf{f}(\mathrm{Id}\otimes\mathrm{T}) = \mathsf{f}(\mathrm{T}) \cdot \mathsf{in_G} \\
\equiv & \quad \{ \quad \text{property (5.41) with } \mathrm{H} := \otimes \text{ and } \mathrm{I} := \langle \mathrm{Id}\,,\mathrm{T}\rangle \quad \} \\
& \mathsf{G}({}^{k}\mathsf{in}) \ \cdot\ \mathsf{f}(\otimes)_{k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \ \cdot\ (\otimes)\mathsf{f}(\langle \mathrm{Id}\,,\mathrm{T}\rangle) \ =\ \mathsf{f}(\mathrm{T}) \cdot \mathsf{in_G} \\
\equiv & \quad \{ \quad \text{property (5.41): } \mathsf{f}(\langle \mathrm{Id}\,,\mathrm{T}\rangle) = \langle \mathsf{f}(\mathrm{Id})\,,\mathsf{f}(\mathrm{T})\rangle = \langle \mathsf{id_G}\,,\mathsf{f}(\mathrm{T})\rangle \quad \} \\
& \mathsf{G}({}^{k}\mathsf{in}) \ \cdot\ \mathsf{f}(\otimes)_{k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \ \cdot\ \mathsf{id_G} \otimes \mathsf{f}(\mathrm{T}) \ =\ \mathsf{f}(\mathrm{T}) \cdot \mathsf{in_G} \\
\equiv & \quad \{ \quad \text{unique extension} \quad \} \\
& \mathsf{f}(\mathrm{T}) = (\!|\mathsf{id_G}\otimes;\ \mathsf{G}({}^{k}\mathsf{in}) \ \cdot\ \mathsf{f}(\otimes)_{k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle}|\!) \\
\equiv & \quad \{ \quad \text{property (5.42)} \quad \} \\
& \mathsf{f}(\mathrm{T}) = (\!|\mathsf{id_G}\otimes;\ \mathsf{G}({}^{k}\mathsf{in}) \ \cdot\ (\mathsf{zip}.\otimes.\mathsf{G})_{k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle}|\!)
\end{array}$$

5.6.6 Summary

For later use we summarise the zips so far constructed.

$$\begin{array}{rcll}
\mathsf{zip}.\mathrm{Id}.\mathsf{G} & = & \mathsf{id_G} & (5.43) \\
\mathsf{zip}.\mathrm{Proj}.\mathsf{G} & = & \mathsf{id}_{\mathrm{Proj}(\mathsf{G}^k)} \quad \text{for } \mathrm{Proj} : 1 \leftarrow k & (5.44) \\
\mathsf{zip}.+.\mathsf{G} & = & \mathsf{Ginl} \triangledown \mathsf{Ginr} & (5.45) \\
\mathsf{zip}.\times.\mathsf{G} & = & \mathsf{Goutl}^{\circ} \blacktriangle \mathsf{Goutr}^{\circ} & (5.46) \\
\mathsf{zip}.\mathrm{K_A}.\mathsf{G} & = & (\mathsf{fan}.\mathsf{G})_{\mathrm{K_A}} & (5.47) \\
\mathsf{zip}.\mathrm{FG}.\mathrm{H} & = & (\mathsf{zip}.\mathrm{F}.\mathrm{H})_{l\mathrm{G}} \cdot \mathrm{F}(\mathsf{zip}.\mathrm{G}.\mathrm{H}) \quad \text{for } \mathrm{H} : 1 \leftarrow l & (5.48) \\
\mathsf{zip}.\Delta_m\mathrm{F}_m.\mathsf{G} & = & \Delta_m(\mathsf{zip}.\mathrm{F}_m.\mathsf{G}) \quad \text{for } \mathrm{F} : m \leftarrow k & (5.49) \\
\mathsf{zip}.\mathrm{T}.\mathsf{G} & = & (\!|\mathsf{id_G}\otimes;\ \mathsf{G}({}^{l}\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{l\langle \mathrm{Id}\,,\,\mathrm{T}\rangle}|\!) \quad \text{for } \mathsf{G} : 1 \leftarrow l & (5.50)
\end{array}$$

5.7 Vetting the candidates

As forewarned, the clauses in the specification of half-zips were postponed until after the construction of suitable candidates. In this section we verify that they are indeed satisfied in each case. Many verifications are entirely straightforward. From lemma (5.30) it follows that the candidate zips for the identity and projection relators, coproduct and product are already vetted since we have shown that the candidate zips indeed have coslok property (5.29). So, we only have to vet the candidate zips for the constant relator, composition of relators, tupling of relators and the relator of a tree type.

5.7.1 Zips are natural transformations

Verification of $\mathsf{zip}.\mathsf{K_A}.\mathsf{G} \,:\, \mathsf{GK_A} \leftarrow \mathsf{K_A G}$, i.e.

$$\mathsf{GK_A R} \cdot (\mathsf{fan}.\mathsf{G})_\mathsf{A} \;=\; (\mathsf{fan}.\mathsf{G})_\mathsf{A} \cdot \mathsf{K_A G R} \quad \text{for each } \mathsf{R} : \mathsf{B} \leftarrow \mathsf{C},$$

is trivial since $\mathsf{K_A X} = \mathsf{id_A}$. Naturality of $\mathsf{zip}.\mathsf{FG}.\mathsf{H}$ follows directly from the naturality of $\mathsf{zip}.\mathsf{F}.\mathsf{H}$ and $\mathsf{zip}.\mathsf{G}.\mathsf{H}$. Similarly, naturality of $\mathsf{zip}.\Delta_m \mathsf{F}_m.\mathsf{G}$ follows from naturality of each of the components $\mathsf{zip}.\mathsf{F}_m.\mathsf{G}$. To establish that naturality of $\mathsf{zip}.\mathsf{T}.\mathsf{G}$ follows from the naturality of $\mathsf{zip}.\otimes.\mathsf{G}$, we calculate, for $\mathsf{G} : 1 \leftarrow k$,

$$\begin{array}{ll}
& \mathsf{G}({}^k\mathsf{T})\mathsf{R} \cdot \mathsf{zip}.\mathsf{T}.\mathsf{G} \;=\; \mathsf{zip}.\mathsf{T}.\mathsf{G} \cdot \mathsf{TGR} \\
\equiv & \qquad \{ \qquad \text{definition } \mathsf{zip}.\mathsf{T}.\mathsf{G}\text{, tree type fusion} \quad \} \\
& \mathsf{G}({}^k\mathsf{T})\mathsf{R} \cdot \mathsf{zip}.\mathsf{T}.\mathsf{G} \;=\; (\!|\mathsf{id_G}\otimes;\; \mathsf{G}({}^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{{}^k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \cdot \mathsf{GR}\otimes\mathsf{id}|\!) \\
\Leftarrow & \qquad \{ \qquad \text{definition } \mathsf{zip}.\mathsf{T}.\mathsf{G}\text{, catamorphism fusion} \quad \} \\
& \quad \mathsf{G}({}^k\mathsf{T})\mathsf{R} \cdot \mathsf{G}({}^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{{}^k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \\
& = \mathsf{G}({}^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{{}^k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \cdot \mathsf{GR}\otimes\mathsf{id} \cdot \mathsf{id}\otimes\mathsf{G}({}^k\mathsf{T})\mathsf{R} \\
\equiv & \qquad \{ \qquad \mathsf{G},\ {}^k_,\ \otimes \text{ functors, } \mathsf{TR}_k \cdot \mathsf{in} \;=\; \mathsf{in} \cdot \mathsf{R}_k \otimes \mathsf{TR}_k \;=\; \mathsf{in} \cdot (\mathrm{Id}\otimes\mathsf{T})\mathsf{R}_k \quad \} \\
& \quad \mathsf{G}({}^k\mathsf{in}) \cdot \mathsf{G}({}^k(\mathrm{Id}\otimes\mathsf{T}))\mathsf{R} \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{{}^k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \\
& = \mathsf{G}({}^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{{}^k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \cdot \mathsf{GR}\otimes\mathsf{G}({}^k\mathsf{T})\mathsf{R} \\
\Leftarrow & \qquad \{ \qquad \text{definition naturality} \quad \} \\
& (\mathsf{zip}.\otimes.\mathsf{G})_{{}^k\langle \mathrm{Id}\,,\,\mathrm{T}\rangle} \,:\, \mathsf{G}({}^k(\mathrm{Id}\otimes\mathsf{T})) \leftarrow \mathsf{G}\otimes\mathsf{G}({}^k\mathsf{T}) \\
\Leftarrow & \qquad \{ \qquad \mathsf{G}({}^k(\mathrm{Id}\otimes\mathsf{T})) = \mathsf{G}({}^k\otimes)({}^k\langle \mathrm{Id}\,,\mathsf{T}\rangle) \\
& \qquad\qquad\quad \text{and } \mathsf{G}\otimes\mathsf{G}({}^k\mathsf{T}) = \otimes(\mathsf{G}^2)({}^k\langle \mathrm{Id}\,,\mathsf{T}\rangle) \quad \} \\
& \mathsf{zip}.\otimes.\mathsf{G} \,:\, \mathsf{G}({}^k\otimes) \leftarrow \otimes(\mathsf{G}^2)
\end{array}$$

5.7.2 Zips are higher-order natural

In this section we verify that the constructed half-zip are higher order natural. That is to say, we verify property (5.23b) of the definition of half-zips, for $\mathsf{F} : m \leftarrow k$ and $\mathsf{G}\,,\mathsf{H} : 1 \leftarrow l$,

$$(\alpha^m)_{({}^l\mathsf{F})} \cdot \mathsf{zip}.\mathsf{F}.\mathsf{H} \subseteq \mathsf{zip}.\mathsf{F}.\mathsf{G} \cdot \mathsf{F}(\alpha^k) \quad \text{for each } \alpha : \mathsf{G} \hookleftarrow \mathsf{H}. \tag{5.51}$$

For $\mathsf{zip}.\mathsf{K_A}$ we calculate:

$$\begin{array}{ll}
& \alpha_{({}^l(\mathsf{K_A}))} \cdot \mathsf{zip}.\mathsf{K_A}.\mathsf{H} \subseteq \mathsf{zip}.\mathsf{K_A}.\mathsf{G} \cdot \mathsf{K_A}\alpha \\
\equiv & \qquad \{ \qquad \text{lifting, } (\mathsf{zip}.\mathsf{K_A}.\mathsf{F})_\mathsf{B} = \mathsf{fan}.\mathsf{F_A},\ \mathsf{K_A X} = \mathsf{id_A} \quad \} \\
& \alpha_{\Delta \mathsf{A}} \cdot \mathsf{fan}.\mathsf{H_A} \subseteq \mathsf{fan}.\mathsf{G_A} \\
\equiv & \qquad \{ \qquad \alpha_\Delta \cdot \mathsf{fan}.\mathsf{H} \,:\, \mathsf{G}\Delta \hookleftarrow \mathrm{Id}\,, \\
& \qquad\qquad\quad \mathsf{fan}.\mathsf{G} \text{ is the largest natural transformation of this type} \quad \} \\
& \mathit{true}
\end{array}$$

For the higher-order naturality of $\mathsf{zip.FG}$, we assume that $\mathsf{zip.F}$ and $\mathsf{zip.G}$ satisfy property (5.51). Assume that $\alpha : \mathsf{H} \hookleftarrow \mathsf{I}$, then we calculate, for $\mathsf{F} : m \leftarrow l$, $\mathsf{G} : l \leftarrow k$, $\mathsf{H}, \mathsf{I} : 1 \leftarrow n$

$$
\begin{array}{cl}
 & \mathsf{zip.FG.H} \cdot \mathsf{FG}(\alpha^k) \\
= & \quad \{ \quad \text{definition } \mathsf{zip.FG} \quad \} \\
 & (\mathsf{zip.F.H})_{^n\mathsf{G}} \cdot \mathsf{F}(\mathsf{zip.G.H}) \cdot \mathsf{FG}(\alpha^k) \\
\supseteq & \quad \{ \quad \mathsf{F} \text{ relator, property (5.51): } (\alpha^l)_{^n\mathsf{G}} \cdot \mathsf{zip.G.I} \subseteq \mathsf{zip.G.H} \cdot \mathsf{G}(\alpha^k) \quad \} \\
 & (\mathsf{zip.F.H})_{^n\mathsf{G}} \cdot \mathsf{F}(\alpha^l)_{^n\mathsf{G}} \cdot \mathsf{Fzip.G.I} \\
\supseteq & \quad \{ \quad \text{lifting, property (5.51): } (\alpha^m)_{(^n\mathsf{F})} \cdot \mathsf{zip.F.I} \subseteq \mathsf{zip.F.H} \cdot \mathsf{F}(\alpha^l) \quad \} \\
 & (\alpha^m)_{(^n\mathsf{F})(^n\mathsf{G})} \cdot (\mathsf{zip.F.I})_{^n\mathsf{G}} \cdot \mathsf{Fzip.G.I} \\
= & \quad \{ \quad \text{“}{}^n\_\text{” functor, definition } \mathsf{zip.FG} \quad \} \\
 & (\alpha^m)_{^n(\mathsf{FG})} \cdot \mathsf{zip.FG.I}
\end{array}
$$

Verification of the higher-order naturality of $\mathsf{zip}.\Delta_m\mathsf{F}_m$ follows from a similar proof to that given for lemma (5.25). For the verification of the higher-order naturality of $\mathsf{zip.T}$ we assume property (5.23b) for $\mathsf{zip}.\otimes$. Then we calculate, for $\mathsf{G}, \mathsf{H} : 1 \leftarrow k$,

$$
\begin{array}{cl}
 & \alpha_{(^k\mathsf{T})} \cdot \mathsf{zip.T.H} \subseteq \mathsf{zip.T.G} \cdot \mathsf{T}\alpha \\
\equiv & \quad \{ \quad \text{definition } \mathsf{zip.T.G}\text{, tree type fusion} \quad \} \\
 & \alpha_{(^k\mathsf{T})} \cdot \mathsf{zip.T.H} \subseteq (\!|\mathsf{id_G}\otimes;\ \mathsf{G}(^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle} \cdot \alpha\otimes\mathsf{id}_{\mathsf{G}(^k\mathsf{T})}|\!) \\
\Leftarrow & \quad \{ \quad \text{definition } \mathsf{zip.T.G}\text{, catamorphism fusion, } \otimes \text{ functor} \quad \} \\
 & \alpha_{(^k\mathsf{T})} \cdot \mathsf{H}(^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{H})_{^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle} \subseteq \mathsf{G}(^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle} \cdot \alpha\otimes\alpha_{(^k\mathsf{T})} \\
\Leftarrow & \quad \{ \quad \alpha : \mathsf{G} \hookleftarrow \mathsf{H} \quad \} \\
 & \mathsf{G}(^k\mathsf{in}) \cdot \alpha_{(^k(\mathrm{Id}\otimes\mathsf{T}))} \cdot (\mathsf{zip}.\otimes.\mathsf{H})_{^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle} \subseteq \mathsf{G}(^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathsf{G})_{^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle} \cdot \alpha\otimes\alpha_{(^k\mathsf{T})} \\
\Leftarrow & \quad \{ \quad \alpha_{(^k(\mathrm{Id}\otimes\mathsf{T}))} = \alpha_{(^k\otimes)(^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle)}\ ,\ \ \alpha\otimes\alpha_{(^k\mathsf{T})} = \otimes(\alpha^2)_{^k\langle \mathrm{Id}\,,\,\mathsf{T}\rangle}\text{, lifting} \quad \} \\
 & \alpha_{(^k\otimes)} \cdot \mathsf{zip}.\otimes.\mathsf{H} \subseteq \mathsf{zip}.\otimes.\mathsf{G} \cdot \otimes(\alpha^2)
\end{array}
$$

5.7.3 Zips are compositional

In order to verify that $\mathsf{zip.FG}$ is compositional, we have to show that, for $\mathsf{F} : m \leftarrow k$, $\mathsf{G} : k \leftarrow l$, $\mathsf{H} : 1 \leftarrow n$, $\mathsf{I} : n \leftarrow o$,

$$\mathsf{zip.FG.HI} = (\mathsf{H}^m)(\mathsf{zip.FG.I}) \cdot (\mathsf{zip.FG.H})_{\mathsf{I}^l}$$

assuming that $\mathsf{zip.F}$ and $\mathsf{zip.G}$ are compositional. We first prove a lemma which starts with the rhs and assumes nothing about compositionality.

Lemma 5.52

$$
\begin{array}{cl}
 & (\mathsf{H}^m)(\mathsf{zip.FG.I}) \cdot (\mathsf{zip.FG.H})_{\mathsf{I}^l} \\
= & ((\mathsf{H}^m)(\mathsf{zip.F.I}) \cdot (\mathsf{zip.F.H})_{\mathsf{I}^k})_{^o\mathsf{G}} \cdot \mathsf{F}((\mathsf{H}^k)(\mathsf{zip.G.I}) \cdot (\mathsf{zip.G.H})_{\mathsf{I}^l})
\end{array}
$$

Proof

$$
\begin{array}{cl}
 & (\mathsf{H}^m)(\mathsf{zip.FG.I}) \cdot (\mathsf{zip.FG.H})_{\mathsf{I}^l} \\
= & \{ \quad \text{definition } \mathsf{zip.FG} \quad \} \\
 & (\mathsf{H}^m)((\mathsf{zip.F.I})_{^o\mathsf{G}} \cdot (^n\mathsf{F})(\mathsf{zip.G.I})) \cdot ((\mathsf{zip.F.H})_{^n\mathsf{G}} \cdot \mathsf{F}(\mathsf{zip.G.H}))_{\mathsf{I}^l} \\
= & \{ \quad \mathsf{H}^m \text{ functor, lifting} \quad \} \\
 & (\mathsf{H}^m)(\mathsf{zip.F.I})_{^o\mathsf{G}} \cdot (\mathsf{H}^m)(^n\mathsf{F})(\mathsf{zip.G.I}) \cdot (\mathsf{zip.F.H})_{(^n\mathsf{G})(\mathsf{I}^l)} \cdot \mathsf{F}(\mathsf{zip.G.H})_{\mathsf{I}^l} \\
= & \{ \quad \mathsf{zip.F.H} : (\mathsf{H}^m)(^n\mathsf{F}) \leftarrow \mathsf{F}(\mathsf{H}^k) \text{ and } \mathsf{zip.G.I} : (\mathsf{I}^k)(^o\mathsf{G}) \leftarrow (^n\mathsf{G})(\mathsf{I}^l). \\
 & \quad\ \text{Hence,} \\
 & \quad\quad (\mathsf{H}^m)(^n\mathsf{F})(\mathsf{zip.G.I}) \cdot (\mathsf{zip.F.H})_{(^n\mathsf{G})(\mathsf{I}^l)} \\
 & \quad\ = (\mathsf{zip.F.H})_{(\mathsf{I}^k)(^o\mathsf{G})} \cdot \mathsf{F}(\mathsf{H}^k)(\mathsf{zip.G.I}) \quad \} \\
 & (\mathsf{H}^m)(\mathsf{zip.F.I})_{^o\mathsf{G}} \cdot (\mathsf{zip.F.H})_{(\mathsf{I}^k)(^o\mathsf{G})} \cdot \mathsf{F}(\mathsf{H}^k)(\mathsf{zip.G.I}) \cdot \mathsf{F}(\mathsf{zip.G.H})_{\mathsf{I}^l} \\
= & \{ \quad \text{lifting, } \mathsf{F} \text{ functor} \quad \} \\
 & ((\mathsf{H}^m)(\mathsf{zip.F.I}) \cdot (\mathsf{zip.F.H})_{\mathsf{I}^k})_{^o\mathsf{G}} \cdot \mathsf{F}((\mathsf{H}^k)(\mathsf{zip.G.I}) \cdot (\mathsf{zip.G.H})_{\mathsf{I}^l})
\end{array}
$$

□

Using the above lemma, verification of compositionality of $\mathsf{zip.FG}$ is straightforward.

$$
\begin{array}{cl}
 & (\mathsf{H}^m)(\mathsf{zip.FG.I}) \cdot (\mathsf{zip.FG.H})_{\mathsf{I}^l} \\
= & \{ \quad \text{lemma (5.52)} \quad \} \\
 & ((\mathsf{H}^m)(\mathsf{zip.F.I}) \cdot (\mathsf{zip.F.H})_{\mathsf{I}^k})_{^n\mathsf{G}} \cdot \mathsf{F}((\mathsf{H}^k)(\mathsf{zip.G.I}) \cdot (\mathsf{zip.G.H})_{\mathsf{I}^l}) \\
= & \{ \quad \text{assumption: } \mathsf{zip.F} \text{ and } \mathsf{zip.G} \text{ are compositional} \quad \} \\
 & (\mathsf{zip.F.HI})_{^n\mathsf{G}} \cdot \mathsf{F}(\mathsf{zip.G.HI}) \\
= & \{ \quad \text{definition } \mathsf{zip.FG} \quad \} \\
 & \mathsf{zip.FG.HI}
\end{array}
$$

Compositionality of $\mathsf{zip}.\Delta_m\mathsf{F}_m$ follows from a dual proof to that given for lemma (5.25). For the verification of compositionality of $\mathsf{zip.T}$, where T is the tree type corresponding to $\otimes$, we use that the definition of $\mathsf{zip.T.F}$ was forced upon us. We have to prove, for $\mathsf{H} : 1 \leftarrow n$, $\mathsf{I} : n \leftarrow o$,

$$\mathsf{zip.T.HI} = \mathsf{H}(\mathsf{zip.T.I}) \cdot (\mathsf{zip.T.H})_{\mathsf{I}}$$

If we define mapping f_{HI} for $\mathsf{F} : l \leftarrow k$,

$$f_{\mathsf{HI}}(\mathsf{F}) = (\mathsf{H}^l)(\mathsf{zip.F.I}) \cdot (\mathsf{zip.F.H})_{\mathsf{I}^k}$$

we have to prove

$$f_{\mathsf{HI}}(\mathsf{T}) = \mathsf{zip.T.HI}$$

In subsection 5.6.5 we showed that this follows if we prove properties (5.40)—(5.42) for f_{HI}.

Property (5.42), i.e. $f_{HI}(\otimes) = \mathsf{zip}.\otimes.HI$, or, expanding the definition of f,

$$H(\mathsf{zip}.\otimes.I) \cdot (\mathsf{zip}.\otimes.H)_{I^2} = \mathsf{zip}.\otimes.HI$$

is just the assumption that $\mathsf{zip}.\otimes$ is compositional. Furthermore, we have to check property (5.41), i.e. for $F : 1 \leftarrow k$, $G : k \leftarrow l$,

$$f_{HI}(Id) = \mathsf{id}_{HI} \quad , \quad f_{HI}(G) = \Delta_k(f_{HI}(G_k)) \quad \text{and} \quad f_{HI}(FG) = f_{HI}(F)_{{}^nG} \cdot F(f_{HI}(G))$$

Expanding the definition of f, for $H : 1 \leftarrow n$, $I : n \leftarrow o$, we have

$$H(\mathsf{zip}.Id.I) \cdot (\mathsf{zip}.Id.H)_I = \mathsf{id}_{HI} \quad ,$$

$$(H^k)(\mathsf{zip}.G.I) \cdot (\mathsf{zip}.G.H)_I = \Delta_k(H(\mathsf{zip}.G_k.I) \cdot (\mathsf{zip}.G_k.H)_I)$$

and

$$\begin{array}{ll} & H(\mathsf{zip}.FG.I) \cdot (\mathsf{zip}.FG.H)_{I^l} \\ = & (H(\mathsf{zip}.F.I) \cdot (\mathsf{zip}.F.H)_{I^k})_{{}^oG} \cdot F((H^k)(\mathsf{zip}.G.I) \cdot (\mathsf{zip}.G.H)_{I^l}) \end{array}$$

The first conjunct follows from the definition of $\mathsf{zip}.Id$, the second conjunct follows from the fact that $\mathsf{zip}.G.F = \Delta_k(\mathsf{zip}.G_k.F)$, and the third conjunct is lemma (5.52). It remains to verify property (5.40), i.e.

$$HI({}^o\mathsf{in}) \cdot f_{HI}(Id \otimes T) = f_{HI}(T) \cdot \mathsf{in}_{HI}$$

or, expanding the definition of f,

$$HI({}^o\mathsf{in}) \cdot H(\mathsf{zip}.Id\otimes T.I) \cdot (\mathsf{zip}.Id\otimes T.H)_I = H(\mathsf{zip}.T.I) \cdot (\mathsf{zip}.T.H)_I \cdot \mathsf{in}_{HI}$$

Recall that $\mathsf{zip}.T.F$ was constructed such that, for $F : l \leftarrow k$,

$$\mathsf{zip}.T.F \cdot ({}^l\mathsf{in})_F = F({}^k\mathsf{in}) \cdot \mathsf{zip}.Id\otimes T.F \tag{5.53}$$

We calculate

$$\begin{array}{ll} & H(\mathsf{zip}.T.I) \cdot (\mathsf{zip}.T.H)_I \cdot \mathsf{in}_{HI} \\ = & \{ \quad \text{lifting, property (5.53)} \quad \} \\ & H(\mathsf{zip}.T.I) \cdot H({}^m\mathsf{in})_I \cdot (\mathsf{zip}.Id\otimes T.H)_I \\ = & \{ \quad H \text{ functor, property (5.53)} \quad \} \\ & HI({}^o\mathsf{in}) \cdot H(\mathsf{zip}.Id\otimes T.I) \cdot (\mathsf{zip}.Id\otimes T.H)_I \end{array}$$

Hence, we have proven that $\mathsf{zip}.T$ is compositional.

5.7.4 Zips respect projections

In this section we verify that each of the candidates respects projections, that is to say, we verify for $F : 1 \leftarrow k$,

$$\mathsf{zip}.F.\mathtt{Proj} = \mathsf{id}_{F(\mathtt{Proj}^k)} \quad \text{for each } \mathtt{Proj} : 1 \leftarrow l.$$

Verification for $\mathsf{zip}.K_A$ is trivial since $\mathsf{fan}.\mathtt{Proj} = \mathsf{id}$:

$$\mathsf{zip}.K_A.\mathtt{Proj} = (\mathsf{fan}.\mathtt{Proj})_{K_A} = \mathsf{id}_{K_A} = \mathsf{id}_{K_A(\mathtt{Proj}^k)}$$

We show that for $F : m \leftarrow k$, $G : k \leftarrow l$ the candidate $\mathsf{zip}.FG$ respects projections assuming that $\mathsf{zip}.F$ and $\mathsf{zip}.G$ respect projections. We calculate for $\mathtt{Proj} : 1 \leftarrow n$,

$$\begin{array}{ll}
 & \mathsf{zip}.FG.\mathtt{Proj} \\
= & \{ \quad \text{definition } \mathsf{zip}.FG \quad \} \\
 & (\mathsf{zip}.F.\mathtt{Proj})_{{}^nG} \cdot F(\mathsf{zip}.G.\mathtt{Proj}) \\
= & \{ \quad \mathsf{zip}.F \text{ and } \mathsf{zip}.G \text{ respect projections} \quad \} \\
 & \mathsf{id}_{F(\mathtt{Proj}^k)({}^nG)} \cdot F(\mathsf{id}^k)_{G(\mathtt{Proj}^l)} \\
= & \{ \quad (\mathtt{Proj}^k)({}^nG) = G(\mathtt{Proj}^l), \text{ identities} \quad \} \\
 & \mathsf{id}_{FG(\mathtt{Proj}^l)}
\end{array}$$

Verification for $\mathsf{zip}.\Delta_m F_m$ follows from a dual proof to that given for lemma (5.25). For the verification that $\mathsf{zip}.T$ respects projections we assume that $\mathsf{zip}.\otimes$ respects projections, then we calculate for $\mathtt{Proj} : 1 \leftarrow k$,

$$\begin{array}{ll}
 & \mathsf{zip}.T.\mathtt{Proj} \\
= & \{ \quad \text{definition } \mathsf{zip}.T \quad \} \\
 & (\!|\mathsf{id}_{\mathtt{Proj}}\otimes;\ \mathtt{Proj}({}^k\mathsf{in}) \cdot (\mathsf{zip}.\otimes.\mathtt{Proj})_{{}^k\langle \mathrm{Id}\,,\,T\rangle}|\!) \\
= & \{ \quad \mathtt{Proj}({}^k\mathsf{in}) = \mathsf{in}_{\mathtt{Proj}},\ \mathsf{zip}.\otimes \text{ respects projections} \quad \} \\
 & (\!|\mathsf{id}_{\mathtt{Proj}}\otimes;\ \mathsf{in}_{\mathtt{Proj}} \cdot \mathsf{id}_{\otimes(\mathtt{Proj}^2)({}^k\langle \mathrm{Id}\,,\,T\rangle)}|\!) \\
= & \{ \quad \text{identities} \quad \} \\
 & (\!|\mathsf{id}_{\mathtt{Proj}}\otimes;\ \mathsf{in}_{\mathtt{Proj}}|\!) \\
= & \{ \quad (\!|\mathsf{id}\otimes;\ \mathsf{in}|\!) = \mathsf{id}_T \quad \} \\
 & \mathsf{id}_{T(\mathtt{Proj})}
\end{array}$$

5.8 Regular relators are commuting

In this section we prove that the class of regular relators is commuting. That is to say, we prove for regular F and G that

$$\mathsf{zip}.F.G = \tau(\mathsf{zip}.G.F)^{\circ}_{\tau}$$

or, equivalently,

$$\mathsf{zip}.G.F = \tau(\mathsf{zip}.F.G)^{\circ}_{\tau}$$

As we have already shown, the pair of relators $(F\,,G)$ is commuting if F or G is recoverable from its cofan function. This dispenses with the cases that either F or G is the identity relator, a projection relator, coproduct or product.

Commutativity of $(FG\,,H)$ follows by construction: the half-zip $\mathsf{zip}.FG.H$ was derived from the requirement of commutativity of $(FG\,,H)$. We assumed the existence of the half-zip $\mathsf{zip}.H.FG$ and commutativity of $(F\,,H)$ and $(G\,,H)$.

Commutativity of $(\Delta_m F_m\,,G)$ follows from commutativity of $(F_m\,,G)$ for each m, since using lemma (5.24),

$$\mathsf{zip}.\Delta_m F_m.G = \Delta_m \mathsf{zip}.F_m.G = \Delta_m(\mathsf{zip}.G.F_m)^{\circ}_{(\tau)} = \tau(\mathsf{zip}.G.\Delta_m F_m)^{\circ}_{\tau}$$

Commutativity of $(T\,,G)$ follows from commutativity of $(\otimes\,,G)$. We have to show that $\mathsf{zip}.T.G = (\mathsf{zip}.G.T)^{\circ}$, hence we define $f_G(F) = (\mathsf{zip}.G.F)^{\circ}_{\tau}$ and we show that properties (5.40)—(5.42) hold for f_G. We start with property (5.40), for $G : 1 \leftarrow k$,

$$\begin{array}{ll} & G({}^{k}\mathsf{in}) \cdot f_G(\mathrm{Id}\otimes T) = f_G(T) \cdot \mathsf{in}_G \\ \equiv & \{ \quad \text{definition } f_G\text{, converse, } {}^{k}\mathsf{in} = \mathsf{in}^{k} \quad \} \\ & \mathsf{in}^{\circ}_G \cdot \mathsf{zip}.G.T = \mathsf{zip}.G.\mathrm{Id}\otimes T \cdot G(\mathsf{in}^{\circ})^{k} \end{array}$$

which follows from functionality of in, $\mathsf{in} : T \leftarrow \mathrm{Id}\otimes T$, and lemma (5.13b). Furthermore, property (5.41) which states that f_G is compositional and respects identities, i.e.

$$f_G(HI) = f_G(H)_{kI} \cdot H(f_G(I)) \;\wedge\; f_G(I) = \Delta_k(f_G(I_k)) \;\wedge\; f_G(\mathrm{Id}) = \mathsf{id}_G$$

is the property that $\mathsf{zip}.G$ is compositional and respects identities. Similarly, property (5.42), i.e.

$$f_G(\otimes) = \mathsf{zip}.\otimes.G$$

is just the assumption that $(\otimes\,,G)$ commutes.

It remains to verify that $(K_A\,,K_B)$ commutes. We have $\mathsf{zip}.K_A.K_B = (\mathsf{fan}.K_B)_{K_A} = \top\!\!\top_{K_B\,,K_A}$, hence, $\mathsf{zip}.K_A.K_B = (\mathsf{zip}.K_A.K_B)^{\circ}$.

This concludes the proof of the zip-a-dee-doo-dah theorem.

5.9 Broadcasts

In section 5.3.4 on strength we claimed that zips were an instance of strength. We showed that the broadcast $\mathsf{zip}.(\times\,B).F_A$ is a strength of endorelator F provided that it is functional and natural in B. In this section we show that this indeed the case for the half-zip constructed in the previous sections. In Chapter 6, we prove that this half-zip is the unique strength for relator F provided F has membership. In other words, it follows that a strength is an instance of a half-zip.

First, we construct the half-zip $\mathsf{zip}.(\times\,B).F$. We calculate,

$$
\begin{array}{ll}
 & \mathsf{zip}.(\times\, B).F \\
= & \{\quad (\times\, B) = (\times)\langle \mathrm{Id}\,,\, K_B\rangle,\ \text{zip of composition (5.38)}\quad\} \\
 & (\mathsf{zip}.\times.F)_{\langle \mathrm{Id}\,,\, K_B\rangle} \cdot (\times)\mathsf{zip}.\langle \mathrm{Id}\,,\, K_B\rangle.F \\
= & \{\quad \text{zip of ``}\times\text{'' (5.32); zip respect tuples (5.39)}\quad\} \\
 & (F\mathsf{outl}^\circ \blacktriangle F\mathsf{outr}^\circ)_{\langle \mathrm{Id}\,,\, K_B\rangle} \cdot (\times)\langle \mathsf{zip}.\mathrm{Id}.F\,,\, \mathsf{zip}.K_B.F\rangle \\
= & \{\quad \text{zip respect identities, } \mathsf{zip}.K_B.F = (\mathsf{fan}.F)_{K_B}\ \text{(5.37)}\quad\} \\
 & (F\mathsf{outl}^\circ \blacktriangle F\mathsf{outr}^\circ)_{\langle \mathrm{Id}\,,\, K_B\rangle} \cdot \mathsf{id}_F \times (\mathsf{fan}.F)_{K_B}
\end{array}
$$

Hence,

$$(\mathsf{zip}.(\times\, B).F)_A \;=\; F\mathsf{outl}^\circ_{A,B} \blacktriangle F\mathsf{outr}^\circ_{A,B} \;\cdot\; \mathsf{id}_{FA} \times (\mathsf{fan}.F)_B \;\;.$$

Next, we verify that $(\mathsf{zip}.(\times\, B).F)_A$ is functional, and natural in B.

Lemma 5.54 Let $\mathsf{str}_{A,B} \;=\; F\mathsf{outl}^\circ_{A,B} \blacktriangle F\mathsf{outr}^\circ_{A,B} \;\cdot\; \mathsf{id}_{FA} \times (\mathsf{fan}.F)_B$, then str is a functional natural transformation of type $F(A \times B) \hookleftarrow FA \times B$.

Proof Naturality of $\mathsf{str}_{A,B} \;:\; F(A\times B) \hookleftarrow FA \times B$ follows from naturality of $\mathsf{zip}_{A,B} \;:\; F(A\times B) \leftarrow FA \times FB$ and naturality of $\mathsf{fan}_B \;:\; FB \hookleftarrow B$. Totality is proved as follows:

$$
\begin{array}{ll}
 & \mathsf{str}{>} \\
= & \{\quad \text{definition } \mathsf{str}\text{, product, domain of intersection}\quad\} \\
 & \mathsf{id} \cap \mathsf{outl}^\circ_{FA,B} \cdot F(\mathsf{outl}_{A,B} \cdot \mathsf{outr}^\circ_{A,B}) \cdot \mathsf{fan}_B \cdot \mathsf{outr}_{FA,B} \\
= & \{\quad \mathsf{outl}_{A,B} \cdot \mathsf{outr}^\circ_{A,B} = \top\!\top_{A,B},\ \text{fan: } F\top\!\top_{A,B} \cdot \mathsf{fan}_B = \top\!\top_{FA,B}\quad\} \\
 & \mathsf{id} \cap \mathsf{outl}^\circ_{FA,B} \cdot \top\!\top \cdot \mathsf{outr}_{FA,B} \\
= & \{\quad \mathsf{outl}\text{ and }\mathsf{outr}\text{ are total}\quad\} \\
 & \mathsf{id}_{FA\times B}
\end{array}
$$

Simplicity of str we verify by:

$$
\begin{array}{ll}
 & \mathsf{str}_{A,B} \cdot \mathsf{str}^\circ_{A,B} \\
= & \{\quad \text{definition } \mathsf{str}\text{, product}\quad\} \\
 & F\mathsf{outl}^\circ_{A,B} \blacktriangle F\mathsf{outr}^\circ_{A,B} \;\cdot\; \mathsf{id}_A \times (\mathsf{fan}_B \cdot \mathsf{fan}^\circ_B) \;\cdot\; F\mathsf{outl}_{A,B} \vartriangle F\mathsf{outr}_{A,B} \\
= & \{\quad \text{product}\quad\} \\
 & F(\mathsf{outl}^\circ_{A,B} \cdot \mathsf{outl}_{A,B}) \cap F\mathsf{outr}^\circ_{A,B} \cdot \mathsf{fan}_B \cdot \mathsf{fan}^\circ_B \cdot F\mathsf{outr}_{A,B} \\
\subseteq & \{\quad \text{lemma (4.41)}\quad\} \\
 & F(\mathsf{outl}^\circ_{A,B} \cdot \mathsf{outl}_{A,B} \cap \mathsf{outr}^\circ_{A,B} \cdot \mathsf{outr}_{A,B}) \\
= & \{\quad \mathsf{outl}^\circ_{A,B} \cdot \mathsf{outl}_{A,B} \cap \mathsf{outr}^\circ_{A,B} \cdot \mathsf{outr}_{A,B} = \mathsf{id}_{A\times B}\quad\} \\
 & \mathsf{id}_{F(A\times B)}
\end{array}
$$

□

Combining lemma (5.54) with the results of section 5.3.4, we get as corollary

Corollary 5.55 $\mathsf{Foutl}^{\circ}_{A,B} \blacktriangle \mathsf{Foutr}^{\circ}_{A,B} \cdot \mathsf{id}_{\mathsf{F}A} \times (\mathsf{fan}.\mathsf{F})_B$ is a strength of relator F.
□

Hence, the construction given for half-zip $\mathsf{zip}.(\times B).\mathsf{F}$ is a strength of relator F.

Chapter 6

Strength, Copies map and Fan

In chapter 4 on membership, we showed that the existence of membership of relator F implied the existence of the canonical fan. In chapter 5 on commuting relators we showed that the existence of the canonical fan implies the existence of a strength of F. Thus a relator with membership has a strength, i.e. is strong.

This chapter reformulates and extends the work of Hoogendijk and De Moor [20]. In [20] an abstract of notion of a fan (satisfied by the canonical fan) is given and it is shown that the fans and strengths of a relator are in one-to-one correspondence. Furthermore, it is shown that the existence of a membership relation implies the existence of a unique fan and (thus) a unique strength. In this chapter we extend this analysis to include Jay's [23] notion of a copies map. The interpretation of a copies map is that it takes an F-shape and a value and generates an F-structure by filling the F-shape with copies of the given value.

We show that the existence of a strength implies the existence of a copies map. Furthermore, we show that for a binary intersection preserving relator F, the existence of a copies map implies the existence of a fan of relator F. Finally, we show that the existence of a fan, not necessarily the canonical fan, implies the existence of a strength. Furthermore, we show that all the constructions are injective. In other words, for a binary intersection preserving relator, strengths, fans and copies maps are all in a one-to-one correspondence.

Moreover, if we assume that the relator has membership then we can prove, just as for fan, that the relator has a unique strength and a unique copies map. We prove the uniqueness using the fact that a fan is the largest relation of its type. Furthermore, we show that an arbitrary natural transformation of type $\mathsf{F} \hookleftarrow \mathsf{G}$ is coherent with respect to the strength and the copies map of F and G.

Recall that all regular relators and the power relator have a membership relation. Hence, these relators have a unique fan, strength and copies map.

Since the definition of strength is only given for endorelators, we will assume throughout the remainder of this chapter a fixed, but arbitrary endorelator F.

6.1 Zips

We define as a shorthand $\mathsf{zip} = \mathsf{zip}.\times.\mathsf{F}$, that is to say,

$$\mathsf{zip} = \mathsf{Foutl}^\circ \blacktriangle \mathsf{Foutr}^\circ \ .$$

Recall that we have the coslok properties (5.35),

$$\mathsf{F}(\mathsf{R} \blacktriangle \mathsf{S}) \cdot \mathsf{zip} = \mathsf{FR} \blacktriangle \mathsf{FS} \ . \tag{6.1}$$

Note that zip is a half-zip. For instance, we have for $\alpha : \mathsf{F} \hookleftarrow \mathrm{Id}$,

$$\mathsf{zip}_{\mathsf{A},\mathsf{B}} \cdot \alpha_\mathsf{A} \times \alpha_\mathsf{B} \supseteq \alpha_{\mathsf{A}\times\mathsf{B}} \ . \tag{6.2}$$

since

$$(\mathsf{zip}.\times.\mathsf{F})_{\mathsf{A},\mathsf{B}} \cdot \alpha_\mathsf{A} \times \alpha_\mathsf{B} \supseteq \alpha_{\mathsf{A}\times\mathsf{B}} \cdot (\mathsf{zip}.\times.\mathrm{Id})_{\mathsf{A},\mathsf{B}} = \alpha_{\mathsf{A}\times\mathsf{B}}$$

Furthermore, for the combination of the natural isomorphism ass, $\mathsf{ass}_{\mathsf{A},\mathsf{B},\mathsf{C}} : (\mathsf{A}\times\mathsf{B})\times\mathsf{C} \leftarrow \mathsf{A}\times(\mathsf{B}\times\mathsf{C})$ and zip, we have,

$$\mathsf{zip}_{\mathsf{A}\times\mathsf{B},\mathsf{C}} \cdot \mathsf{zip}_{\mathsf{A},\mathsf{B}} \times \mathsf{id}_{\mathsf{FC}} \cdot \mathsf{ass}_{\mathsf{FA},\mathsf{FB},\mathsf{FC}} = \mathsf{Fass}_{\mathsf{A},\mathsf{B},\mathsf{C}} \cdot \mathsf{zip}_{\mathsf{A},\mathsf{B}\times\mathsf{C}} \cdot \mathsf{id}_\mathsf{A} \times \mathsf{zip}_{\mathsf{B},\mathsf{C}} \tag{6.3}$$

or, equivalently, since ass is an isomorphism,

$$\mathsf{zip}^\circ_{\mathsf{A},\mathsf{B}} \times \mathsf{id}_{\mathsf{FC}} \cdot \mathsf{zip}^\circ_{\mathsf{A}\times\mathsf{B},\mathsf{C}} \cdot \mathsf{Fass}_{\mathsf{A},\mathsf{B},\mathsf{C}} = \mathsf{ass}_{\mathsf{FA},\mathsf{FB},\mathsf{FC}} \cdot \mathsf{id}_\mathsf{A} \times \mathsf{zip}^\circ_{\mathsf{B},\mathsf{C}} \cdot \mathsf{zip}^\circ_{\mathsf{A},\mathsf{B}\times\mathsf{C}} \tag{6.4}$$

Note that $\mathsf{zip}^\circ = \mathsf{Foutl} \vartriangle \mathsf{Foutr}$, so equation (6.4) is a statement in Map since all components involved are functions. Now, property (6.4) follows by a straightforward calculation using the universal property of product.

6.2 Constructing copies map, fan and strength

We start with the definition of a *copies map* [23] and show that having a strength, we can construct a copies map. And we show that for a binary intersection preserving relator, we can construct a fan from a copies map.

Definition 6.5 A *copies map* cop is a functional natural transformation of type $\mathsf{FA} \hookleftarrow \mathsf{F1}\times\mathsf{A}$ such that,

$$\mathsf{F!}_\mathsf{A} \cdot \mathsf{cop}_\mathsf{A} = \mathsf{outl}_{\mathsf{F1},\mathsf{A}} \ . \tag{6.6}$$

□

The interpretation of a copies map is that it takes an F-shape, i.e. an element from F1, and a value and fills the F-shape with copies of the given value. Equation (6.6) expresses that the shape of the result is the same as the given F-shape.

Recall that the interpretation of a strength is a so-called broadcast. That is to say, $\mathsf{str}_{\mathsf{A},\mathsf{B}} : \mathsf{F}(\mathsf{A}\times\mathsf{B}) \leftarrow \mathsf{FA}\times\mathsf{B}$ takes an F-structure of A's and a value of type B and pairs all the elements of the F-structure with the given value of type B. This suggest that $\mathsf{str}_{1,\mathsf{B}} : \mathsf{F}(1\times\mathsf{B}) \leftarrow \mathsf{F1}\times\mathsf{B}$ and $\mathsf{cop}_\mathsf{B} : \mathsf{FB} \leftarrow \mathsf{F1}\times\mathsf{B}$ are closely related. Indeed, we have,

Lemma 6.7 If str is a strength of relator F then $\mathsf{F lid}_A \cdot \mathsf{str}_{1,A}$ is a copies map of F (where $\mathsf{lid}_A : A \leftarrow 1 \times A$ is the obvious natural isomorphism).

Proof

$$
\begin{array}{ll}
 & \mathsf{F!}_A \cdot \mathsf{Flid}_A \cdot \mathsf{str}_{1,A} \\
= & \quad \{ \quad \text{terminality: } !_A \cdot \mathsf{lid}_A = !_{1\times A} = \mathsf{outl}_{1,A} \quad \} \\
 & \mathsf{Foutl}_{1,A} \cdot \mathsf{str}_{1,A} \\
= & \quad \{ \quad \text{property (5.17)} \quad \} \\
 & \mathsf{outl}_{\mathsf{F}1,A}
\end{array}
$$

□

Comparing the interpretation of a fan and a copies map one can say that a copies map is the functional counterpart of the nondeterministic fan. A fan takes a value and generates an arbitrary F-structure filled with copies of the value, whereas the copies map takes a specific F-shape and generates an F-structure of that shape filled with copies of the value. In other words, if we first generate an arbitrary F-shape and we feed it together with a value to a copies map, we should get the same result as with the fan. Indeed, this is true for a binary intersection preserving relator F. In [1] it has been shown that product and coproduct preserves binary intersections. Furthermore, a tree type relator preserves binary intersections if the base relator preserves binary intersections. Hence, it follows that all regular relators preserves binary intersections.

Lemma 6.8 If cop is a copies map of binary intersection preserving relator F then

$$\mathsf{fan}_A = \mathsf{cop}_A \cdot \mathsf{outr}^{\circ}_{\mathsf{F}1,A}$$

is a fan of F.

Proof Naturality of $\mathsf{fan}_A : \mathsf{F}A \hookleftarrow A$ follows from naturality of $\mathsf{cop}_A : \mathsf{F}A \hookleftarrow \mathsf{F}1 \times A$ and naturality of $\mathsf{outr}^{\circ}_{\mathsf{F}1,A} : \mathsf{F}1 \times A \leftarrow A$. Top strictness we verify by,

$$
\begin{array}{ll}
 & \mathsf{F}\top\!\!\top_{B,A} \cdot \mathsf{cop}_A \cdot \mathsf{outr}^{\circ}_{\mathsf{F}1,A} \\
= & \quad \{ \quad \top\!\!\top_{B,A} = !^{\circ}_B \cdot !_A, \text{ copies map (6.6)} \quad \} \\
 & \mathsf{F!}^{\circ}_B \cdot \mathsf{outl}_{A,\mathsf{F}1} \cdot \mathsf{outr}^{\circ}_{\mathsf{F}1,A} \\
= & \quad \{ \quad \mathsf{outl} \cdot \mathsf{outr} = \top\!\!\top, \text{ domains (2.28): } !_B \text{ total} \quad \} \\
 & \top\!\!\top_{\mathsf{F}B,A}
\end{array}
$$

For preservation of binary intersection, we prove for each $R, S : B \leftarrow A$,

$$
\begin{array}{ll}
 & \mathsf{F}(R \cap S) \cdot \mathsf{fan}_A \\
\subseteq & \quad \{ \quad \text{monotonicity} \quad \} \\
 & \mathsf{F}R \cdot \mathsf{fan}_A \cap \mathsf{F}S \cdot \mathsf{fan}_A
\end{array}
$$

$$
\begin{array}{ll}
\subseteq & \{ \quad \text{modular law} \quad \} \\
& (\mathsf{FR} \cap \mathsf{FS} \cdot \mathsf{fan}_A \cdot \mathsf{fan}_A{}^\circ) \cdot \mathsf{fan}_A \\
\subseteq & \{ \quad \text{claim: } \mathsf{FR} \cap \mathsf{FS} \cdot \mathsf{fan}_A \cdot \mathsf{fan}_A{}^\circ \subseteq \mathsf{FS} \quad \} \\
& (\mathsf{FR} \cap \mathsf{FS}) \cdot \mathsf{fan}_A \\
\subseteq & \{ \quad \text{assumption: } \mathsf{F} \text{ preserves binary intersections} \quad \} \\
& \mathsf{F}(\mathsf{R} \cap \mathsf{S}) \cdot \mathsf{fan}_A
\end{array}
$$

Hence,

$$\mathsf{F}(\mathsf{R} \cap \mathsf{S}) \cdot \mathsf{fan}_A \; = \; \mathsf{FR} \cdot \mathsf{fan}_A \cap \mathsf{FS} \cdot \mathsf{fan}_A$$

For the claim, we prove

$$
\begin{array}{ll}
& \mathsf{FR} \cap \mathsf{FS} \cdot \mathsf{fan}_A \cdot \mathsf{fan}_A^\circ \\
= & \{ \quad \text{definition } \mathsf{fan}_A \quad \} \\
& \mathsf{FR} \cap \mathsf{FS} \cdot \mathsf{cop}_A \cdot \mathsf{outr}^\circ_{\mathsf{F}1,A} \cdot \mathsf{outr}_{\mathsf{F}1,A} \cdot \mathsf{cop}_A{}^\circ \\
\subseteq & \{ \quad \text{modular law} \quad \} \\
& \mathsf{FS} \cdot \mathsf{cop} \cdot (\mathsf{cop}^\circ \cdot \mathsf{FS}^\circ \cdot \mathsf{FR} \cdot \mathsf{cop} \cap \mathsf{outr}^\circ_{\mathsf{F}1,A} \cdot \mathsf{outr}_{\mathsf{F}1,A}) \cdot \mathsf{cop}^\circ \\
\subseteq & \{ \quad \mathsf{FS}^\circ \cdot \mathsf{FR} \subseteq \mathsf{F}\top\!\!\top_{A,A} = \mathsf{F}!_A{}^\circ \cdot \mathsf{F}!_A, \text{ copies map (6.6) (twice)} \quad \} \\
& \mathsf{FS} \cdot \mathsf{cop}_A \cdot (\mathsf{outl}^\circ_{\mathsf{F}1,A} \cdot \mathsf{outl}_{\mathsf{F}1,A} \cap \mathsf{outr}^\circ_{\mathsf{F}1,A} \cdot \mathsf{outr}_{\mathsf{F}1,A}) \cdot \mathsf{cop}_A{}^\circ \\
\subseteq & \{ \quad \mathsf{outl}^\circ_{\mathsf{F}1,A} \cdot \mathsf{outl}_{\mathsf{F}1,A} \cap \mathsf{outr}^\circ_{\mathsf{F}1,A} \cdot \mathsf{outr}_{\mathsf{F}1,A} = \mathsf{id}, \mathsf{cop} \text{ simple} \quad \} \\
& \mathsf{FS}
\end{array}
$$

□

Note that $\mathsf{outr}^\circ_{\mathsf{F}1,A} = \top\!\!\top_{\mathsf{F}1,A} \vartriangle \mathsf{id}_A$, so the interpretation of $\mathsf{outr}^\circ_{\mathsf{F}1,A}$ is that it generates from a value of A, a pair consisting of an arbitrary F-shape and the given value.

Combining the construction of a copies map from a strength and the construction of a fan from a copies map gives,

Corollary 6.9 Let F be a binary intersection preserving relator with a strength str, then

$$\mathsf{fan}_A = \mathsf{F}\mathsf{lid}_A \cdot \mathsf{str}_{1\,,A} \cdot \mathsf{outr}^\circ_{\mathsf{F}1,A}$$

is a fan of F.
□

In subsection 5.3.4 and section 5.9 we have shown that $(\mathsf{zip}.(\times A).\mathsf{F})_B$ is a strength of relator F. Using the inductive construction of the half-zips it followed that

$$(\mathsf{zip}.(\times A).\mathsf{F})_B = (\mathsf{zip}.\times.\mathsf{F})_{A,B} \cdot \mathsf{id}_A \times (\mathsf{fan}.\mathsf{F})_B$$

where $\mathsf{fan}.\mathsf{F}$ is the *canonical* fan of relator F, i.e. the fan constructed using the membership relation of F. However, we can prove that $(\mathsf{zip}.(\times A).\mathsf{F})_B$ is a strength of F only assuming that $\mathsf{fan}.\mathsf{F}$ is a fan of F according to the definition of fan (4.39),

Lemma 6.10 If fan is a fan of relator F then

$$\mathsf{str}_{A\,,\,B} = \mathsf{zip}_{A\,,B} \cdot \mathsf{id}_{FA} \times \mathsf{fan}_B$$

is a strength of F.

Proof Naturality and functionality of $\mathsf{str}_{A\,,\,B}$ follow from lemma (5.54). For unit coherence we prove:

$$
\begin{array}{ll}
 & \mathsf{F}\mathsf{rid}_A \cdot \mathsf{zip}_{A\,,1} \cdot \mathsf{id}_{FA} \times \mathsf{fan}_1 \\
= & \{ \quad \mathsf{rid}_A = \mathsf{id}_A \blacktriangle !^{\circ}_A, \text{ slok property of zip (6.1)} \quad \} \\
 & \mathsf{id}_{FA} \blacktriangle \mathsf{F}!^{\circ}_A \cdot \mathsf{id}_{FA} \times \mathsf{fan}_1 \\
= & \{ \quad \text{product} \quad \} \\
 & \mathsf{id}_{FA} \blacktriangle (\mathsf{F}!^{\circ}_A \cdot \mathsf{fan}_A) \\
= & \{ \quad \mathsf{fan}\ \text{fan: } \mathsf{F}!^{\circ}_A \cdot \mathsf{fan}_A = !^{\circ}_{FA},\ \mathsf{id}_{FA} \blacktriangle !^{\circ}_{FA} = \mathsf{rid}_{FA} \quad \} \\
 & \mathsf{rid}_{FA}
\end{array}
$$

Associative coherence follows from:

$$
\begin{array}{ll}
 & \mathsf{str}_{A\times B\,,C} \cdot \mathsf{str}_{A\,,B} \times \mathsf{id}_C \cdot \mathsf{ass}_{FA\,,B\,,C} \\
= & \{ \quad \text{definition of } \mathsf{str}\text{, product} \quad \} \\
 & \mathsf{zip}_{A\times B\,,C} \cdot \mathsf{zip}_{A\,,B} \times \mathsf{id}_{FC} \cdot (\mathsf{id}_{FA} \times \mathsf{fan}_B) \times \mathsf{fan}_C \cdot \mathsf{ass}_{FA\,,B\,,C} \\
= & \{ \quad \text{associate} \quad \} \\
 & \mathsf{zip}_{A\times B\,,C} \cdot \mathsf{zip}_{A\,,B} \times \mathsf{id}_{FC} \cdot \mathsf{ass}_{FA\,,FB\,,FC} \cdot \mathsf{id}_{FA} \times (\mathsf{fan}_B \times \mathsf{fan}_C) \\
= & \{ \quad \text{property (6.3)} \quad \} \\
 & \mathsf{F}\mathsf{ass}_{A\,,B\,,C} \cdot \mathsf{zip}_{A\,,B\times C} \cdot \mathsf{id}_{FA} \times \mathsf{zip}_{B\,,C} \cdot \mathsf{id}_{FA} \times (\mathsf{fan}_B \times \mathsf{fan}_C) \\
\supseteq & \{ \quad \text{product, property (6.2): } \mathsf{fan} : \mathsf{F} \hookleftarrow \mathrm{Id} \quad \} \\
 & \mathsf{F}\mathsf{ass}_{A\,,B\,,C} \cdot \mathsf{zip}_{A\,,B\times C} \cdot \mathsf{id}_{FA} \times \mathsf{fan}_{B\times C} \\
= & \{ \quad \text{definition } \mathsf{str} \quad \} \\
 & \mathsf{F}\mathsf{ass}_{A\,,B\,,C} \cdot \mathsf{str}_{A\,,B\times C}
\end{array}
$$

Equality now follows because $\mathsf{str} \cdot \mathsf{str} \times \mathsf{id} \cdot \mathsf{ass}$ and $\mathsf{F}\mathsf{ass} \cdot \mathsf{str}$ are both functions.
□

We have shown how one can construct a fan from a copies map, and how one construct a strength from a fan. Combining both constructions gives,

Corollary 6.11 If cop is a copies map of binary intersection preserving relator F then,

$$\mathsf{str}_{A\,,B} = \mathsf{zip}_{A\,,B} \cdot \mathsf{id}_{FA} \times (\mathsf{cop}_B \cdot \mathsf{outr}^{\circ}_{F1\,,B})$$

is a strength of F.

Proof Follows by combining the constructions of lemmas (6.8) and (6.10).
□

The construction of corollary (6.11) is the combination of first constructing a fan, which is nondeterministic, from a copies map, and then constructing a strength from a fan. However, both a copies map and a strength are functions. This suggests that there is a more direct way of constructing a strength from a copies map. Indeed, we have,

Corollary 6.12 If cop is a copies map of binary intersection preserving relator F then,

$$
\begin{aligned}
& \mathsf{zip}_{A,B} \cdot \mathsf{id}_{FA} \times (\mathsf{cop}_B \cdot \mathsf{outr}^{\circ}_{F1,B}) \\
= & \\
& \mathsf{zip}_{A,B} \cdot \mathsf{outl}_{FA,B} \vartriangle (\mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B)
\end{aligned}
$$

Proof We calculate,

$$
\begin{aligned}
& \mathsf{zip}_{A,B} \cdot \mathsf{id}_{FA} \times (\mathsf{cop}_B \cdot \mathsf{outr}^{\circ}_{F1,B}) \\
= & \quad \{ \quad \text{definition } \times \quad \} \\
& \mathsf{zip}_{A,B} \cdot \mathsf{outl}_{FA,B} \vartriangle (\mathsf{cop}_B \cdot \mathsf{outr}^{\circ}_{F1,B} \cdot \mathsf{outr}_{FA,B}) \\
\supseteq & \quad \{ \quad \mathsf{outr}^{\circ}_{F1,B} \cdot \mathsf{outr}_{FA,B} = \top\!\!\top_{F1,FA} \times \mathsf{id}_B \supseteq \mathsf{F!}_A \times \mathsf{id}_B \quad \} \\
& \mathsf{zip}_{A,B} \cdot \mathsf{outl}_{FA,B} \vartriangle (\mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B)
\end{aligned}
$$

Now, equality follows if we prove that $\mathsf{zip}_{FA,B} \cdot \mathsf{outl}_{A,B} \vartriangle (\mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B)$ is total:

$$
\begin{aligned}
& (\mathsf{zip}_{A,B} \cdot \mathsf{outl}_{FA,B} \vartriangle (\mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B))> \\
= & \quad \{ \quad \text{definition zip, product} \quad \} \\
& (\mathsf{Foutl}^{\circ}_{A,B} \cdot \mathsf{outl}_{FA,B} \cap \mathsf{Foutr}^{\circ}_{A,B} \cdot \mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B)> \\
= & \quad \{ \quad \text{domain intersection (2.27)} \quad \} \\
& \mathsf{id}_{A\times B} \cap \mathsf{outl}^{\circ}_{FA,B} \cdot \mathsf{Foutl}_{A,B} \cdot \mathsf{Foutr}^{\circ}_{A,B} \cdot \mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B \\
= & \quad \{ \quad \mathsf{Foutl}_{A,B} \cdot \mathsf{Foutr}^{\circ}_{A,B} = \mathsf{F}\top\!\!\top_{A,B} = \mathsf{F!}^{\circ}_A \cdot \mathsf{F!}_B \quad \} \\
& \mathsf{id}_{A\times B} \cap \mathsf{outl}^{\circ}_{FA,B} \cdot \mathsf{F!}^{\circ}_A \cdot \mathsf{F!}_B \cdot \mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B \\
= & \quad \{ \quad \text{copies map: } \mathsf{F!}_B \cdot \mathsf{cop}_B = \mathsf{outl}_{F1,B}\text{, naturality outl} \quad \} \\
& \mathsf{id}_{A\times B} \cap \mathsf{outl}^{\circ}_{FA,B} \cdot \mathsf{F!}^{\circ}_A \cdot \mathsf{F!}_A \cdot \mathsf{outl}_{FA,B} \\
= & \quad \{ \quad \text{domain (2.24), } \mathsf{F!}_A \cdot \mathsf{outl}_{A,B} \text{ total} \quad \} \\
& \mathsf{id}_{FA\times B}
\end{aligned}
$$

□

Now, note that $\mathsf{outl}_{FA,B} \vartriangle (\mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B)$ is a function and $\mathsf{zip}_{A,B}$ is a partial function (simple) for binary intersection preserving relator F. The interpretation of $\mathsf{cop}_B \cdot \mathsf{F!}_A \times \mathsf{id}_B$ is that it takes the shape of the left-argument and fills it with the value of the right argument.

Similarly, we can construct a copies map from a fan,

Corollary 6.13 If fan is a fan of relator F then,

$$\mathsf{cop}_A = \mathsf{F!}^\circ_A \blacktriangle \mathsf{fan}_A$$

is a copies map of F.

Proof

$$\begin{array}{ll}
 & \mathsf{cop}_A \\
= & \quad \{ \quad \text{constructs of lemma (6.10) and (6.7)} \quad \} \\
 & \mathsf{Flid}_A \cdot \mathsf{zip}_{1,A} \cdot \mathsf{id}_{\mathsf{F1}} \times \mathsf{fan}_A \\
= & \quad \{ \quad \text{definition } \mathsf{zip}\text{, product} \quad \} \\
 & \mathsf{Flid}_A \cdot \mathsf{Foutl}^\circ_{1,A} \blacktriangle (\mathsf{Foutr}^\circ_{1,A} \cdot \mathsf{fan}_A) \\
= & \quad \{ \quad \mathsf{lid}_A = \mathsf{outr}_{1,A}\text{, modular identity} \quad \} \\
 & \mathsf{F}(\mathsf{outr}_{1,A} \cdot \mathsf{outl}^\circ_{1,A}) \blacktriangle \mathsf{fan}_A \\
= & \quad \{ \quad \mathsf{outr}_{1,A} \cdot \mathsf{outl}^\circ_{1,A} = \top\!\top_{A,1} = \mathsf{!}^\circ_A \quad \} \\
 & \mathsf{F!}^\circ_A \blacktriangle \mathsf{fan}_A
\end{array}$$

□

For the interpretation of $\mathsf{F!}^\circ \blacktriangle \mathsf{fan}$, we have,

$$x\,(\mathsf{F!}^\circ \blacktriangle \mathsf{fan})\,(y\,,z) \quad \equiv \quad y\;\mathsf{F!}\;x \;\wedge\; x\;\mathsf{fan}\;z$$

Hence, F-structure x has shape y and is filled with copies of the value z.

6.3 One-to-one correspondence

Next, we want to prove that all the constructions given are injective for a binary intersection preserving relator. We prove that in the following way. We have shown how one can construct a copies map from a strength, and a fan from a copies map, and a strength from fan. If we prove that the resulting strength equals the original strength it follows that the construction of a copies map or a fan from a strength is injective. Similarly, we prove this for copies map and fan.

Lemma 6.14 If str is a strength of binary intersection preserving relator F then,

$$\mathsf{str}_{A,B} = \mathsf{zip}_{A,B} \cdot \mathsf{id}_{\mathsf{FA}} \times (\mathsf{Flid}_B \cdot \mathsf{str}_{1,B} \cdot \mathsf{outr}^\circ_{\mathsf{F1},B}) \tag{6.15}$$

Proof

$$\begin{array}{ll}
 & \mathsf{zip}_{A,B} \cdot \mathsf{id}_{\mathsf{FA}} \times (\mathsf{Flid}_B \cdot \mathsf{str}_{1,B} \cdot \mathsf{outr}^\circ_{\mathsf{F1},B}) \\
= & \quad \{ \quad \text{definition } \mathsf{zip}\text{, product} \quad \} \\
 & \mathsf{Foutl}^\circ_{A,B} \blacktriangle (\mathsf{F}(\mathsf{outr}^\circ_{A,B} \cdot \mathsf{lid}_B) \cdot \mathsf{str}_{1,B} \cdot \mathsf{outr}^\circ_{\mathsf{F1},B})
\end{array}$$

$$
\begin{array}{ll}
= & \{ \quad \text{product: } \mathsf{outr}^\circ_{A,B} \cdot \mathsf{lid}_B = !^\circ_A \times \mathsf{id}_B \quad \} \\
& \mathsf{F outl}^\circ_{A,B} \blacktriangle (\mathsf{F}(!^\circ_A \times \mathsf{id}_B) \cdot \mathsf{str}_{1,B} \cdot \mathsf{outr}^\circ_{\mathsf{F}1,B}) \\
\supseteq & \{ \quad \text{naturality } \mathsf{str}_{A,B} \ : \ \mathsf{F}(A \times B) \hookleftarrow \mathsf{F}A \times B \quad \} \\
& \mathsf{F outl}^\circ_{A,B} \blacktriangle (\mathsf{str}_{A,B} \cdot \mathsf{F}!^\circ_A \times \mathsf{id}_B \cdot \mathsf{outr}^\circ_{\mathsf{F}1,B}) \\
= & \{ \quad \mathsf{F}! \text{ total, naturality of } \mathsf{outr} \quad \} \\
& \mathsf{F outl}^\circ_{A,B} \blacktriangle (\mathsf{str}_{A,B} \cdot \mathsf{outr}^\circ_{A,B}) \\
= & \{ \quad \text{modular identity, } \mathsf{str} \text{ function} \quad \} \\
& \mathsf{str}_{A,B} \cdot (\mathsf{str}^\circ_{A,B} \cdot \mathsf{F outl}^\circ_{A,B}) \blacktriangle \mathsf{outr}^\circ_{A,B} \\
= & \{ \quad \mathsf{str} \text{ strength: (5.17)} \quad \} \\
& \mathsf{str}_{A,B} \cdot \mathsf{outl}^\circ_{A,B} \blacktriangle \mathsf{outr}^\circ_{A,B} \\
= & \{ \quad \text{product} \quad \} \\
& \mathsf{str}_{A,B}
\end{array}
$$

Now, equality follows because from lemma's (6.9) and (6.10) it follows the the rhs of (6.15) is also a strength, hence, a function.
□

And the constructions of a fan or a strength from a copies map is injective,

Lemma 6.16 If cop is a copies map of relator F then,

$$\mathsf{cop}_A = \mathsf{F}!^\circ_A \blacktriangle (\mathsf{cop}_A \cdot \mathsf{outr}^\circ_{\mathsf{F}1,A})$$

Proof

$$
\begin{array}{ll}
& \mathsf{F}!^\circ_A \blacktriangle (\mathsf{cop}_A \cdot \mathsf{outr}^\circ_{\mathsf{F}1,A}) \\
= & \{ \quad \mathsf{cop} \text{ function, modular identity} \quad \} \\
& \mathsf{cop}_A \cdot (\mathsf{cop}^\circ_A \cdot \mathsf{F}!^\circ_A) \blacktriangle \mathsf{outr}^\circ_{\mathsf{F}1,A} \\
= & \{ \quad \text{copies map: } \mathsf{cop}^\circ_A \cdot \mathsf{F}!^\circ_A = \mathsf{outl}^\circ_{\mathsf{F}1,A}\text{, product} \quad \} \\
& \mathsf{cop}_A
\end{array}
$$

□

Finally, the constructions of a strength or a copies map from a fan is injective,

Lemma 6.17 If fan is a fan of relator F then,

$$\mathsf{fan}_A = \mathsf{F lid}_A \cdot \mathsf{zip}_{1,A} \cdot \mathsf{id}_{\mathsf{F}1} \times \mathsf{fan}_A \cdot \mathsf{outr}^\circ_{\mathsf{F}1,A}$$

Proof

$$
\begin{array}{ll}
& \mathsf{F lid}_A \cdot \mathsf{zip}_{1,A} \cdot \mathsf{id}_{\mathsf{F}1} \times \mathsf{fan}_A \cdot \mathsf{outr}^\circ_{\mathsf{F}1,A} \\
= & \{ \quad \mathsf{lid}_A = {!_A}^\circ \blacktriangle \mathsf{id}_A\text{, naturality } \mathsf{outr} \quad \}
\end{array}
$$

$$\begin{array}{ll}
& \mathsf{F}(!_A{}^\circ \blacktriangle \mathsf{id}_A) \cdot \mathsf{zip}_{1,A} \cdot \mathsf{outr}^\circ_{\mathsf{F}1,\mathsf{F}A} \cdot \mathsf{fan}_A \\
= & \quad \{ \quad \text{property } \mathsf{zip}\ (6.1) \quad \} \\
& \mathsf{F}!_A{}^\circ \blacktriangle \mathsf{Fid}_A \cdot \mathsf{outr}^\circ_{\mathsf{F}1,\mathsf{F}A} \cdot \mathsf{fan}_A \\
= & \quad \{ \quad \text{computation rule } \mathsf{outr},\ \mathsf{F}!_A \text{ total} \quad \} \\
& \mathsf{fan}_A
\end{array}$$

□

Corollary 6.18 For a binary intersection preserving relator F, the strengths, copies maps and fans are all in one-to-one correspondence.
□

6.4 Uniqueness of strength and copies map

Suppose relator F has membership mem. Then, as we have shown earlier, $\mathsf{mem}\backslash\mathsf{id}$ is a fan of F and, assuming the identification axiom, is indeed the *unique* fan of F. We called $\mathsf{mem}\backslash\mathsf{id}$ the "canonical fan" of F. Using the constructions of lemmas (6.13) and (6.10) we obtain a *canonical copies map*

$$\mathsf{F}!^\circ \blacktriangle \mathsf{fan}$$

and a *canonical strength*

$$\mathsf{zip} \cdot \mathsf{id} \times \mathsf{fan}$$

where fan denotes the canonical fan of relator F. The uniqueness of the canonical fan and the one-to-one correspondence between fans, strengths and copies maps guarantees that the canonical copies map and the canonical strength are also unique. The assumption is, however, that F preserves binary intersections. This assumption is unnecessary as we now show.

Lemma 6.19 For a relator F with membership, the canonical strength is the unique strength of F.

Proof We prove that the canonical strength is the largest strength of F. Then uniqueness follows because inclusion of functions is equality of functions. So, let θ be a strength of F and str the canonical strength of F. Then,

$$\begin{array}{ll}
& \theta_{A,B} \subseteq \mathsf{str}_{A,B} \\
\equiv & \quad \{ \quad \text{definition canonical strength, } \mathsf{zip}\text{, product} \quad \} \\
& \theta_{A,B} \subseteq \mathsf{Foutl}^\circ_{A,B} \blacktriangle (\mathsf{Foutr}^\circ_{A,B} \cdot \mathsf{fan}_B) \\
\equiv & \quad \{ \quad \blacktriangle \leftrightarrow \vartriangle\text{, universal propery product (2.52)} \quad \} \\
& \theta_{A,B} \cdot \mathsf{outl}^\circ_{\mathsf{F}A,B} \subseteq \mathsf{Foutl}^\circ_{A,B} \wedge \theta_{A,B} \cdot \mathsf{outr}^\circ_{\mathsf{F}A,B} \subseteq \mathsf{Foutr}^\circ_{A,B} \cdot \mathsf{fan}_B \\
\equiv & \quad \{ \quad \text{shunting of functions} \quad \}
\end{array}$$

$$
\begin{array}{ll}
& \mathsf{Foutl}_{A,B} \cdot \theta_{A,B} \subseteq \mathsf{outl}_{FA,B} \ \wedge\ \mathsf{Foutr}_{A,B} \cdot \theta_{A,B} \cdot \mathsf{outr}^{\circ}_{FA,B} \subseteq \mathsf{fan}_B \\
\Leftarrow & \{ \quad \text{property (5.17), } \mathsf{fan}_B \text{ largest nat. trans. of type } FB \hookleftarrow B \quad \} \\
& \mathsf{Foutr}_{A,B} \cdot \theta_{A,B} \cdot \mathsf{outr}^{\circ}_{FA,B} \ :\ FB \hookleftarrow B \\
\equiv & \{ \quad \text{naturality } \mathsf{outr}_{A,\_},\ \theta_{A,\_},\ \mathsf{outr}^{\circ}_{FA,\_} \quad \} \\
& \mathit{true}
\end{array}
$$

□

Note, however, that in the proof of lemma (6.19) we made only use of the fact that $\theta_{A,B}$ is a functional natural transformation and property (5.17) which followed from the coherence property with rid. In other words we have proven,

Corollary 6.20 If relator F has membership and $\theta_{A,B}$ is a functional natural transformation of type $F(A \times B) \hookleftarrow FA \times B$ and is coherent with rid, i.e.

$$\mathsf{Frid}_A \cdot \theta_{A,1} = \mathsf{rid}_{FA}$$

then $\theta_{A,B}$ is the unique strength of relator F.
□

The importance of corollary (6.20) is that it provides an easy way to check whether a constructed function is a strength for a specific datatype. For instance, it follows that

$$
\begin{array}{rcl}
\mathsf{str}([\,], b) & = & [\,] \\
\mathsf{str}(\mathsf{cons}(a, x), b) & = & \mathsf{cons}((a, b), \mathsf{str}(x, b))
\end{array}
$$

defines a strength for (cons) lists. First, since str is a polymorphic function it follows from “Theorems for Free!” that str is a natural transformation. Secondly, it is not difficult to verify that str preserves the original list in the left-argument of all the pairs. Hence, without a proof we know that str is coherent with respect to ass and thus str is a strength of the datatype list.

Just as for strength, we can prove uniqueness of copies map,

Lemma 6.21 For a relator F with membership, the canonical copies map is the unique copies map of F.

Proof Just as for the canonical strength, we prove that the canonical copies map is the largest copies map of F. So, let γ be a copies map and cop the canonical copies map of F, then,

$$
\begin{array}{ll}
& \gamma_A \subseteq \mathsf{cop}_A \\
\equiv & \{ \quad \text{canonical copies map} \quad \} \\
& \gamma_A \subseteq F!^{\circ}_A \blacktriangle \mathsf{fan}_A \\
\equiv & \{ \quad \blacktriangle \leftrightarrow \vartriangle, \text{ universal propery product (2.52)} \quad \} \\
& \gamma_A \cdot \mathsf{outl}^{\circ}_{F1,A} \subseteq F!^{\circ}_A \wedge \gamma_A \cdot \mathsf{outr}^{\circ}_{F1,A} \subseteq \mathsf{fan}_A \\
\equiv & \{ \quad \text{shunting of functions} \quad \}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathsf{F!}_A \cdot \gamma_A \subseteq \mathsf{outl}_{F1,A} \;\wedge\; \gamma_A \cdot \mathsf{outr}^{\circ}_{F1,A} \subseteq \mathsf{fan}_A \\
\Leftarrow & \quad \{ \quad \text{copies map: } \mathsf{F!}_A \cdot \gamma_A = \mathsf{outl}_{F1,A}, \\
 & \qquad\quad \mathsf{fan}_A \text{ largest nat. trans. of type } FA \hookleftarrow A \;\; \} \\
 & \gamma_A \cdot \mathsf{outr}^{\circ}_{F1,A} \;:\; FA \hookleftarrow A \\
\equiv & \quad \{ \quad \gamma_A \;:\; FA \hookleftarrow F1 \times A, \mathsf{outr}^{\circ}_{F1,A} \;:\; F1 \times A \leftarrow A \;\; \} \\
 & \mathit{true}
\end{array}
$$

□

The proofs of lemma's (6.19) and (6.21) are nice examples of the power of the calculus of largest natural transformations. It simplifies many arguments for generic constructs like strength and copies map. Another beautiful example is the following, concerning so-called *strong* natural transformations,

Definition 6.22 Let F and G be strong relators, with strength $\mathsf{str}.F$ and $\mathsf{str}.G$ respectively. A natural transformation $\alpha : F \hookleftarrow G$ is said to be *strong* if

$$
\begin{tikzcd}
FA \times B \arrow[d, "(\mathsf{str}.F)_{A,B}"'] & GA \times B \arrow[l, "\alpha_A \times \mathsf{id}_B"'] \arrow[d, "(\mathsf{str}.G)_{A,B}"] \\
F(A \times B) & G(A \times B) \arrow[l, "\alpha_{A \times B}"]
\end{tikzcd}
$$

commutes.
□

Surprisingly, this condition is always satisfied if relator F has membership,

Lemma 6.23 Let F and G be relators with membership. Then any $\alpha : F \hookleftarrow G$ is strong.

Proof Let $\mathsf{str}.F$ and $\mathsf{str}.G$ denote the unique, canonical strengths of relators F and G respectively; then we have to prove:

$$(\mathsf{str}.F)_{A,B} \cdot \alpha_A \times \mathsf{id}_B = \alpha_{A \times B} \cdot (\mathsf{str}.G)_{A,B}$$

The containment $\supseteq$ follows from the theory of commuting relators since the canonical strength $(\mathsf{str}.F)_{A,B}$ is the half-zip $(\mathsf{zip}.(\times B).F)_A$. For the other containment we reason as follows:

$$
\begin{array}{ll}
 & (\mathsf{str}.F)_{A,B} \cdot \alpha_A \times \mathsf{id}_B \subseteq \alpha_{A \times B} \cdot (\mathsf{str}.G)_{A,B} \\
\equiv & \quad \{ \quad \mathsf{str} \text{ function, shunting } \;\; \} \\
 & \alpha_A \times \mathsf{id}_B \cdot (\mathsf{str}.G)^{\circ}_{A,B} \subseteq (\mathsf{str}.F)^{\circ}_{A,B} \cdot \alpha_{A \times B} \\
\equiv & \quad \{ \quad \text{converse } \;\; \} \\
 & (\mathsf{str}.G)_{A,B} \cdot \alpha^{\circ}_A \times \mathsf{id}_B \subseteq \alpha^{\circ}_{A \times B} \cdot (\mathsf{str}.F)_{A,B}
\end{array}
$$

We continue with the lhs:

$$
\begin{array}{ll}
& (\mathsf{str.G})_{A,B} \cdot \alpha^{\circ}_{A} \times \mathsf{id}_B \\
= & \{ \quad \text{canonical strength, product} \quad \} \\
& (\mathsf{Goutl}^{\circ}_{A,B} \cdot \alpha^{\circ}_{A}) \blacktriangle (\mathsf{Goutr}^{\circ}_{A,B} \cdot (\mathsf{fan.G})_B) \\
\subseteq & \{ \quad \alpha : \mathsf{F} \hookleftarrow \mathsf{G} \quad \} \\
& (\alpha^{\circ}_{A \times B} \cdot \mathsf{Foutl}^{\circ}_{A,B}) \blacktriangle (\mathsf{Goutr}^{\circ}_{A,B} \cdot (\mathsf{fan.G})_B) \\
\subseteq & \{ \quad \text{modular law} \quad \} \\
& \alpha^{\circ}_{A \times B} \cdot \mathsf{Foutl}^{\circ}_{A,B} \blacktriangle (\alpha_{A \times B} \cdot \mathsf{Goutr}^{\circ}_{A,B} \cdot (\mathsf{fan.G})_B) \\
\subseteq & \{ \quad \alpha : \mathsf{F} \hookleftarrow \mathsf{G} \quad \} \\
& \alpha^{\circ}_{A \times B} \cdot \mathsf{Foutl}^{\circ}_{A,B} \blacktriangle (\mathsf{Foutr}^{\circ}_{A,B} \cdot \alpha_B \cdot (\mathsf{fan.G})_B) \\
\subseteq & \{ \quad \alpha \cdot \mathsf{fan.G} : \mathsf{F} \hookleftarrow \mathrm{Id}, \text{ and } \mathsf{fan.F} \text{ largest of that type} \quad \} \\
& \alpha^{\circ}_{A \times B} \cdot \mathsf{Foutl}^{\circ}_{A,B} \blacktriangle (\mathsf{Foutr}^{\circ}_{A,B} \cdot (\mathsf{fan.F})_B) \\
= & \{ \quad \text{canonical strength} \quad \} \\
& \alpha^{\circ}_{A \times B} \cdot (\mathsf{str.F})_{A,B}
\end{array}
$$

□

As said before, we find this result is quite surprising. It gives some confidence that the definition of a datatype as “relator with membership” [20] is a proper one. First of all, a relator with membership has a strength, i.e. is strong, which Moggi [39] regards fundamental to computation. Secondly, all natural transformations between two relators with membership are strong too.

6.5 Shapely functors

Barry Jay [23] defines the notion of a *shapely* functor and the notion of *copyable* natural transformation between *shapely* functors. By definition a *shapely* functor is a so-called pullback preserving functor equipped with a copies map. In [40], Oege de Moor shows that a relator preserves pullbacks precisely when it preserves binary intersections. In other words, it follows that a binary intersection preserving relator with membership is shapely. Hence, all regular relators are shapely.

A natural transformation $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$ is said to be *copyable* if it is coherent with respect to the copies maps of F and G. That is to say, if the following diagram commutes,

$$
\begin{tikzcd}
\mathsf{F}1 \times A \arrow[d, "(\mathsf{cop.F})_A"'] & \mathsf{G}1 \times A \arrow[l, "\alpha_1 \times \mathsf{id}_A"'] \arrow[d, "(\mathsf{cop.G})_A"] \\
\mathsf{F}A & \mathsf{G}A \arrow[l, "\alpha_A"]
\end{tikzcd}
$$

It follows trivially that,

Lemma 6.24 Let F and G be relators with membership. Then any $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$ is copyable.

Proof Follows directly from lemma (6.23). We have for the unique, canonical copies map $(\mathsf{cop.F})_A = \mathsf{Foutr}_{1,A} \cdot (\mathsf{str.F})_{1,B}$. Using this, we prove

$$
\begin{array}{ll}
 & (\mathsf{cop.F})_A \cdot \alpha_1 \times \mathsf{id}_A = \alpha_A \cdot (\mathsf{cop.G})_A \\
\equiv & \quad \{ \quad \text{remark above} \quad \} \\
 & \mathsf{Foutr}_{1,A} \cdot (\mathsf{str.F})_{1,B} \cdot \alpha_1 \times \mathsf{id}_A = \alpha_A \cdot \mathsf{Goutr}_{1,A} \cdot (\mathsf{str.G})_{1,B} \\
\Leftarrow & \quad \{ \quad \alpha_A \cdot \mathsf{Goutr}_{1,A} = \mathsf{Foutr}_{1,A} \cdot \alpha_{1\times A}, \text{Leibniz} \quad \} \\
 & (\mathsf{str.F})_{1,B} \cdot \alpha_1 \times \mathsf{id}_A = \alpha_{1\times A} \cdot (\mathsf{str.G})_{1,B} \\
\equiv & \quad \{ \quad \text{lemma (6.23)} \quad \} \\
 & \textit{true}
\end{array}
$$

□

6.6 Coherence versus largest natural transformations

In the previous section we showed that every natural transformation of type $\mathsf{F} \hookleftarrow \mathsf{G}$ is coherent with respect to the corresponding strengths and copies maps. Both facts we proved using the fact that the canonical fan is the largest natural transformation of its type. Similarly, we have that a membership relation is the largest natural transformation of its type. In this section we show that the property "largest natural transformation of its type" for fans and membership can be expressed as a coherence requirement as well. Furthermore, this coherence requirement can be expressed just as for $\mathsf{zip.F}$ as a higher-order naturality requirement.

We start with membership. We consider the collection of membership relations $\mathsf{mem.F}$ indexed by a relator F. From the fact that $\mathsf{fan.G}$ is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \mathsf{G}$ it follows that for each $\alpha : \mathsf{F} \hookleftarrow \mathsf{G}$,

$$\mathsf{mem.F} \cdot \alpha \subseteq \mathsf{mem.G} \tag{6.25}$$

since $\mathsf{mem.F} \cdot \alpha$ has type $\mathrm{Id} \hookleftarrow \mathsf{G}$. Furthermore, it follows that

$$\mathsf{mem.Id} = \mathsf{id} \tag{6.26}$$

since the identification axiom states that id is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \mathrm{Id}$.

So, equations (6.25) and (6.26) are a consequence of the fact that membership is the largest natural transformation. The other way around is also true. Assume equations (6.25) and (6.26). Then, for each $\alpha : \mathrm{Id} \hookleftarrow \mathsf{F}$,

$$
\begin{array}{ll}
 & (6.25) \\
\Rightarrow & \{ \quad \mathsf{F} := \mathrm{Id},\ \mathsf{G} := \mathsf{F} \quad \} \\
 & \mathsf{mem}.\mathrm{Id} \cdot \alpha \subseteq \mathsf{mem}.\mathsf{F} \\
\equiv & \{ \quad (6.26)\text{: } \mathsf{mem}.\mathrm{Id} = \mathsf{id} \quad \} \\
 & \alpha \subseteq \mathsf{mem}.\mathsf{F}
\end{array}
$$

Hence, $\mathsf{mem}.\mathsf{F}$ is the largest natural transformation of type $\mathrm{Id} \hookleftarrow \mathsf{F}$.

Next, we want to express requirement (6.25) as a higher order naturality property. We do this in the same way as we derived the higher order naturality property of $\mathsf{zip}.\mathsf{F}$. In order to see in what sense $\mathsf{mem}.\mathsf{F}$ is natural in F let us denote it according to our convention for denoting natural transformations. Specifically, we write mem_F instead of $\mathsf{mem}.\mathsf{F}$. Thus mem is a collection of arrows indexed by a relator. Now observe that the naturality of $\mathsf{mem}_\mathsf{F} : \mathrm{Id} \hookleftarrow \mathsf{F}$ takes the form

$$\mathsf{mem}_\mathsf{F} \ : \ \mathsf{K}_{\mathrm{Id}}(\mathsf{F}) \hookleftarrow \mathit{ID}(\mathsf{F}) \tag{6.27}$$

where K_{Id} denotes the constant mapping which maps each relator to the identity relator, and *ID* denotes the identity mapping on relators. If the collection mem is itself a natural transformation then the mappings K_{Id} and *ID* are the object maps of the functors between which mem is a natural transformation. So, what are the corresponding arrow maps of these functors? The obvious choice is to define for natural transformation α, $\mathsf{K}_{\mathrm{Id}}(\alpha) = \mathsf{id}$ and $\mathit{ID}(\alpha) = \alpha$. It is trivially verified that this defines two functors on the functor category.

Now, typing (6.27) of mem_F for each F suggest the following "free theorem": mem is a higher order natural transformation with typing:

$$\mathsf{mem} : \mathsf{K}_{\mathrm{Id}} \leftarrow \mathit{ID} \tag{6.28}$$

Expanding the definition of a natural transformation gives us:

$$\mathsf{K}_{\mathrm{Id}}(\alpha) \cdot \mathsf{mem}_\mathsf{G} \ = \ \mathsf{mem}_\mathsf{F} \cdot \mathit{ID}(\alpha) \quad \text{for each } \alpha : \mathsf{F} \leftarrow \mathsf{G}.$$

And using the definition of K_{Id} and *ID* gives,

$$\mathsf{mem}_\mathsf{G} \ = \ \mathsf{mem}_\mathsf{F} \cdot \alpha \quad \text{for each } \alpha : \mathsf{F} \leftarrow \mathsf{G}. \tag{6.29}$$

Just as equation (5.9) for $\mathsf{zip}.\mathsf{F}$, equation (6.29), motivated by type considerations, is not precisely property (6.25). However, if $\alpha : \mathsf{F} \leftarrow \mathsf{G}$ is a function we can prove that $\mathsf{mem}_\mathsf{F} \cdot \alpha$ is a membership relation of G, hence, equation (6.29) follows from uniqueness of membership. For an arbitrary natural transformation, demanding equality is too severe. If we take for instance $\alpha_\mathsf{A} := \perp\!\!\!\perp_{\mathsf{FA},\mathsf{GA}}$, we have $\mathsf{mem}_\mathsf{F} \cdot \perp\!\!\!\perp_{\mathsf{FA},\mathsf{GA}} = \perp\!\!\!\perp_{\mathsf{A},\mathsf{GA}}$ and in general it is the case that $\mathsf{mem}_\mathsf{G} \neq \perp\!\!\!\perp_{\mathsf{A},\mathsf{GA}}$. So, if we want to include arbitrary natural transformations we have to relax requirement (6.29) using an inclusion instead of an equality. Having an inclusion, the requirement for α can be relaxed as well. So, equation (6.29) becomes,

$$\mathsf{mem}_\mathsf{G} \supseteq \mathsf{mem}_\mathsf{F} \cdot \alpha \quad \text{for each } \alpha : \mathsf{F} \hookleftarrow \mathsf{G},$$

which is equation (6.25). In other words, the fact that a membership relation for each relator is the largest natural transformation of its type can more or less be expressed by:

$$\mathsf{mem} : \mathsf{K}_{\mathrm{Id}} \leftarrow ID \quad \text{and} \quad \mathsf{mem}_{\mathrm{Id}} = \mathsf{id} \ .$$

Just as for membership, we can derive that a fan for each relator is the largest natural transformation of its type can be expressed by,

$$\mathsf{fan} : ID \leftarrow \mathsf{K}_{\mathrm{Id}} \quad \text{and} \quad \mathsf{fan}_{\mathrm{Id}} = \mathsf{id} \ .$$

So, we have shown that membership and fan are the largest natural transformations of their types followed from considering their "free property", a higher-order naturality property, which we derived from the typing of the individual members $\mathsf{mem.F}$ and $\mathsf{fan.F}$.

6.7 Conclusion

In this chapter we proved that for a binary intersection preserving relator the notions of fan, copies map and strength coincide. Furthermore, we proved that for a relator with membership, not necessarily binary intersection preserving, these three concepts are uniquely defined. The copies map and strength are based on a functional paradigm. Since we are working in a relational framework we are able to define a notion like the canonical fan. The notion of a fan is a much simpler concept, despite the nondeterminism involved, than strength and somewhat simpler than copies map. For all three concepts we can express a coherence property: for a copies map the notion of a copyable natural transformation, for strength the notion of a strong natural transformation. For fans, the coherence property is just the fact that the canonical fan is the largest fan of its type, and this fact is predicted by the free theorem derived from the typing of fans.

As we have seen, the calculus of largest natural transformations is a powerful tool. We proved the existence and uniqueness of strength and copies map, and we proved that every natural transformation is strong and copyable. In other words, for a natural transformation between relators with membership the requirements "being strong" or "being copyable" are superfluous.

It is of course quite likely that we have missed out a number of generic operations on datatypes, i.e. operations that are common to all datatypes. It remains to be seen whether these can be constructed using membership. However, we have shown that this is indeed the cases for fans, strengths and copies maps.

Chapter 7

Epilogue

In this thesis, we have presented the basis of a theory of datatypes that is truly generic in a mathematically precise sense. We have formulated a precise, verifiable definition of the notion of a polytypic function. Specifically, a polytypic function should satisfy the higher-order naturality property predicted by its type. The higher-order naturality property captures the informal notion that a polytypic function instantiated at different type constructors should behave in related ways.

We have given a generic characterization of a membership relation and shown that a membership relation is the largest natural transformation of its type. The calculus of largest natural transformations was shown to be very powerful. In particular, using this calculus we were able to prove several uniqueness results. For instance, for a relator F with membership there exists a unique strength. We showed that the property "being a largest of its type" can be formulated as a higher-order naturality property. This higher-order naturality is predicted by the type of a membership relation. So, indeed the notion of a membership relation is truly polytypic. The same holds for fans, the notion dual to membership. Again, fans are the largest natural transformations of their type, and this property can be formulated as a higher-order naturality property.

The notion of higher-order naturality played a central rôle in the discussion of commuting datatypes and the construction of the candidate zips. Normally, the type of a natural transformation does not imply uniqueness. Many different natural transformations of the same type can exist. However, in the case of higher-order naturality we were able to establish several uniqueness results. The higher-order naturality of the zip operation implies a slok-property. In case of coproduct, product and projections this slok-property has at most one solution and predicts the only possible candidates for the corresponding zip operation. Furthermore, the uniqueness of a solution of a slok-property implies that the corresponding zip operations are compositional. We got this result because we proved that we have the same typing rules for the slok-category as for the functor category. In other words, a slok-property is preserved by the normal constructions on natural transformations. Although the theory of slok-properties seems trivial it provided us with a powerful tool for constructing polytypic functions and establishing their properties.

The discovery of the higher-order naturality property is a major advance on our earlier work [4] in which the commuting requirement was substantially more operational in flavour and hence *ad hoc*. In our earlier work we only required the slok-properties expressing the requirement that a zip operation should preserve the shape of the data structures. It was a pleasant surprise to us that the higher-order naturality of the zip operations implied the slok-properties. So, the preservation of shape was a consequence of the higher-order naturality requirement.

We showed that a special class of zips, the broadcast functions, coincide with the notion of a strength — a notion which according to Moggi [39] is fundamental to computation. Furthermore, we proved that this strength is unique and that every natural transformation between two relators with membership is coherent with the corresponding strengths. Similarly, we showed that a relator with membership has a unique copies map and all natural transformations between relators with membership are coherent with their corresponding copies maps. In other words, for a natural transformation between relators with membership the requirements "being strong" or "being copyable" are superfluous.

A new development in this thesis is that we have considered non-endo relators as well. We have demonstrated how to cope cleanly with non-endo relators thus overcoming a limitation of all other work in this field published to date that we know of (including our own).

Our goal has been to calculate programs: that is, to construct programs just by syntactic manipulation instead of being led by operational interpretations and giving informal arguments for their correctness. The calculus we have used is the calculus of relations extended with a few categorical concepts. However, this thesis is not on the calculus of relations nor on category theory. We have used the calculus of relations and category theory only as a mathematical language in order to express our notions and theorems in a precise and concise way.

Using this calculus we were able to achieve an optimum level of abstraction enabling us to define generic concepts and reason generically about those concepts. An immediate benefit is a substantial reduction in the burden of proof. An example is the generic definition of a datatype. Were we only able to define the notion of a datatype by giving an inductive definition we would be obliged to give an inductive proof every time we want to prove a property for all datatypes. However, we defined a datatype as a relator with membership and using this definition we were able to prove in one go that all regular relators have a strength. This in contrast to, for example, Tuijnman [51] who established a similar result by giving an inductive definition for the class of regular datatypes. Another example is structure multiplication. Lillie proved this property but only in the case of lists and with the aid of a very long proof cluttered with case distinctions between the two constructors nill and cons of the datatype list. Since we formulated the problem at a higher-level of abstraction we were able to give one proof for *all* datatypes at once, and at the same time we were not hindered by unnecessary details.

Several challenges remain. For instance, having a number of generic, or polytypical concepts such as catamorphism — and its friends anamorphism, hylomorphism etc. —, zips, fans, strengths and copies map, one may wonder: are there others? And more importantly, can they be expressed using the existing polytypical building blocks mentioned earlier? Or is it the case that there are other fundamental polytypical notions yet to be discovered which we can then add to our repertoire of generic constructions?

We have argued that a polytypic function should be higher-order natural. So, the question remains: are all the polytypic functions considered by Jeuring and others [27, 28, 35] higher-order natural, and what does the higher-order naturality look like in each of the individual cases? For instance, Jeuring defines a polytypic size function $(\mathsf{size}.\mathsf{F})_\mathsf{A}$ of type $\mathit{Nat} \leftarrow \mathsf{FA}$. The intended interpretation of the size function is that it returns the number of elements stored in a data structure. So, the "free theorem" suggested by the type of size.F is,

$$\mathsf{size}.\mathsf{G} = \mathsf{size}.\mathsf{F} \cdot \alpha \quad \Leftarrow \quad \alpha : \mathsf{F} \leftarrow \mathsf{G} \ . \tag{7.1}$$

Indeed, we would expect this property for functional and *proper* natural transformation α since the interpretation of a *proper* natural transformation is that the number of elements remains the same[1]. It is not the case that (7.1) completely characterizes the size function. A question is thus, what are the other requirements for the size function such that we can characterize — possible uniquely — the polytypic size function? If we do not have a non-inductive characterization of the size function it will be very complicated to prove the higher-order naturality requirement for size. For instance, if we only have an inductive definition of the size function we have to give a proof by induction on two levels: one for F and one for G. This would yield an explosion in the number of cases to be considered.

Another challenge is directly connected with the previous problem and has to do with unicity. A major frustration is that we have been unable to establish a general unicity property of the "zip" operators even though in every individual case that we have studied we can prove unicity. This suggests that our requirements can be made stronger and, in the process, yet simpler and more elegant. The general question is: is it always possible to derive some uniqueness properties for a polytypic program from its higher-order naturality property?

Broader questions concern how the notion of polytypy relates to other notions of generic programming. Design patterns [18] have recently attracted substantial interest as a way of categorizing (and reusing) common building blocks in object-oriented programming but we are unable to comment at this stage on whether there is any relation between this work and our own. There does appear, however, to be a close connection to the notion of "adaptive" object-oriented programming [30] which aims to abstract away from the inheritance structure of an object-oriented program. The field of generic programming is new and initial progress likely to be slow but the challenge is worthwhile and inviting.

[1] Note that we have not been able to give a formal proof of this interpretation. We were only able to prove that the *set* of elements remains the same which does not necessarily imply that the *number* of elements is preserved.

Appendix A

Typing rules

A.1 The τ-Δ calculus

$$\Delta_k f_k \in \mathcal{C}^k \;\Leftarrow\; f_k \in \mathcal{C} \;,$$
$$\Delta_k F_k : \mathcal{C}^k \leftarrow \mathcal{D} \;\Leftarrow\; F_k : \mathcal{C} \leftarrow \mathcal{D} \;,$$
$$\Pi_k F_k : \mathcal{C}^k \leftarrow \mathcal{D}^k \;\Leftarrow\; F_k : \mathcal{C} \leftarrow \mathcal{D} \;,$$
$$F^k : \mathcal{C}^k \leftarrow \mathcal{D}^k \;\Leftarrow\; F : \mathcal{C} \leftarrow \mathcal{D} \;.$$

$$\Delta_k F_k \;:\; m{*}k \leftarrow n \;\Leftarrow\; F_k : m \leftarrow n \;,$$
$$\Pi_k F_k \;:\; m{*}k \leftarrow n{*}k \leftarrow n \;\Leftarrow\; F_k : m \leftarrow n \;,$$
$$F^k \;:\; m{*}k \leftarrow n{*}k \;\Leftarrow\; F : m \leftarrow n \;,$$
$${}^kF \;:\; k{*}m \leftarrow k{*}n \;\Leftarrow\; F : m \leftarrow n \;.$$

$$\Delta_k \alpha_k \;:\; \Delta_k F_k \leftarrow \Delta_k G_k \;\Leftarrow\; \alpha_k : F_k \leftarrow G_k \;,$$
$$\Pi_k \alpha_k \;:\; \Pi_k F_k \leftarrow \Pi_k G_k \;\Leftarrow\; \alpha_k : F_k \leftarrow G_k \;,$$
$$\alpha^k : F^k \leftarrow G^k \;\Leftarrow\; \alpha : F \leftarrow G \;,$$
$${}^k\alpha : {}^kF \leftarrow {}^kG \;\Leftarrow\; \alpha : F \leftarrow G \;.$$

A.2 Membership and fans

$$\mathsf{mem}.F : \mathrm{Id} \hookleftarrow F \;,\; \mathsf{mem}.F : 1 \leftarrow 1 \;\Leftarrow\; F : 1 \leftarrow 1 \;,$$
$$\mathsf{mem}.F \;:\; \mathrm{Id} \hookleftarrow \Delta_k F \;,\; \mathsf{mem}.F : k \leftarrow k \;\Leftarrow\; F : 1 \leftarrow k \;,$$
$$\mathsf{mem}.F \;:\; (\Delta_l)^k \hookleftarrow \Delta_k F \;,\; \mathsf{mem}.F \;:\; l{*}k \leftarrow k \;\Leftarrow\; F : l \leftarrow k \;.$$

$$\mathsf{fan}.F : F \hookleftarrow \mathrm{Id} \;,\; \mathsf{fan}.F : 1 \leftarrow 1 \;\Leftarrow\; F : 1 \leftarrow 1 \;,$$
$$\mathsf{fan}.F \;:\; F\Delta_k \hookleftarrow \mathrm{Id} \;,\; \mathsf{fan}.F : 1 \leftarrow 1 \;\Leftarrow\; F : 1 \leftarrow k \;,$$
$$\mathsf{fan}.F \;:\; F\Delta_k \hookleftarrow \Delta_l \;,\; \mathsf{fan}.F : l \leftarrow 1 \;\Leftarrow\; F : l \leftarrow k \;.$$

A.3 Zips

$$\mathsf{zip.F.G} : \mathsf{GF} \leftarrow \mathsf{FG} \quad , \quad \mathsf{zip.F.G} : 1 \leftarrow 1 \quad \Leftarrow \quad \mathsf{F} : 1 \leftarrow 1 \; , \; \mathsf{G} : 1 \leftarrow 1 \;\; ,$$

$$\mathsf{zip.F.G} : \mathsf{G}({}^{l}\mathsf{F}) \leftarrow \mathsf{F}(\mathsf{G}^{k}) \quad , \quad \mathsf{zip.F.G} : 1 \leftarrow l{*}k \quad \Leftarrow \quad \mathsf{F} : 1 \leftarrow k \; , \; \mathsf{G} : 1 \leftarrow l \;\; ,$$

$$\mathsf{zip.F.G} : (\mathsf{G}^{m})({}^{l}\mathsf{F}) \leftarrow ({}^{n}\mathsf{F})(\mathsf{G}^{k}) \quad , \quad \mathsf{zip.F.G} : n{*}m \leftarrow l{*}k \quad \Leftarrow \quad \mathsf{F} : m \leftarrow k \; , \; \mathsf{G} : n \leftarrow l \;\; .$$

Bibliography

[1] C.J. Aarts, R.C. Backhouse, P. Hoogendijk, T.S. Voermans, and J. van der Woude. A relational theory of datatypes. Available via World-Wide Web at `http://www.win.tue.nl/win/cs/wp/papers`, September 1992.

[2] R.C. Backhouse, P. de Bruin, P. Hoogendijk, G. Malcolm, T.S. Voermans, and J. van der Woude. Polynomial relators. In M. Nivat, C.S. Rattray, T. Rus, and G. Scollo, editors, *Proceedings of the 2nd Conference on Algebraic Methodology and Software Technology, AMAST'91*, pages 303–326. Springer-Verlag, Workshops in Computing, 1992.

[3] R.C. Backhouse, P. de Bruin, G. Malcolm, T.S. Voermans, and J. van der Woude. Relational catamorphisms. In Möller B., editor, *Proceedings of the IFIP TC2/WG2.1 Working Conference on Constructing Programs from Specifications*, pages 287–318. Elsevier Science Publishers B.V., 1991.

[4] R.C. Backhouse, H. Doornbos, and P. Hoogendijk. Commuting relators. Available via World-Wide Web at `http://www.win.tue.nl/win/cs/wp/papers`, September 1992.

[5] R.C. Backhouse and P. Hoogendijk. Elements of a relational theory of datatypes. In B. Möller, H.A. Partsch, and S.A. Schuman, editors, *Formal Program Development. Proc. IFIP TC2/WG 2.1 State of the Art Seminar, Rio de Janeiro, Jan. 1992*, volume 755 of *LNCS*, pages 7–42. Springer-Verlag, 1993.

[6] Richard Bird, Oege de Moor, and Paul Hoogendijk. Generic functional programming with types and relations. *J. of Functional Programming*, 6(1):1–28, January 1996.

[7] Richard S. Bird and Oege de Moor. *Algebra of Programming*. Prentice-Hall International, 1996.

[8] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design*. Springer-Verlag, 1987. NATO ASI Series, vol. F36.

[9] R.S. Bird and P. Wadler. *Introduction to Functional Programming*. Prentice-Hall, 1988.

[10] J.H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, 1971.

[11] R.P. Dilworth. Non-commutative residuated lattices. *Transactions of the American Mathematical Society*, 46:426–444, 1939.

[12] H. Doornbos. *Reductivity arguments and program construction*. PhD thesis, Eindhoven University of Technology, Department of Mathematics and Computing Science, June 1996.

[13] Henk Doornbos and Roland Backhouse. Induction and recursion on datatypes. In B. Möller, editor, *Mathematics of Program Construction, 3rd International Conference*, volume 947 of *LNCS*, pages 242–256. Springer-Verlag, July 1995.

[14] Henk Doornbos and Roland Backhouse. Reductivity. *Science of Computer Programming*, 26(1–3):217–236, 1996.

[15] Achim Jung (Editor). Domains and denotational semantics: History, accomplishments and open problems. *Bulletin of the European Association for Computer Science*, 59:227–256, June 1996.

[16] Maarten M. Fokkinga. *Law and Order in Algorithmics*. PhD thesis, Universiteit Twente, The Netherlands, 1992.

[17] P.J. Freyd and A. Scedrov. *Categories, Allegories*. North-Holland, 1990.

[18] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Mass., 1995.

[19] C.A.R. Hoare and Jifeng He. The weakest prespecification. *Fundamenta Informaticae*, 9:51–84, 217–252, 1986.

[20] Paul Hoogendijk and Oege de Moor. What is a datatype? Technical Report 96/16, Department of Mathematics and Computing Science, Eindhoven University of Technology, 1996. Submitted to Science of Computer Programming. Available via World-Wide Web at `http://www.win.tue.nl/win/cs/wp/papers`.

[21] P. Jansson. Polytypism and polytypic unification. Master's thesis, Chalmers University of Technology and University of Göteborg, 1995.

[22] P. Jansson and J. Jeuring. Polyp - a polytypic programming language extension. In *POPL '97: The 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 470–482. ACM Press, 1997.

[23] C.B. Jay. Matrices, monads and the fast fourier transform. Technical Report UTS-SOCS-93.13, University of Technology, Sydney, 1993.

[24] C.B. Jay. Polynomial polymorphism. In R. Kotagiri, editor, *Proceedings of the Eighteenth Australasian Computer Science Conference: Glenelg, South Australia 1–3 February, 1995*, volume 17, pages 237–243. A.C.S. Communications, 1995.

[25] C.B. Jay. A semantics for shape. *Science of Computer Programming*, 25:251–283, 1995.

[26] C.B. Jay and J.R.B. Cockett. Shapely types and shape polymorphism. In D. Sannella, editor, *Programming Languages and Systems - ESOP '94: 5th European Symposium on Programming, Edinburgh, U.K., April 1994, Proceedings*, Lecture Notes in Computer Science, pages 302–316. Springer Verlag, 1994.

[27] J. Jeuring. Polytypic pattern matching. In *Conference Record of FPCA '95, SIGPLAN-SIGARCH-WG2.8 Conference on Functional Programming Languages and Computer Architecture*, pages 238–248, 1995.

[28] J. Jeuring and P. Jansson. Polytypic programming. In J. Launchbury, E. Meijer, and T. Sheard, editors, *Proceedings of the Second International Summer School on Advanced Functional Programming Techniques*, pages 68–114. Springer-Verlag, 1996. LNCS 1129.

[29] J. Lambek. A fixpoint theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

[30] Karl J. Lieberherr, Ignacio Silva-Lepe, and Cun Xiao. Adaptive object-oriented programming using graph-based customization. *Comm.A.C.M.*, 37(5):94–101, May 1994.

[31] G. Malcolm. Homomorphisms and promotability. In J.L.A. van de Snepscheut, editor, *Conference on the Mathematics of Program Construction*, pages 335–347. Springer-Verlag LNCS 375, 1989.

[32] G. Malcolm. *Algebraic data types and program transformation*. PhD thesis, Groningen University, 1990.

[33] G. Malcolm. Data structures and program transformation. *Science of Computer Programming*, 14(2–3):255–280, October 1990.

[34] L. Meertens. Algorithmics – towards programming as a mathematical activity. In *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289–334. North-Holland, 1986.

[35] Lambert Meertens. Calculate polytypically! In Herbert Kuchen and S. Doaitse Swierstra, editors, *Proceedings of the Eighth International Symposium PLILP '96 Programming Languages: Implementations, Logics and Programs*, volume 1140 of *Lecture Notes in Computer Science*, pages 1–16. Springer Verlag, 1996.

[36] E. Meijer, M.M. Fokkinga, and R. Paterson. Functional programming with bananas, lenses, envelopes and barbed wire. In *FPCA91: Functional Programming Languages and Computer Architecture*, volume 523 of *LNCS*, pages 124–144. Springer-Verlag, 1991.

[37] R. Milner. A theory of type polymorphism in programming. *J. Comp. Syst. Scs.*, 17:348–375, 1977.

[38] R. Milner. The standard ML core language. *Polymorphism*, II(2), October 1985.

[39] E. Moggi. Notions of computation and monads. *Information and Computation*, 93(1):55–92, 1991.

[40] O. de Moor. *Categories, Relations and Dynamic Programming*. PhD thesis, Oxford University Laboratory, Programming Research Group, April 1992.

[41] Gordon D. Plotkin. Lambda-definability in the full type hierarchy. In J.P. Seldin and J.R. Hindley, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, London, 1980.

[42] B. Randell, G. Ringland, and W.A. Wulf (Eds.). *Software 2000: A View of the Future*. European Commission, Brussels, 1994.

[43] J.C. Reynolds. Types, abstraction and parametric polymorphism. In R.E. Mason, editor, *IFIP '83*, pages 513–523. Elsevier Science Publishers, 1983.

[44] F.J. Rietman. A note on extensionality. In J. van Leeuwen, editor, *Proceedings Computer Science in the Netherlands 91*, pages 468–483, 1991.

[45] J. Riguet. Relations binaires, fermetures, correspondances de Galois. *Bulletin de la Société Mathématique de France*, 76:114–155, 1948.

[46] G. Schmidt and T. Ströhlein. Relation algebras: Concept of points and representability. *Discrete Mathematics*, 54:83–92, 1985.

[47] G. Schmidt and T. Ströhlein. *Relations and Graphs, Discrete Mathematics for Computer Scientists*. EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin Heidelberg, 1993.

[48] C. Strachey. Fundamental concepts in programming languages. Lecture Notes, International Summer School in Computer Programming, Copenhagen, August 1967.

[49] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73–89, 1941.

[50] Alfred Tarski and Steven Givant. *A Formalization of Set Theory without Variables*, volume 41 of *Colloquium Publications*. American Mathematical Society, Providence, Rhode Island, 1987.

[51] D. Tuijnman. *A Categorical Approach to Functional Programming*. PhD thesis, Department of Computer Science, University of Ulm, Germany, 1996.

[52] P. Wadler. Theorems for free! In *4'th Symposium on Functional Programming Languages and Computer Architecture, ACM, London*, September 1989.

Index

Samenvatting

Introductie

De eigenschap “generiek” — in de betekenis van “algemeen, niet specifiek of speciaal” — is zonder meer een noodzakelijke eigenschap voor computer software. Computers zijn immers geschikt voor meerdere doeleinden en de meest succesvolle toepassingen zijn software systemen die vanwege hun brede toepasbaarheid geschikt zijn voor een grote markt en tevens aangepast kunnen worden aan de specifieke eisen van een klant. Voorbeelden van dergelijke systemen zijn workflow management systemen, logistieke systemen en systemen voor financiële administratie.

In dit proefschrift gaan we niet in op zulke grootschalige systemen. We zullen ons beperken tot kleine programma’s en programma-onderdelen (algoritmen). Ook op dit niveau is de mogelijkheid om zogenaamde “generieke” programmacode te schrijven om veel voorkomende programma-patronen te beschrijven van groot belang voor het hergebruiken van programmacodes en bevorderen van de produktiviteit van de programmeur. Van veel programmeertalen en programmeer-methologieën wordt beweerd dat ze enige vorm van genericiteit ondersteunen. Voorbeelden zijn “overloading”, de generieke klasse van ADA, polymorfie van functionele programmeertalen en “inheritance” bij object georiënteerd programmeren.

Hergebruik is niet de enige reden waarom generieke programma’s van belang zijn. Een andere reden is de mogelijkheid om het achterliggende concept bij verschillende maar toch gelijksoortige algoritmen te kunnen unificeren. In [35] vraagt Meertens zich af wat opwindender is: weer een ander algoritme te vinden of te ontdekken dat twee bekende algoritmen voorbeelden zijn van één, abstracter algoritme. Dit laatste kan tot nieuwe inzichten leiden, aanleiding geven voor het vinden van andere verbanden, het mogelijk maken om onze kennis te organiseren en te structureren, en er uiteindelijk voor zorgen dat vraagstukken die ooit wetenschappelijke hoogstandjes waren, routinematige taken worden.

Er zijn nog twee andere redenen die we van belang achten. De eerste reden is de toename in compactheid en precisie die een hogere graad van abstractie biedt. De tweede reden is de afname van de bewijsverplichting aangezien het niet nodig is steeds min of meer hetzelfde argument te herhalen voor de vele specifieke gevallen.

Genericiteit wordt in het algemeen belangrijk gevonden. Het is echter moeilijk objectief te zijn over de graad van genericiteit. De term “generiek” kan net zoals andere modewoorden straffeloos worden toegepast op bijna alles. Het doel van dit proefschrift is om een bijdrage te leveren aan de ontwikkeling van een wiskundige theorie van generiek programmeren, waarbij verifieerbare criteria worden gegeven voor de notie van genericiteit.

Gegevensstructuren

Veelal bepaalt de structuur van de invoer voor een programma de structuur van het te schrijven programma. Een voorbeeld hiervan is een programma dat de som van een lijst van getallen berekent. Een lijst kan worden opgevat als ofwel een lege lijst, ofwel een getal gevolgd door een (mogelijk lege) lijst van getallen. Het meest voor de hand liggende programma voor het berekenen van de som van een lijst getallen werkt als volgt. Eerst wordt er gecontroleerd of de lijst leeg is; in dat geval is het antwoord nul. Indien de lijst niet leeg is, wordt het eerste getal opgeteld bij de som van de lijst van de overige getallen.

Het ligt voor de hand dat het zo ook mogelijk is een programma te schrijven dat alle getallen optelt die zijn opgeslagen in een ingewikkeldere gegevensstructuur dan bijvoorbeeld lijsten. Het programma volgt de structuur van een specifieke gegevensstructuur en telt alle getallen op die het onderweg tegenkomt.

Op deze manier kun je voor iedere specifieke gegevensstructuur een programma schrijven dat de som van alle getallen berekent. Met de generieke concepten beschreven in dit proefschrift is het mogelijk om één algemeen, abstract programma-schema te schrijven, waarin de specifieke vorm van de gegevensstructuur nog moet worden ingevuld. Door het invullen van de vorm van de gegevensstructuur wordt het bijbehorende programma verkregen.

Een programma waarin de specifieke vorm van de gegevensstructuur waarop het programma gaat werken nog moet worden ingevuld, noemen we “polytypiek”. In dit proefschrift beschrijven we verschillende polytypieke concepten, waarmee dergelijke polytypieke programma’s geschreven kunnen worden.

“Element van”-test

Eén van de belangrijkste polytypieke concepten die we beschouwen in dit proefschrift is de notatie van een “element van”-test. Dit is de test die bepaalt of een waarde is opgeslagen in een gegevensstructuur. We geven een polytypieke karakterisatie van deze test. Dat wil zeggen dat we precies die eigenschappen definiëren waaraan een “element van”-test voor een willekeurige gegevensstructuur moet voldoen. We gaan zelfs een stapje verder: we definiëren een “gegevensstructuur” als een wiskundige object waarvoor een “element van”-test bestaat. Met andere woorden, de minimale eis voor een gegevensstructuur is dat opgeslagen waarden geïnspecteerd kunnen worden. De “element van”-test blijkt fundamenteel te zijn. Het is mogelijk om andere, ingewikkelde noties voor een gegevensstructuur uit te drukken in termen van de bijbehorende “element van”- test.

Verwisselen van gegevensstructuren

Een probleem dat we in detail uitwerken in dit proefschrift is de generalisatie van matrix-transpositie. Als je een matrix van dimensies n bij m beschouwt als een lijst ter lengte n die lijsten ter lengte m bevat, dan is de transpositie van deze matrix een lijst ter lengte m die lijsten ter lengte n bevat. Het algemene probleem is om een programma te schrijven dat een gegevensstructuur van het type F die gegevensstructuren van het type G bevat, te transformeren naar een gegevensstructuur van het type G die gegevensstructuren van het type F bevat. Hiervoor geven we een polytypieke karakterisatie van zo’n transformatie. Dat wil zeggen dat we een definitie geven waarin de specifieke keuze voor de buitenste en de binnenste gegevensstructuur nog niet zijn ingevuld. Door het invullen van een keuze voor de twee soorten gegevensstructuren krijg je de specificatie van de transformatie van die twee gegevensstructuren. Door bijvoorbeeld voor de twee gegevensstructuren lijsten te nemen, wordt matrixtranspositie verkregen.

Hogere orde natuurlijkheid

Een belangrijk hulpmiddel bij het abstract definiëren van gegevensstructuur-transpositie is de notie van "hogere orde natuurlijkheid". Hogere orde natuurlijkheid drukt uit dat de verschillende transposities behorende bij gerelateerde gegevensstructuren ook gerelateerd zijn. Voor elk polytypiek programma is een hogere orde natuurlijkheid eigenschap te bepalen. Als deze hogere orde natuurlijk eigenschap geldt dan betekent dit dus dat twee verschillende programma's die verkregen zijn door het invullen van twee gerelateerde gegevensstructuren ook gerelateerd zijn. Aangezien we dit een belangrijke eigenschap vinden voor polytypieke programma's eisen we dat voor elke polytypiek programma de bijbehorende hogere orde natuurlijkheid eigenschap moet gelden. Op deze manier wordt gegarandeerd dat een polytypiek programma zich gelijksoortig gedraagt voor verschillende gegevensstructuren. Inderdaad geldt voor gegevensstructuur-transpositie en de "element van"-test de bijbehorende hogere orde natuurlijk eigenschappen.

Hogere orde natuurlijkheid voldoet aan de eisen die we ons in het begin hebben gesteld. Het is verifieerbaar en draagt bij aan een toename in compactheid en precisie met name bij bewijsvoering.

Curriculum Vitae

Paul Ferenc Hoogendijk

3 maart 1968	Geboren te Eindhoven
juli 1986	Diploma VWO Eindhovens Protestants Lyceum
juni 1991	Doctoraal examen Technische Informatica Technische Universiteit Eindoven
juli 1991 – juli 1995	Onderzoeker in opleiding in dienst van de Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), verbonden aan de Technische Universiteit Eindhoven