Categories of Containers

Thesis submitted for the degree of
Doctor of Philosophy
at the University of Leicester

by

Michael Gordon Abbott BA (Cambridge)
Department of Computer Science
University of Leicester

August 2003

Categories of Containers

by

Michael Abbott

This thesis develops a new approach to the theory of datatypes based on separating data and storage resulting in a class of datatype called a *container*. The *extension* of a container is a functor which can be regarded as a generalised polynomial functor in type variables. A representation theorem allows every natural transformation between container functors to be represented as a unique pair of morphisms in a category.

Under suitable assumptions on the ambient category container functors are closed under composition, limits, coproducts and the construction of initial algebras and final coalgebras. These closure properties allow containers to provide a functorial semantics for an important class of polymorphic types including the *strictly positive* types.

Since polymorphic functions between functorial polymorphic types correspond to natural transformations, every polymorphic function can be represented as a container morphism; this simplifies reasoning about such functions and provides a framework for developing concepts of generic programming.

Intuitionistic well-founded trees or *W-types* are the initial algebras of container functors in one parameter; the construction of the initial algebra of a container in more than one parameter leads to the solution of a problem left incomplete by earlier work of Dybjer.

We also find that containers provide a suitable framework to define the *derivative* of a functor as a kind of linear exponential. We show that the use of containers is essential for this approach.

The theory is developed in the context of a fairly general category to allow for a wide choice of applications. We use the language of dependent type theory to develop the theory of containers in an arbitrary extensive locally cartesian closed category; much of the development in this thesis can also be generalised to display map categories. We develop the appropriate internal language and its interpretation in a category with families.

Contents

Chapter 1

Introduction

This chapter introduces the thesis with an overview of the chapters, and provides some basic definitions and useful background results.

1.1 Introduction

This thesis develops a theory of data types in the form of generalised polynomials, referred to here as *containers*. Each container is represented by an internal family in a suitable ambient category $\mathbb{C}$ and has an *extension*, or associated *container functor*, assigning to each $X \in \mathbb{C}$ a type $TX \in \mathbb{C}$ constructed using local exponentials. Thus the extension of a container can be thought of as a polymorphic data type in one or more variables.

The polymorphic datatypes represented by containers have a couple of special characteristics. Firstly, the assignment of types to type variables in a container extension is functorial; this means that polymorphic types contravariant in type variables are not captured by containers. Secondly, containers can be regarded as defining a "data independent" representation of datatypes.

Polymorphic functions between functorial polymorphic datatypes can be regarded as natural transformations between the functors associated with the datatypes (Wadler, 1990; Bainbridge et al., 1990). In this thesis I show that it is possible to define morphisms between containers to precisely represent natural transformations between container functors (theorem 4.3.3); this provides a simple abstract description of polymorphic functions and helps in the proof of a number of important results about containers.

The heart of this thesis is the construction of a category $\mathscr{G}$ of containers together with a full and faithful functor $T : \mathscr{G} \to [\mathbb{C}, \mathbb{C}]$ assigning to each container its extension. An important generalisation of this construction is the extension of a container to multiple parameters: this corresponds to a polymorphic type with multiple type variables $X_1, \ldots, X_n$. This notion of a container is extended to define for each set I the category $\mathscr{G}_I$ of I-indexed containers with extension functor $T : \mathscr{G}_I \to [\mathbb{C}^I, \mathbb{C}]$.

I show that these containers have the right closure properties to model the *strictly positive* types. Containers can therefore be used to provide a categorical semantics for a important class of particularly well-behaved polymorphic types. The thesis concludes with an application of containers to the derivatives of functors.

The theory of containers is developed in a fairly general framework. The ambient category $\mathbb{C}$ is required to be be extensive and locally cartesian closed; with the addition of W-types and their dual, M-types, this is sufficient to model strictly positive types and obtain the results of this thesis. In particular, the category $\mathbb{C}$ itself supports a full interpretation of dependent type theory with Sigma, Pi, equality and coproduct types, in other words, a model of Martin-Löf type theory without universes (Martin-Löf, 1974).

The theory of containers can be interpreted in any category supporting this structure. This includes any elementary topos with a natural numbers object, including for example the effective topos (Hyland, 1982). In effect this thesis develops a theory for the interpretation of "total" functional programming, see for example Turner (1996). Further work will be required to apply this theory to "partial" programming and categories of domains; this is discussed briefly in chapter 7.

Throughout the rest of this thesis I will follow well established mathematical convention in writing "we" in reference to the author (myself) and the reader, in recognition of the effort required of any reader of mathematics.

Acknowledgements

This thesis was developed under the supervision of Neil Ghani and with advice from Thorsten Altenkirch. I would like to express my grateful thanks to Neil for encouraging me to register as a PhD student in the first place and for his continuous encouragement of my research. Summaries of some of the results in this thesis were previously published as joint work with Neil, Thorsten and Conor McBride (Abbott

et al., 2003a,b).

Chapters 2 and 3 present my interpretation of fairly standard material derived largely from Streicher (1991), Hofmann (1994, 1997b) and Jacobs (1999). The first two sections of chapter 5 provide a review of standard material including some folklore. The rest of the thesis is original material, except as cited.

The conviction that containers must be closed under the type forming constructions of strictly positive types was originally due to Thorsten, as was the conviction that least and greatest fixed points of containers could be constructed uniformly from an initial algebra of positions. I dedicate proposition 4.3.5 to Neil.

I would like to thank Vincent Schmitt for allowing me time to finish this work while working on his research grant EPSRC GR/R63004/01; I also thank the University of Leicester for funding my conference visit to present Abbott et al. (2003b). Finally I would like to thank Bart Jacobs for writing Jacobs (1999) and diligently answering my questions about the exercises for over a year. This gave me the impetus to resume academic study; I also thank Peter Johnstone for his attempts to encourage me many years ago and I regret my obtuseness at the time.

1.2 Overview of the Thesis

The present chapter introduces the material of this thesis and provides some background definitions. In particular, a handful of important categorical reasoning principles about the ambient category need to be introduced: intensional choice, Currying and the role of disjoint coproducts.

The rest of the thesis makes extensive use of the dependently typed internal language associated with the ambient category $\mathbb{C}$, and chapters 2 and 3 are concerned with developing this language and its semantics in some detail.

Chapter 2 concentrates on defining the syntax of the internal language and its "naïve" interpretation in $\mathbb{C}$. The language allows us to freely treat an object $B \in \mathbb{C}/A$ of a slice as a dependent type $A \vdash B$ or even $a : A \vdash B(a)$, and we will freely move between the various representations of objects and morphisms obtained by mixing the internal language with the more conventional language of categories.

The internal language developed here is very close to Martin-Löf type theory (Martin-Löf, 1974; Nordström et al., 1990), and supports a variety of categorical interpretations. The presentation here differs from other presentations in that the purely "substitutional" or "dependently algebraic" part of the language is carefully separated from the type theory; this helps when dissecting the interpretation of the language.

In chapter 3 we formalise the interpretation of the language and introduce fibrations as a framework. Most of this thesis is implicitly developed in the context of fibrations (for example, theorem 4.3.3 depends on the use of *fibred* natural transformations), but we will restrict ourselves to working in the *codomain* fibration throughout this thesis. In particular this means that *every* morphism in the ambient category can stand as a type in the type theory

As observed in chapter 7 all of the work in this thesis can be transferred (with a number of precautions) into the context of a more general fibration, probably a "full comprehension fibration" (Jacobs, 1999, 1991). In chapter 3 we set up some of the appropriate machinery, but we stop with the introduction of categories with families.

In chapter 4 we show that (depending on the ambient category $\mathbb{C}$) the categories $\mathscr{G}_I$ have limits and coproducts preserved by T. Similarly, functor composition is reflected by an operation of container composition making the categories of containers into a bicategory $\mathscr{G}$ equipped with a bifunctor into the 2-category of categories.

Two further notions are introduced in this chapter. An important notion is that of a *cartesian* morphism of containers, which can be regarded as a kind of "linear" function; this plays a key role in the development of derivatives. We also reference the definition of containers to two previous definitions, namely the shapely types of Jay and Cockett (1994) and the partial products of Dyckhoff and Tholen (1987).

In chapter 5 we examine the fixed points of container functors and show that containers are closed under the construction of initial algebras and final coalgebras. In the case of a container in one parameter this is precisely the construction of the W-type or its dual, the M-type; in the case of containers in multiple parameters we need to solve a tricky equation in inductive families.

At the end of chapter 5 we are in a position to see that containers provide a good model for *strictly positive* types, namely those polymorphic types built up from type variables using standard type forming operations but with restricted function types.

Chapter 6 on derivatives introduces a promising application of containers. In the construction of fixed points we begin to see the significance of "paths" into the data (in effect this is the import of proposition 5.5.1); in chapter 6 we make the paths into the data more explicit.

The notion of the "derivative" of a functor as a linear exponential is rather surprising; in particular, the fact that the equations familiar from calculus (Leibniz, 1684) are satisfied is unexpected. As example 6.4.7 shows, this does not work for arbitrary functors, but containers introduce a class of functors with well defined derivatives.

Finally in chapter 7 we review the work left open by this thesis; two areas of future development are of particular interest. First note that although we have extensively used the language of dependent types to discuss containers, the containers in this thesis cannot themselves capture dependent types. This can be achieved by allowing the index "set" I in $\mathscr{G}_I$ to be an object of $\mathbb{C}$; however, to develop this in a fully satisfactory way the abstract framework needs more work.

Secondly, the work on derivatives is capable of further development. In Joyal (1986) and Hasegawa (2002) we learn the significance of *analytic* functors, which can be captured as *quotient containers* by a suitable generalisation. Further it should be possible to reduce the dependence on "decidability" in the work on derivatives.

Introducing the Internal Language

The presentation of the internal language and its interpretation in chapters 2 and 3 addresses a long standing preoccupation of the author, and can be read somewhat separately from the rest of this thesis. The language of dependently typed algebra is particularly powerful and is used as the basis for this thesis; we develop enough of the theory of its semantics in fibrations to express the results about containers presented here.

This thesis concentrates on developing categories of containers in a category $\mathbb{C}$ with no further imposed type structure, and in particular *every* morphism in $\mathbb{C}$ is allowed to stand as a type. This substantially simplifies the development and allows us to ignore the distinction between $\mathbb{C}$ and its type theory most of the time.

More interesting systems become possible when a distinction is made between the category of *contexts* and the types in each context: this is the motivation behind the introduction of fibrations in chapter 3. However, when dealing with *dependent* type theory every type can generate a new context, so there is an intimate relationship between the category of contexts and the fibres of types. For the purposes of this thesis this is conveniently captured by the *categories with families* of Dybjer (1995), though one interesting weakness remains incompletely addressed: the category with families associated with $\mathbb{C}^I$ with contexts in $\mathbb{C}$ does not adequately describe the *morphisms* of $\mathbb{C}^I$.

1.3 Introducing Containers

A container in one parameter is given by an internal family $B \in \mathbb{C}/A$, which we'll write as $(A \triangleright B)$: the object B can be regarded as an "A-indexed family" of objects of $\mathbb{C}$. The *extension* of the container $(A \triangleright B)$ is then the functor $T_{A \triangleright B} : \mathbb{C} \to \mathbb{C}$ defined by the expression

$$T_{A \triangleright B} X \equiv \sum a : A.\ X^{B(a)} = \sum_A X^B \quad .$$

Here the first form (with a bound variable) uses the internal dependently typed language; the second form is the categorical equivalent. Functors of this form can be regarded as generalised polynomials or as data independent polymorphic datatypes.

Containers as Generalised Polynomials

Let $\mathbb{C}$ be a category with finite products and finite distributive coproducts, in other words the morphism $(A \times B) + (A \times C) \to A \times (B + C)$ is iso. A polynomial functor $\mathbb{C} \to \mathbb{C}$ can be described as inductively constructed from the identity functor $\mathrm{id}_\mathbb{C}$ (representing the type variable X), constant types K and products and coproducts $\times$ and $+$. Every such type expression can be normalised into a functor of the form

$$PX \equiv \sum n : \mathbb{N}.\ A_n \times X^n \quad ;$$

functors of this form on **Set** are referred to as *normal* functors in Hasegawa (2002) and correspond to the shapely types of Jay (1995).

If we assume that $\mathbb{C}$ has countable coproducts (and thus has a natural numbers object $\mathbb{N}$) the family $n : \mathbb{N} \vdash A_n$ of coefficients of the polynomial can be internalised as the morphism

$$\sum_\mathbb{N} A = \sum n : \mathbb{N}.\ A_n \xrightarrow{\pi} \mathbb{N}$$

with projection π taking $(n \in \mathbb{N}, a \in A_n)$ to n.

Similarly we can internalise the finite set of size n as a family $n : \mathbb{N} \vdash \mathrm{Fin}_n$, for example as the morphism $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ taking (n,m) to $n+m+1$. In particular, define the family

$$(n,a) : \sum_{\mathbb{N}} A \ \vdash \ B(n,a) \equiv \mathrm{Fin}_n$$

by pullback of Fin_n along $\pi : \sum_{\mathbb{N}} A \to \mathbb{N}$. The extension of this container is isomorphic to the original polynomial functor, showing that polynomial functors can be captured as containers.

Containers can be regarded as a generalisation of the standard theory of polynomials where sums of arbitrary size of arbitrary exponentials are allowed.

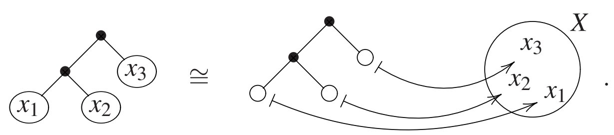
Containers as Concrete Datatypes

An element of the extension of a container $(A \triangleright B)$ at an object X corresponds to a pair $(a, f) \in \sum_A X^B$, where $a \in A$ and $f : B(a) \to X$. This has a very concrete physical interpretation in terms of a computational data structure:

- The element $a \in A$ represents a choice of "shape" of data structure. Thus the extension $T_{A \triangleright B} X$ can be regarded as a union over the shapes in A; this is the meaning of $\sum_A$.

- The function $f : B(a) \to X$ fills in all the "positions" associated with the chosen shape, ie $B(a)$, with values from the type X. When each $B(a)$ is finite this can be implemented as an array in a concrete implementation.

A couple of examples should help to clarify this. A list $[x_1, \ldots, x_n]$ consists of a number n and a function $n \to X$, thus we can write $\mathrm{List}(X) \cong \sum n : \mathbb{N}. X^n$. Similarly, a binary tree can be described by its underlying shape (obtained by deleting the data stored at the leaves) together with a function mapping the positions of those leaves to the data thus:



Note that theorem 4.3.3 tells us that every polymorphic function between container types can be captured by a function on shapes together with a function on positions for each shape in the domain. In other words, every polymorphic function can be described as a "data independent" function.

As an example consider the function $\mathsf{rev} : \mathrm{List}\,X \to \mathrm{List}\,X$ to reverse a list; this can be captured as the container morphism consisting of the identity on $\mathbb{N}$ (in other words, the length of a reversed list is unchanged) together with a function $r_n : \mathrm{Fin}_n \to \mathrm{Fin}_n$ for each $n : \mathbb{N}$ which reverses the order of elements in Fin_n; writing the elements of Fin_n as $\{0, \dots, n-1\}$, we can write $r_n m \equiv n - 1 - m$. Having made this observation it is a triviality to compute that $\mathsf{rev} \cdot \mathsf{rev} = \mathrm{id}$ by simply computing $r_n r_n m = r_n(n-1-m) = n-1-(n-1-m) = m$. Comparison of this simple proof with the proof of this identity by induction shows the power of the container formulation.

1.4 Background Definitions

For the rest of this chapter we will define the notational background for the rest of the thesis. A more formal development of the language is in chapters 2 and 3.

The following three notations will be used interchangeably as convenient for an object of the slice category $\mathbb{C}/A$:

$$B \in \mathbb{C}/A \qquad\qquad A \vdash B \qquad\qquad a : A \vdash B(a) \ .$$

The underlying object of $B \in \mathbb{C}/A$ is written as one of

$$\textstyle\sum_A B \in \mathbb{C} \qquad\qquad \vdash \sum a : A.\ B(a) \qquad\qquad A.B \in \mathbb{C}$$

with projection (or display map)

$$\pi_B : \textstyle\sum_A B \longrightarrow A \qquad \text{or} \qquad (a,b) : \sum a : A. B(a) \vdash \pi_B(a,b) = a \ .$$

The decomposition of a Sigma type into its components will be done without comment, so in general $\Gamma, A, B \vdash C$, an object of the slice category $\mathbb{C}/\Gamma.A.B \cong (\mathbb{C}/\Gamma.A)/\pi_B$, can be written in any of the following equivalent forms without further comment:

$$\begin{array}{cc} \Gamma, A, B \vdash C & \Gamma,\ x : \sum_A B \vdash C(\pi x, \pi' x) \\ \Gamma,\ a : A,\ b : B \vdash C(a,b) & \Gamma,\ (a,b) : \sum_A B \vdash C(a,b) \ . \end{array}$$

The category $\mathbb{C}$ will be assumed to be pullback complete and locally cartesian closed throughout. Each morphism $\gamma : \Delta \to \Gamma$ in $\mathbb{C}$ induces three functors between the slice categories $\mathbb{C}/\Gamma$ and $\mathbb{C}/\Delta$:

$$\begin{tikzcd} \mathbb{C}/\Gamma \arrow[r, "\gamma^*"] & \mathbb{C}/\Delta \arrow[l, bend right, "\Sigma_\gamma"'] \arrow[l, bend left, "\Pi_\gamma"] \end{tikzcd} \qquad \sum_\gamma \dashv \gamma^* \dashv \prod_\gamma \;.$$

The action of the pullback functor γ^* will be written using substitution in the internal language wherever possible, and where practical pulling back along a display map π_A will be elided.

For example, given $\vdash X$ and $A \vdash B$ the exponential $X^B \cong \prod_B \pi_B^* X$ can be written in any of the following equivalent forms:

$$A \;\vdash\; X^B \qquad A \;\vdash\; (\pi_A^* X)^B \qquad a\!:\!A \;\vdash\; X^{B(a)} \;,$$

and given $\vdash a\!:\!A$ (which is to say, a map $a : 1 \to A$) then we can write

$$\vdash\; a^*(X^B) \cong X^{a^*B} = X^{B(a)} \;.$$

Just as composition of maps f and g in $\mathbb{C}$ will be written $f \cdot g$ throughout, so also we'll write $f \cdot g : C^A$ for the composite of terms of type $f : C^B$ and $g : B^A$.

A collection of objects $(A_i)_{i \in I}$ indexed by a set can also be written in the type theory as $I \vdash A$ or $i\!:\!I \vdash A_i$; we'll mix notations as appropriate, sometimes writing $\vec{A}$ when convenient. We'll write $(\delta_{i,j})_{i,j \in I}$ for the $I \times I$-indexed collection of objects with $\delta_{i,i} = 1$ and $\delta_{i,j} = 0$ when $i \neq j$.

We will also use the notation of Sigma and Pi types for products and coproducts in $\mathbb{C}$ indexed by a set, so that $\sum_I B$ can denote the colimit $\varinjlim_{i \in I} B_i$. This conflating of notation should not cause any confusion for two reasons: firstly, it will be clear in each context whether the index object is a set or an object of $\mathbb{C}$; and secondly, formally coproducts and products *are* Sigma and Pi types respectively in the appropriate framework. This point will not be developed further in this thesis.

Interchanging Sigma and Pi

This subsection collects together a couple of results involving the interchange of Sigma and Pi types. The language used here is an application of the conventions above.

Lemma 1.4.1 *If $\mathbb{C}$ is locally cartesian closed then the following* intensional choice *isomorphism holds for types $\Gamma, A, B \vdash C$:*

$$\Gamma \;\vdash\; \prod_A \sum_B C \cong \sum_{\prod_A B} \prod_A \varepsilon^* C$$

where $\Gamma, \prod_A B, A \vdash \varepsilon : B$ *is derived from the evaluation counit* $\Gamma, A \vdash \pi_A^* \prod_A B \to B$ *so*

$$\sum_{\prod_A B} \prod_A \varepsilon^* C = \sum f : \prod_A B.\ \prod a : A.\ C(a, fa) \ .$$

Proof. Given $f : \prod_A \sum_B C$ define $\varphi f \equiv (\pi \cdot f, \pi' \cdot f)$; given $f : \prod_A B$, $g : \prod a : A. C(a, fa)$ define $\psi(f,g) \equiv \lambda a.(fa, ga)$, and it is easy to calculate $(\psi \varphi f)a = (\psi(\pi \cdot f, \pi' \cdot f))a = (\pi fa, \pi' fa) = fa$ and similarly $(\varphi \psi(f,g))a = (fa, ga)$. $\square$

In constructive type theory this principle is often referred to simply as "choice", but of course this begs the question of the role of the existential operator $\sum_B C$. Although "intensional choice" holds in any locally cartesian closed category, choice in the form "epis split" (where the logic of existence is captured by the *image* of the display map π_C above) cannot normally be expected to hold. We make no use of images or the standard logic of subobjects in this thesis, except for reasoning with equality types.

The following two principles are very familiar and are an immediate consequence of local cartesian closure.

Lemma 1.4.2 *If* $\mathbb{C}$ *is locally cartesian closed then for types* $\Gamma, A \vdash B$ *and* $\Gamma \vdash C$ *there is an isomorphism (the* Frobenius *property)*

$$\Gamma \vdash \sum_A (\pi_A^* C \times B) \cong C \times \sum_A B$$

and the following Curry *isomorphism holds*

$$\Gamma \vdash \prod_A (\pi_A^* C)^B \cong C^{\sum_A B} \ .$$

Proof. First observe that $\pi_A^*(U^C) \cong (\pi_A^* U)^{\pi_A^* C}$ (we will see this in proposition 3.4.7); we can now compute $\mathbb{C}/\Gamma(\sum_A(\pi_A^* C \times B), U) \cong \mathbb{C}/\Gamma.A(\pi_A^* C \times B, \pi_A^* U) \cong \mathbb{C}/\Gamma.A(B, \pi_A^*(U^C)) \cong \mathbb{C}/\Gamma(\sum_A B, U^C) \cong \mathbb{C}/\Gamma(C \times \sum_A B, U)$ natural in U.

Now use Frobenius to calculate $\mathbb{C}/\Gamma(U, \prod_A (\pi_A^* C)^B) \cong \mathbb{C}/\Gamma.A(\pi_A^* U, (\pi_A^* C)^B) \cong \mathbb{C}/\Gamma.A(\pi_A^* U \times B, \pi_A^* C) \cong \mathbb{C}/\Gamma(\sum_A (\pi_A^* U \times B), C) \cong \mathbb{C}/\Gamma(U \times \sum_A B, C) \cong \mathbb{C}/\Gamma(U, C^{\sum_A B})$ natural in U. $\square$

Disjoint Coproducts

Disjoint coproducts play a key role in the type theory and are essential for proving both the preservation of coproducts by the functor T (proposition 4.5.2) and key properties about derivatives of containers.

Definition 1.4.3 *Coproducts* $\sum_I A$ *for* $(A_i)_{i\in I}$ *are* disjoint *iff*

- *each coprojection* $\kappa_i : A_i \to \sum_I A$ *is a monomorphism;*
- *given indexes* $i \neq j \in I$ *the square*

$$\begin{tikzcd} 0 \arrow[r] \arrow[d] & A_i \arrow[d, tail, "\kappa_i"] \\ A_j \arrow[r, tail, "\kappa_j"] & \sum_{i\in I} A_i \end{tikzcd}$$

  is a pullback.

A category is extensive *iff it has finite disjoint coproducts which are preserved by pullbacks.*

Note that when the ambient category is locally cartesian closed coprojections are automatically mono and coproducts are automatically preserved by pullbacks. Note also that in this situation any morphism into 0 is automatically an isomorphism, and so in particular an arrow $1 \to 0$ exists only when $\mathbb{C}$ is degenerate.

A couple of important results follow immediately from the existence of disjoint coproducts.

Proposition 1.4.4 *If* $\mathbb{C}$ *is locally cartesian closed and has an initial object* 0 *and* I*-indexed coproducts then coproducts are disjoint iff the functor*

$$\vec{\kappa}^* : \mathbb{C}\Big/\textstyle\sum_{i\in I} A_i \longrightarrow \prod_{i\in I}(\mathbb{C}/A_i)$$

taking $\sum_I A \vdash B$ *to* $(A_i \vdash \kappa_i^* B)_{i\in I}$ *is an equivalence.*

Proof. ($\Longrightarrow$ **)** First we'll show that $\vec{\kappa}^* = (\kappa_i^*)_{i\in I}$ is full and faithful. Note that in the fibre $\mathbb{C}/\sum_I A$ we can write $1 \cong \sum_{i\in I}\kappa_i$, and since products distribute over coproducts any X in this fibre can be written as $X \cong X \times \sum_I \vec{\kappa} \cong \sum_{i\in I}(X \times \kappa_i) \cong \sum_{i\in I}\sum_{\kappa_i}\kappa_i^* X$ (here $\sum_I$ is the external coproduct and $\sum_{\kappa_i}$ is the left adjoint to κ_i^*) and calculate $(\prod_{i\in I}\mathbb{C}/A_i)(\vec{\kappa}^* X, \vec{\kappa}^* Y) = \prod_{i\in I}\mathbb{C}/A_i(\kappa_i^* X, \kappa_i^* Y) \cong \prod_{i\in I}\mathbb{C}/\sum_I A(\sum_{\kappa_i}\kappa_i^* X, Y) \cong \mathbb{C}/\sum_I A(\sum_{i\in I}\sum_{\kappa_i}\kappa_i^* X, Y) \cong \mathbb{C}/\sum_I A(X, Y)$.

To show that $\vec{\kappa}^*$ is essentially surjective let $(A_i \vdash B_i)_{i\in I}$ be given and construct $\sum_I A \vdash \coprod \vec{B} \equiv \sum_{i\in I}\sum_{\kappa_i} B_i$; in other words, if each B_i is a map $\pi_{B_i} : \sum_A B_i \to A_i$ then $\coprod \vec{B}$ is the map $\pi_{\coprod \vec{B}} \equiv \sum_I \vec{\pi_B} : \sum_I \overrightarrow{\sum_A B} \to \sum_I A$. We now need to show that $\kappa_i^* \coprod \vec{B} \cong B_i$.

First it will be helpful to calculate $\kappa_j^* \sum_{\kappa_i} X$ for any $A_i \vdash X$: since each κ_i is a monomorphism it is easy to verify that $\kappa_i^* \sum_{\kappa_i} X \cong X$. On the other hand, if $i \neq j$ then by disjointness of coproducts $\kappa_j^* \sum_{\kappa_i} X \cong \sum_{!_{A_j}} !_{A_i}^* X \cong \sum_{!_{A_j}} 0 \cong 0$ where $!_{A_i} : 0 \to A_i$. Now we can compute $\kappa_j^* \coprod^{\circ} \vec{B} = \kappa_j^* \sum_{i\in I} \sum_{\kappa_i} B_i \cong \sum_{i\in I} \kappa_j^* \sum_{\kappa_i} B_i \cong B_j$.

($\Longleftarrow$) Conversely, let $\vec{\kappa}^*$ be an equivalence with adjoint $\coprod^{\circ}$. Fix $j \in I$ and let $\vec{\delta}_j \equiv (\delta_{i,j} \in \mathbb{C}/A_i)_{i\in I}$ be defined as $\delta_{i,j} = 0$ for $i \neq j$ and $\delta_{j,j} = 1 = \mathrm{id}_{A_j}$. We can see that $\coprod^{\circ} \vec{\delta}_j \cong \kappa_j$ by the following chain of reasoning natural in $u \in \mathbb{C}/\sum_I A$:

$$
\begin{array}{cl}
\coprod^{\circ} \vec{\delta}_j \longrightarrow u & \text{in } \mathbb{C}\Big/\sum_I A \\
\hline\hline
\delta_{i,j} \longrightarrow \kappa_i^* u & \text{in } \prod_{i\in I}(\mathbb{C}/A_i) \\
\hline\hline
1 \longrightarrow \kappa_j^* u & \text{in } \mathbb{C}/A_j\text{, since } \delta_{i,j} \cong 0 \text{ otherwise} \\
\hline\hline
\kappa_j \cong \sum_{\kappa_j} 1 \longrightarrow u & \text{in } \mathbb{C}\Big/\sum_I A \quad ,
\end{array}
$$

and then disjointness of products follows as $\kappa_i^* \kappa_j \cong \kappa_i^* \coprod^{\circ} \vec{\delta}_j \cong 0$. $\square$

We will write

$$\coprod^{\circ}_I : \prod_{i\in I}(\mathbb{C}/A_i) \longrightarrow \mathbb{C}\Big/\sum_{i\in I} A_i$$

for the adjoint to $\vec{\kappa}^*$ constructed above; in the case of binary coproducts we'll write

$$A + C \;\vdash\; B \mathbin{\overset{\circ}{+}} D$$

for the "parallel" coproduct of $A \vdash B$ and $C \vdash D$.

The following two useful equations can be extracted from the proof of proposition 1.4.4. We will use these in the proof of proposition 4.5.2.

Corollary 1.4.5 *If $\mathbb{C}$ is locally cartesian closed and has disjoint coproducts then for $(A_i \vdash B_i)_{i\in A}$ and $\sum_I A \vdash C$ the following isomorphisms hold:*

$$\kappa_i^* \Big(\coprod^{\circ}_{i\in I} B_i\Big) \cong B_i \qquad\qquad \sum_{i\in I} \sum_{A_i} \kappa_i^* C \cong \sum_{\sum_I A} C \quad . \qquad \square$$

The following lemma involving disjoint coproducts will be used in the proof of theorem 6.4.3.

Lemma 1.4.6 *If $\mathbb{C}$ has disjoint coproducts then any morphism $f : 1 + A \to 1 + B$ in $\mathbb{C}$ satisfying $f \cdot \kappa = \kappa$ can be written as $f = 1 + g$ for a* unique *$g : A \to B$.*

Proof. First observe that f can be factorised as $(1+\pi')\cdot[\kappa;(\mathrm{id}_A, f\cdot\kappa')] = [\kappa; f\cdot\kappa'] = [f\cdot\kappa; f\cdot\kappa'] = f$ thus

$$\begin{tikzcd}
 & 1+A \\
1+A \arrow[ur, bend left, "\mathrm{id}_{1+A}"] \arrow[r, "{[\kappa;(\mathrm{id}_A, f\cdot\kappa')]}"] \arrow[dr, bend right, "f"'] & 1+A\times B \arrow[u, "1+\pi"'] \arrow[d, "1+\pi'"] \\
 & 1+B
\end{tikzcd} .$$

In other words, f can be written as a map $1 \to 1 \stackrel{\circ}{+} A^*B$ in $\mathbb{C}/1+A$, and by proposition 1.4.4 this map corresponds to a unique $1 \to A^*B$ in $\mathbb{C}/A$; this gives us the required $g : A \to B$. $\square$

We will assume henceforth that all coproducts in the ambient category $\mathbb{C}$ are disjoint.

Full and Faithful Functors

We conclude with a couple of observations about full and faithful functors. First note that if $F:\mathbb{A}\to\mathbb{B}$ and $G:\mathbb{C}\to\mathbb{D}$ are full and faithful functors then so is the functor $F\times G:\mathbb{A}\times\mathbb{C}\to\mathbb{B}\times\mathbb{D}$.

Given a full and faithful functor $G:\mathbb{C}\to\mathbb{B}$ the result below allows us to construct a functor into $\mathbb{C}$ from a functor into $\mathbb{B}$ by specifying only the objects.

Lemma 1.4.7 *If $G:\mathbb{C}\to\mathbb{B}$ is a full and faithful functor then any map of objects $H_0 : \mathbb{A}\to\mathbb{C}$ which agrees with some functor $F:\mathbb{A}\to\mathbb{B}$, making $GH_0X \cong FX$ for each object $X\in\mathbb{A}$, extends to a functor $H:\mathbb{A}\to\mathbb{C}$ with $GH\cong F$.*

Proof. Write $\theta_X : GH_0X \cong FX$ and $\varphi_{X,Y} : \mathbb{B}(GX,GY)\cong\mathbb{C}(X,Y)$ and for $f:X\to Y$ in $\mathbb{A}$ define $Hf \equiv \varphi(\theta_Y\cdot Ff\cdot\theta_X^{-1})$. By construction $\theta : F\to GH$ is a natural isomorphism, and so in particular by reflection the assignment H is functorial. $\square$

Similarly limits and colimits can be reflected along a full and faithful functor.

Lemma 1.4.8 *Given a full and faithful functor $G:\mathbb{C}\to\mathbb{B}$ and a diagram $D:\mathbb{J}\to\mathbb{C}$ then if the composite diagram GD has a limit or colimit in $\mathbb{B}$ isomorphic to some GX then X is the limit or colimit respectively of D in $\mathbb{C}$.*

Proof. Let $GX \cong \varinjlim GD$ then $\mathbb{C}^{\mathbb{J}}(D,\Delta U)\cong\mathbb{B}^{\mathbb{J}}(GD,\Delta GU)\cong\mathbb{B}(GX,GU)\cong\mathbb{C}(X,U)$ natural in U, and dually for the limit. $\square$

Chapter 2

The Internal Language

In this chapter we introduce the formalism of a dependently typed internal language and show that it can be a useful tool for deducing properties of categories with enough structure to interpret the language.

2.1 Introducing the Internal Language

The theory of containers is a theory of a sub-class of fibred functors $\mathbb{C}^I \to \mathbb{C}^J$ between set-indexed powers of a locally cartesian closed category $\mathbb{C}$. To support the exposition of this theory we need to develop enough of the theory of fibred functors and the associated internal language. In particular, categorical reasoning in this framework can be substantially simplified once the internal language has been successfully set up.

The internal language gives us a tool for reasoning about functors and morphisms in a category $\mathbb{C}$. The internal language provides a number of important technical advantages of which three are of particular interest.

- Reasoning with variables. The internal language allows us to use a term language to reason about morphisms in $\mathbb{C}$; in certain cases this can substantially simplify the arguments.
- Implicit weakening. Reasoning with variables allows applications of weakening functors $\pi_B^* : \mathbb{C}/\Gamma \to \mathbb{C}/\Gamma.B$ to be implicit, which again simplifies arguments.
- Strictness of functors, natural transformations and type constructors. By reasoning with the internal language and its associated interpretation we are able

to treat pulling back along a substitution as a strictly commutative operation and can ignore coherence isomorphisms.

In this chapter we introduce the syntax of the internal language, and in the next we will described its detailed interpretation via "categories with families". This approach will allow us to reason at a variety of different levels as appropriate and move fairly freely between the different interpretations.

$$\begin{tikzcd}[column sep=large]
\mathbb{C},\mathbb{C}^I \arrow[r, leftrightarrow, "\substack{\text{abstraction}\\ \&\\ \text{localisation}}"] & \text{fibrations} \arrow[r, leftrightarrow, "\substack{\text{splitting,}\\ \text{types \&}\\ \text{terms}}"] & \substack{\text{categories with}\\ \text{families}} \arrow[r, leftrightarrow, "\substack{\text{variables,}\\ \text{implicit}\\ \text{weakening}}"] & \substack{\text{internal}\\ \text{language}}
\end{tikzcd}$$

Figure 2.1: Context of the Internal Language

Figure 2.1 summarises the structure of the presentation. After an introduction to a simply typed internal language and an overview of its limitations we will work along the above diagram from right to left in this chapter and the next.

A Simply Typed Internal Language

A good point of introduction is the internal language for a category with products. This will allow us to present the basics of the full internal language and to indicate some of the reasons why the simple language is insufficient for this thesis.

The details are fairly straightforward and illuminating, and it will then be possible to appeal to this explanation when we come to examine the internal language for a category with families. The following is pretty standard.

Definition 2.1.1 *A* simply typed signature Σ *is given by a set* $|\Sigma|$ *of* types *together with a map* $\mathscr{F}_\Sigma$ *assigning to each sequence of types* $\langle\sigma_1,\ldots,\sigma_n\rangle \in |\Sigma|^\star$ *and* $\tau \in |\Sigma|$ *a set* $\mathscr{F}_\Sigma(\langle\sigma_1,\ldots,\sigma_n\rangle,\tau)$ *of* function symbols*. Each* $f \in \mathscr{F}_\Sigma(\langle\sigma_1,\ldots,\sigma_n\rangle,\tau)$ *is written as* $f : \sigma_1,\ldots,\sigma_n \to \tau$*.*

From any category $\mathbb{C}$ with products we can derive the standard signature $\Sigma_\mathbb{C}$ for that category by defining $|\Sigma_\mathbb{C}| \equiv \mathrm{ob}\,\mathbb{C}$ and $\mathscr{F}_\Sigma(\langle\sigma_1,\ldots,\sigma_n\rangle,\tau) \equiv \mathbb{C}(\sigma_1\times\cdots\times\sigma_n,\ \tau)$ for a canonical choice of iterated product.

Conversely, an *interpretation* of a signature Σ in a category $\mathbb{C}$ is given by assignments $[\![-]\!] : \Sigma \to \mathrm{ob}\,\mathbb{C}$ and $[\![-]\!] : \mathscr{F}_\Sigma(\langle\sigma_1,\ldots,\sigma_n\rangle,\tau) \to \mathbb{C}([\![\langle\sigma_1,\ldots,\sigma_n\rangle]\!],[\![\tau]\!])$

where $[\![\langle\sigma_1,\ldots,\sigma_n\rangle]\!] \equiv [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!]$ for a canonical choice of iterated product. It is clear that the standard signature $\Sigma_{\mathbb{C}}$ is equipped with an interpretation $[\![-]\!] : \Sigma_{\mathbb{C}} \to \mathbb{C}$ into its original category.

Given a simply typed signature we can derive a language of "simply typed terms in context". Every term is defined in some context $\Gamma \vdash$ where Γ is a finite sequence of $x : \sigma$ variable-type bindings and is assigned a particular type. The statement that t is a well formed term of type τ in context Γ is written $\Gamma \vdash t : \tau$; this is referred to as a "type judgement".

Terms are constructed from the signature using two rules. Firstly, each variable in a context can be written as a term:

$$x_1 : \sigma_1,\ \ldots,\ x_n : \sigma_n \ \vdash\ x_i : \sigma_i \qquad \text{for } 1 \leq i \leq n.$$

Secondly, for each function symbol $f : \sigma_1,\ldots,\sigma_n \to \tau$ terms in any context Γ can be constructed inductively:

$$\frac{\Gamma \vdash t_1 : \sigma_1 \quad \ldots \quad \Gamma \vdash t_n : \sigma_n}{\Gamma \vdash f(t_1,\ldots,t_n) : \tau} \ .$$

The set of terms in context generated by these rules is the "language generated by Σ" which we can conveniently refer to as $\mathscr{L}_\Sigma$, or as $\mathscr{L}_{\mathbb{C}}$ when $\Sigma = \Sigma_{\mathbb{C}}$.

An interpretation $[\![-]\!] : \Sigma \to \mathbb{C}$ extends in a straightforward way to an interpretation on the entire language. Define the interpretation of a term $\Gamma \vdash t : \tau$ to be a morphism $[\![t]\!] : [\![\Gamma]\!] \to [\![\tau]\!]$ as follows:

$$\begin{aligned}
[\![\Gamma]\!] = [\![x_1 : \sigma_1,\ldots,x_n : \sigma_n]\!] &\equiv [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!] \\
[\![x_i]\!] &\equiv \pi_i : [\![\Gamma]\!] \longrightarrow [\![\sigma_i]\!] \\
[\![f(t_1,\ldots,t_n)]\!] &\equiv [\![f]\!] \cdot ([\![t_1]\!],\ldots,[\![t_n]\!]) \ .
\end{aligned}$$

Associated with this syntactic construction we have a notion of substitution. Here I will define simultaneous substitution over the entire context, but it is quite usual to define substitution of a single variable.

Definition 2.1.2 *Given contexts Δ and $\Gamma = (x_1 : \sigma_1,\ldots,x_n : \sigma_n)$ a* context morphism *$\gamma : \Delta \to \Gamma$ is given by a tuple of terms $(\Gamma \vdash \gamma_1 : \sigma_1,\ \ldots,\ \Gamma \vdash \gamma_n : \sigma_n)$. Given a term $\Gamma \vdash t : \tau$ the* substitution of γ in t *is a term $\Delta \vdash t[\gamma] : \tau$ defined inductively as follows:*

$$\begin{aligned}
x_i[\gamma] &\equiv \gamma_i \\
f(t_1,\ldots,t_m)[\gamma] &\equiv f(t_1[\gamma],\ldots,t_m[\gamma]) \ .
\end{aligned}$$

Define the interpretation of γ to be $[\![\gamma]\!] \equiv ([\![\gamma_1]\!], \ldots, [\![\gamma_n]\!]) : [\![\Delta]\!] \to [\![\Gamma]\!]$ and then we can observe that $[\![t[\gamma]]\!] = [\![t]\!] \cdot [\![\gamma]\!]$. Composition of context morphisms $\Xi \xrightarrow{\delta} \Delta \xrightarrow{\gamma} \Gamma$ can be defined using substitution by defining $(\gamma \cdot \delta)_i \equiv \gamma_i[\delta]$ for $1 \leq i \leq n$.

Note that substitution in a single variable $t[u/y]$ can be constructed by building a suitable context morphism, in particular for $\Gamma, y : \sigma \vdash t : \tau$ and $\Gamma \vdash u : \sigma$ single variable substitution is obtained by defining $u/y : \Gamma \to \Gamma, \sigma$ by $(u/y)_i \equiv x_i$ for $i \leq n$ and $(u/y)_{n+1} \equiv u$.

The point of introducing this language is the calculus that comes with it. In particular we can reason using equality and substitution in a very familiar way, and when working in the language associated with a category all the conclusions reached while working in the language can be reflected directly back into corresponding conclusions in the original category.

To make this precise we'll introduce the notion of an equational theory.

Definition 2.1.3 *An* equation in context *in the language $\mathscr{L}_\Sigma$ is a pair of terms of the same type in the same context, written $\Gamma \vdash t_1 = t_2 : \tau$.*

An equational theory presentation $\mathscr{T} = (\Sigma, \mathscr{E})$ *is given by a signature Σ together with a set $\mathscr{E}$ of equations in $\mathscr{L}_\Sigma$.*

The derived *equational theory* is the set of all derivable equations under the following rules of deduction (where each $t = u$ abbreviates an equation in context $\Gamma \vdash t = u : \tau$)

$$\text{reflexivity: } \frac{}{t = t} \qquad \text{transitivity: } \frac{t = u \quad u = v}{t = v} \qquad \text{symmetry: } \frac{t = u}{u = t}$$

together with the two rules involving substitution

$$\frac{\Gamma \vdash t : \tau \quad \gamma = \gamma' : \Delta \longrightarrow \Gamma}{\Delta \vdash t[\gamma] = t[\gamma']} \qquad \frac{\Gamma \vdash t = u : \tau \quad \gamma : \Delta \longrightarrow \Gamma}{\Delta \vdash t[\gamma] = u[\gamma]} .$$

That these rules are *correct* is captured by the observation that if $\mathbb{C} \models \mathscr{E}$ and $\mathscr{E} \vdash u = v$ then $\mathbb{C} \models u = v$. That these rules are *complete* is expressed by saying that $\mathscr{E} \models u = v$ implies that $\mathscr{E} \vdash u = v$.

Say that an interpretation of Σ in a category $\mathbb{C}$ *respects an equation* $\Gamma \vdash t = u : \tau$ iff $[\![t]\!] = [\![u]\!]$ in $\mathbb{C}$. We can observe that an interpretation will respect every equation in a presentation iff it respects every equation in the derived theory.

Finally observe that there is a canonical equational theory $\mathscr{T}_\mathbb{C}$ for the language $\mathscr{L}_\mathbb{C}$ obtained by defining $t = u$ iff $[\![t]\!] = [\![u]\!]$ and we have the following central result.

Theorem 2.1.4 *Two parallel morphisms $f,g:X \rightrightarrows Y$ in a category $\mathbb{C}$ are equal iff the equation $x:X \vdash f(x) = g(x):Y$ holds in $\mathscr{T}_{\mathbb{C}}$.*

The moral of this theorem is that we can perform equational reasoning in the calculus of $\mathscr{T}_{\mathbb{C}}$ to conclude truths about equalities in $\mathbb{C}$.

Adding Types to the Simple Language

The language presented above captures the basics of equational reasoning in a category, but we will be interested in adding structure to the types. In this section I describe how finite products and exponentials are handled in this language. The handling of equalisers and coproducts is postponed to the more general language.

Each type is described here by describing three types of rules: type formers, term formers and equations. Type formers can be regarded as adding an algebra of types to the set $|\Sigma|$ and similarly term formers provide rules for constructing terms into and out of the constructed types; indeed, term forming rules normally come in pairs: an introduction and an elimination rule.

The status of products in the language is a little odd, as they are implicitly present (in particular, note that the language can only be interpreted in a category with finite products). However, the language so far does not allow us to capture products as types.

The terminal object 1 is captured in a theory $\mathscr{T} = (\Sigma, \mathscr{E})$ by adding a constant $1 \in |\Sigma|$ together with a (constant) function symbol $*: \ \to 1$ and equations $\Gamma \vdash t = u : 1$ for all terms $\Gamma \vdash t, u : 1$.

Binary products are captured by requiring that for each pair of types $\sigma, \tau \in |\Sigma|$ there is a type $\sigma \times \tau \in |\Sigma|$ together with terms $\pi : \sigma \times \tau \to \sigma$, $\pi' : \sigma \times \tau \to \tau$ and $(,) : \sigma, \tau \to \sigma \times \tau$ (we'll write $(s,t) \equiv (,)(s,t)$ for short) satisfying equations $\pi(s,t) = s$, $\pi'(s,t) = t$ and $(\pi u, \pi' u) = u$.

It is clear that any interpretation of a $\mathscr{T}$ in a category $\mathbb{C}$ with the above structure must have $[\![1]\!] \cong 1$ and $[\![\sigma \times \tau]\!] \cong [\![\sigma]\!] \times [\![\tau]\!]$.

Function types are also easy to add to the language (an equational presentation of cartesian closed categories appears in Lambek and Scott, 1986). In the standard development of type theory (Barendregt, 1992) the existence of function types is assumed from the beginning, but I prefer to separate out the exponential part of the theory from the purely algebraic part.

For a theory to have function types we have a type $\sigma \Rightarrow \tau \in |\Sigma|$ (also written τ^σ for conciseness where convenient) for each pair of types together with two term forming rules

$$\frac{\Gamma, x\!:\!\sigma \vdash t\!:\!\tau}{\Gamma \vdash \lambda x\!:\!\sigma.\, t\!:\!\sigma \Rightarrow \tau} \qquad \frac{\Gamma \vdash t\!:\!\sigma \Rightarrow \tau \quad \Gamma \vdash u\!:\!\sigma}{\Gamma \vdash tu\!:\!\tau}$$

satisfying the equations $(\lambda x : \sigma.t)u = t[u/x]$ and $\lambda x : \sigma.\, fx = f$.

Note that in the term tu constructed here there is some ambiguity as to whether t is a term of type $\tau^\sigma = \sigma \Rightarrow \tau$ or a function symbol $\sigma \to \tau$. In practice this confusion is normally of little significance, as the two roles are interchangeable.

2.2 The Dependently Typed Internal Language

In Seely (1984) a theory and language of dependent types based on Martin-Löf's type theory (Martin-Löf, 1974; Nordström et al., 1990) is described together with its interpretation in a locally cartesian closed category. This interpretation (and its variants, Streicher, 1991; Hofmann, 1997a; Crole, 1993; Jacobs, 1996, 1999) is the basis for the internal language described here.

As pointed out in Hofmann (1994) the naïve interpretation of the dependently typed language in a locally cartesian closed category does not work without some adjustment. This adjustment, being the interposition of a "category with attributes" (Hofmann, 1994) or equivalently a "category with families" (Dybjer, 1995), increases the technical complexity of the interpretation but has substantial technical advantages. In particular, coherence isomorphisms that plague the detailed development of the categorical theory vanish once categories with families are introduced.

In this chapter we concentrate on a purely syntactic presentation of the internal language and an overview of the naïve interpretation. This presentation is rather schematic: a fuller description can be found in Hofmann (1997a), Jacobs (1999) or Nordström et al. (1990).

Defining the Language

The definition of the dependently typed internal language requires a rather complex mutual induction over the language being defined. This induction leaves places for types and function symbols to be inserted into the language.

As before, a dependently typed signature Σ consists of a set $T_\Sigma \equiv |\Sigma|$ of types and a set $\mathscr{F}_\Sigma$ of function symbols. The difference is that the sets of types and function symbols depend on the *terms* in the language, hence the mutual induction mentioned. We capture this dependency by including contexts in the description of the signature.

So we have to define three concepts simultaneously: well formed contexts, types in context and typed terms in context. Each of these is captured by "judgements" written as follows:

$\Gamma \vdash$	Γ is a well formed context
$\Gamma \vdash A$	A is a well formed type in context Γ
$\Gamma \vdash t : A$	t is a well formed term of type A in context Γ

At the same time these judgements are defined using the notions of a morphism of contexts $\gamma : \Delta \to \Gamma$ and substitution of types and terms along a context morphism. Note that the rules of the language will ensure that $\Gamma \vdash t : A$ implies $\Gamma \vdash A$ implies $\Gamma \vdash$.

A valid signature $\Sigma = (T_\Sigma, \mathscr{F}_\Sigma)$ defines for each valid context Γ a set of types $T_\Sigma(\Gamma)$ and for each well formed type in context $\Gamma \vdash A$ a set of function symbols $\mathscr{F}_\Sigma(\Gamma, A)$. Of course the valid contexts and types depend on the rules for constructing the language as described below, and also depend on the signature through the induction.

Here we will define contexts to be built from the empty context by adding in valid types in context (here $\diamond$ represents the empty context which is always valid).

$$\frac{}{\diamond \vdash} \qquad\qquad \frac{\Gamma \vdash \quad \Gamma \vdash A}{\Gamma,\, a : A \vdash} \ .$$

The variable a here is "new", which is to say that it is not already used in Γ. I will not discuss the various techniques used to avoid variable capture or to implement α-equivalence. It may be best to think of variables in the context as merely placeholders for deBruin indexes (de Bruijn, 1972); however in definition 3.3.4 we will see an alternative representation of variables. In this spirit the extended context may be written as Γ, A if the variable does not need a name in context.

The above rules define the construction of contexts from well formed types. The rest of the language needs a simultaneous definition of types, terms, context morphisms and substitution.

A context morphism is written as $\gamma : \Delta \to \Gamma$; this form implies $\Delta \vdash$ and $\Gamma \vdash$, as can be seen from the rules below for constructing judgements of this form. When Γ is the context $x_1:A_1,\dots,x_n:A_n$ a morphism $\gamma=(\gamma_1,\dots,\gamma_n)$ is a set of n terms of the form

$$\Delta\vdash\gamma_1:A_1\quad\Delta\vdash\gamma_2:A_2[\gamma_1/x_1]\quad\dots\quad\Delta\vdash\gamma_n:A_n[\gamma_1/x_1,\dots,\gamma_{n-1}/x_{n-1}]\ .$$

This can be defined inductively by the following two rules ($\diamond_\Gamma$ names the unique empty context morphism into the empty context):

$$\frac{\Gamma\vdash}{\diamond_\Gamma:\Gamma\longrightarrow\diamond}\qquad\qquad\frac{\gamma:\Delta\longrightarrow\Gamma\quad\Gamma\vdash A\quad\Delta\vdash t:A[\gamma]}{(\gamma,t):\Delta\longrightarrow\Gamma,A}\ .$$

Types are introduced into the language as substitution instances of types drawn from the signature through the following single rule:

$$\frac{\gamma:\Delta\longrightarrow\Gamma\quad A\in T_\Sigma(\Gamma)}{\Delta\vdash A(\gamma)}\ .$$

Terms come in two forms, just as in the simply typed language. Firstly, every variable in a context can be a term:

$$x_1:A_1,\dots,x_n:A_n\vdash x_i:A_i\quad\text{for }1\le i\le n\ ;$$

secondly, terms are constructed by filling in the place-holders in a function symbol:

$$\frac{\gamma:\Delta\longrightarrow\Gamma\quad\Gamma\vdash A\quad f\in\mathscr{F}_\Sigma(\Gamma,A)}{\Delta\vdash f(\gamma):A[\gamma]}\ .$$

Finally we have to simultaneously define substitution and context morphism composition with the rules (for context morphisms $\Xi\xrightarrow{\delta}\Delta\xrightarrow{\gamma}\Gamma$, $A\in T_\Sigma(\Gamma)$, $f\in\mathscr{F}_\Sigma(\Gamma,A)$ and $\Gamma\vdash t:B[\gamma]$):

$$\begin{array}{lll} A(\gamma)[\delta]\equiv A(\gamma\cdot\delta) & f(\gamma)[\delta]\equiv f(\gamma\cdot\delta) & x_i[\gamma]\equiv\gamma_i \\ \diamond_\Delta\cdot\delta\equiv\diamond_\Xi & (\gamma,t)\cdot\delta\equiv(\gamma\cdot\delta,\ t[\delta])\ . & \end{array}\tag{2.1}$$

We can now see that the syntactic structure of the language is very simple: every type is of the form $A(\gamma)$ and every term is either a variable or else of the form $f(\gamma)$, where each γ is itself just a sequence of terms. The substitution $t[\gamma]$ and $\gamma\cdot\delta$ syntax is, in effect, part of the meta-language, as the rules above allow substitution to be eliminated from all terms.

Several important differences between the language described here and its practical application are worth describing. In practice the empty substitution $\diamond_\Gamma$ is never written, and constant types and function symbols are introduced into the language without ceremony.

Types in the signature will be introduced by statements of the form $\vdash A$ where we mean $A \in T_\Sigma(\diamond)$ and $a\!:\!A \vdash B(a)$ for $B \in T_\Sigma(A(\diamond_\diamond))$ and so on. This last form may often be abbreviated to $A \vdash B$ where convenient.

Often it is not necessary to write the entire context morphism. For example, given constant types $\vdash A$, $A \vdash B$ and $A,B \vdash C$ we should properly write $a\!:\!A, b\!:\!B(a) \vdash C(a,b)$ and $C(t_1,t_2)$ for any substitution instance with $t_1 : A$ and $t_2 : A(t_1)$. In practice, however, the value for b will generally fully determine a which can therefore often be elided as $C(t_2)$.

Note also that the form of arguments described here is rather simplified over practice. In particular, instead of writing $f(a,b)$ it may be convenient use subscripts for some arguments, so we may write this as $f_a(b)$. Finally, it is often convenient to elide the brackets thus: $f_a b$.

Equational Reasoning

Equational reasoning in the language $\mathscr{L}_\Sigma$ is made more complex by the dependency of types on terms. In particular, equality of terms may result in equality of types, which of course then allows further terms to be constructed which did not previously exist. This means that the rules for equational reasoning described here will feed back into the definition of the language above.

For example, the language with two types $\vdash A$ and $\vdash B$ and one function symbol $A \vdash f : B$ has only one non-trivial term, namely f. If we add the type equation $\vdash A = B$ to the language then the set of terms now includes arbitrary iterations f^n of f. Thus adding equalities can expand the language, so a completely separate treatment of the language and its equality is impractical.

Corresponding to the three well formed typing judgements in the core language we have three equality judgements corresponding to equality of contexts, types and terms respectively:

$$\Gamma = \Delta \vdash \qquad\qquad \Gamma \vdash A = B \qquad\qquad \Gamma \vdash t = u : A \ ,$$

together with an equality on context morphisms $\gamma = \gamma' : \Delta \to \Gamma$. The equalities on types and terms are the most fundamental, and equality on contexts and context morphisms can be seen to be derived from these.

Observe first that equality is an equivalence relation on types

$$\frac{\Gamma \vdash A}{\Gamma \vdash A = A} \qquad \frac{\Gamma \vdash A = B}{\Gamma \vdash B = A} \qquad \frac{\Gamma \vdash A = B \quad \Gamma \vdash B = C}{\Gamma \vdash A = C}$$

and similarly for terms

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash t = t : A} \qquad \frac{\Gamma \vdash t = u : A}{\Gamma \vdash u = t : A} \qquad \frac{\Gamma \vdash t = u : A \quad \Gamma \vdash u = v : A}{\Gamma \vdash t = v : A} .$$

Equality of types allows us to transfer terms between equal types

$$\frac{\Gamma \vdash A = B \quad \Gamma \vdash t : A}{\Gamma \vdash t : B} ;$$

applying this rule allows new terms to be formed.

Four further rules complete the core system of the language. Firstly equality is preserved by substitution

$$\frac{\gamma : \Delta \longrightarrow \Gamma \quad \Gamma \vdash A = B}{\Delta \vdash A[\gamma] = B[\gamma]} \qquad \frac{\gamma : \Delta \longrightarrow \Gamma \quad \Gamma \vdash t = u : A}{\Delta \vdash t[\gamma] = u[\gamma] : A[\gamma]}$$

and finally substitution along equal context morphisms preserves equality

$$\frac{\gamma = \gamma' : \Delta \longrightarrow \Gamma \quad \Gamma \vdash A}{\Delta \vdash A[\gamma] = A[\gamma']} \qquad \frac{\gamma = \gamma' : \Delta \longrightarrow \Gamma \quad \Gamma \vdash t = u : A}{\Delta \vdash t[\gamma] = u[\gamma] : A[\gamma]} .$$

Since both contexts and context morphisms are constructed iteratively from types and terms we can simply define context and context morphism equality by putting $\diamond = \diamond \vdash$, $\diamond_\diamond = \diamond_\diamond : \diamond \rightarrow \diamond$ and the rules

$$\frac{\Gamma = \Delta \vdash \quad \Gamma \vdash A = B}{(\Gamma, A) = (\Delta, B)} \qquad \frac{\gamma = \gamma' : \Delta \longrightarrow \Gamma \quad \Delta \vdash t = u : A[\gamma]}{(\gamma, t) = (\gamma', u) : \Delta \longrightarrow \Gamma, A} .$$

Define a *type equation* in context Γ to be a pair of types (A, B) with $\Gamma \vdash A$ and $\Gamma \vdash B$. Similarly define a *term equation* in context $\Gamma \vdash A$ to be a pair of terms (u, v) with $\Gamma \vdash u : A$ and $\Gamma \vdash v : A$. Given sets $\mathscr{E}_T(\Gamma)$ of type equations and $\mathscr{E}_{\mathscr{F}}(\Gamma, A)$ of term equations we introduce these equations into the the language via rules

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B \quad (A, B) \in \mathscr{E}_T(\Gamma)}{\Gamma \vdash A = B} \qquad \frac{\Gamma \vdash t : A \quad \Gamma \vdash u : A \quad (t, u) \in \mathscr{E}_{\mathscr{F}}(\Gamma, A)}{\Gamma \vdash t = u : A} .$$

We are now in a position to define a complete dependently typed language with equality as a mutual induction of type, term and equality formers.

Definition 2.2.1 *A* dependently typed theory *is a system $\mathscr{L} = (T, \mathscr{F}, \mathscr{E}_T, \mathscr{E}_{\mathscr{F}})$ of mutually dependent types, function symbols and equations as follows (for each context Γ and each type $\Gamma \vdash A$):*

- $T(\Gamma)$ *is a set of basic types in context* Γ*;*
- $\mathscr{F}(\Gamma, A)$ *is a set of function symbols of type A;*
- $\mathscr{E}_T(\Gamma)$ *is a set of type equations in context* Γ*;*
- $\mathscr{E}_{\mathscr{F}}(\Gamma, A)$ *is a set of term equations of type A.*

2.3 Interpreting the Language

A definitive interpretation of the language $\mathscr{L}$ will have to wait for the next chapter, when we will have established the machinery of "categories with families" (Hofmann, 1994) which will be the basis for our interpretation. However, an outline of the interpretation due to Seely (1984) will be helpful.

In this thesis we will be using the language $\mathscr{L}$ as the internal language of a given "ambient" category $\mathbb{C}$. This derivation of $\mathscr{L} = \mathscr{L}_{\mathbb{C}}$ will allow us to make a number of further assumptions about $\mathscr{L}$, but first we will discuss the interpretation of a general $\mathscr{L}$ in any suitable $\mathbb{C}$.

The Naïve Interpretation

The basic idea is to interpret each context Γ as an object $[\![\Gamma]\!] \in \mathbb{C}$ and each type $\Gamma \vdash A$ as a morphism $[\![A]\!] : [\![\Gamma, A]\!] \to [\![\Gamma]\!]$. To make the notation slightly less cumbersome we'll write Γ for $[\![\Gamma]\!]$, $\Gamma.A$ for $[\![\Gamma, A]\!]$ and $\pi_A : \Gamma.A \to \Gamma$ for $[\![A]\!] \in \mathbb{C}/\Gamma$.

The empty context $\diamond$ is then interpreted as the terminal object 1, and thus the interpretation of a context $A_1, \ldots, A_n \vdash$ is a chain of morphisms

$$A_1.\cdots.A_n \xrightarrow{\pi_{A_n}} \cdots \longrightarrow A_1.A_2 \xrightarrow{\pi_{A_2}} A_1 \xrightarrow{\pi_{A_1}} 1 \quad .$$

A term $\Gamma \vdash t : A$ is interpreted as a morphism $[\![t]\!] : \Gamma \to \Gamma.A$ making the diagram below commute:

$$\begin{tikzcd} \Gamma \arrow[rr, "{[\![t]\!]}"] \arrow[dr, "\mathrm{id}_\Gamma"'] & & \Gamma.A \arrow[dl, "\pi_A"] \\ & \Gamma & \end{tikzcd} \quad .$$

The key to the interpretation is the implementation of substitution by pullback. Given a context morphism $\gamma : \Delta \to \Gamma$ with interpretation $[\![\gamma]\!]$ and a type $\Gamma \vdash A$ the interpretation of $\Delta \vdash A[\gamma]$ is required to make the square below a pullback square:

$$\begin{tikzcd} \Delta.A[\gamma] \arrow[r] \arrow[d, "\pi_{A[\gamma]}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Gamma.A \arrow[d, "\pi_A"] \\ \Delta \arrow[r, "{[\![\gamma]\!]}"'] & \Gamma \end{tikzcd} .$$

It is then clear that interpretations of terms $\Delta \vdash t : A[\gamma]$ are in bijection with interpretations of context morphisms $\Delta \to \Gamma.A$ with first component equal to γ, as required by the underlying language.

Unfortunately, unless pullbacks can be assigned so that $\delta^*\gamma^* f = (\gamma \cdot \delta)^* f$ in general, this interpretation will fail to satisfy such equations as $[\![A[\gamma][\delta]]\!] = [\![A[\gamma \cdot \delta]]\!]$. This problem is identified and solved in Hofmann (1994) and in section 3.3.

Essentially however, the idea of the interpretation is that a type $\Gamma \vdash A$ represents an object $A \in \mathbb{C}/\Gamma$, each term of type A is a global section of A in $\mathbb{C}/\Gamma$, and context morphisms $\gamma : \Delta \to \Gamma$ induce a pullback operation $\gamma^* : \mathbb{C}/\Gamma \to \mathbb{C}/\Delta$ which is reflected by the substitution operation $A \mapsto A[\gamma]$.

The Category of Context Morphisms

The contexts and context morphisms of $\mathscr{L}$ themselves form a category $\mathscr{C} \equiv \mathscr{C}(\mathscr{L})$ with a canonical interpretation of $\mathscr{L}$ in $\mathscr{C}$.

First note that there are three context morphisms of particular interest: $\mathrm{id}_\Gamma : \Gamma \to \Gamma$, $\pi_B : \Gamma, B \to \Gamma$ and $t/b : \Gamma \to (\Gamma, b : B)$ as follows, for context $\Gamma = (A_1, \ldots, A_n)$, type $\Gamma \vdash B$ and term $\Gamma \vdash t : B$:

$$\begin{aligned} \mathrm{id}_\Gamma &\equiv (x_1 : A_1,\ \ldots,\ x_n : A_n\ \vdash\ x_1 : A_1,\ \ldots,\ x_n : A_n) \\ \pi_B &\equiv (x_1 : A_1,\ \ldots,\ x_n : A_n,\ b : B\ \vdash\ x_1 : A_1,\ \ldots,\ x_n : A_n) \\ t/b &\equiv (x_1 : A_1,\ \ldots,\ x_n : A_n\ \vdash\ x_1 : A_1,\ \ldots,\ x_n : A_n,\ t : B) \end{aligned}$$

It will also be convenient to write π'_B for the term $\Gamma, b : B \vdash b : B$, which allows us to write $\mathrm{id}_{\Gamma,B} = (\pi_B, \pi'_B)$; the following equations now follow from the definitions:

$$\pi_B \cdot (\gamma, t) = \gamma \qquad \pi'_B[(\gamma, t)] = t \qquad \gamma = (\pi_B \cdot \gamma, \pi'_B[\gamma]) \ .$$

It is clear that contexts and context morphisms form a category with identity id_Γ and that the assignment $(\Gamma \vdash A) \mapsto \pi_A$ gives us the first part of a naïve interpretation (where each context is interpreted as itself).

Similarly, the interpretation of a term $\Gamma \vdash t : A$ is just the one variable substitution morphism t/b (and indeed $\pi_A \cdot (t/b) = \mathrm{id}_\Gamma$, as required).

This particular interpretation of $\mathscr{L}$ works without any qualification, which means that we can work in the category $\mathscr{C}(\mathscr{L})$ with the naïve interpretation if desired. We can see that $\mathscr{C}(\mathscr{L}_\mathbb{C}) \simeq \mathbb{C}$, so nothing categorical will be lost by this translation.

To complete this interpretation we need to know that we can indeed interpret substitution by pullback, as shown by the following proposition.

Proposition 2.3.1 *In $\mathscr{C}(\mathscr{L})$ substitution along $\gamma : \Delta \to \Gamma$ creates pullbacks: for any $\Gamma \vdash A$ the context morphism $\pi_{A[\gamma]}$ is the pullback along γ of π_A.*

Proof. We want to show that there is a pullback square of context morphisms

$$\begin{tikzcd} \Delta, A[\gamma] \arrow[r, "\gamma_A"] \arrow[d, "\pi_{A[\gamma]}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Gamma, A \arrow[d, "\pi_A"] \\ \Delta \arrow[r, "\gamma"'] & \Gamma \end{tikzcd} .$$

Define $\gamma_A \equiv (\gamma \cdot \pi_{A[\gamma]}, \pi'_{A[\gamma]})$ and observe that by construction $\pi_A \cdot \gamma_A = \gamma \cdot \pi_{A[\gamma]}$. Now let Ξ be a context with morphisms $\delta : \Xi \to \Delta$ and $\xi : \Xi \to \Gamma, A$ satisfying $\pi_A \cdot \xi = \gamma \cdot \delta$.

For any possible factorisation $\alpha : \Xi \to \Delta, A[\gamma]$ satisfying $\pi_{A[\gamma]} \cdot \alpha = \delta$ and $\gamma_A \cdot \alpha = \xi$ the definition of γ_A allows us to write $\gamma_A \cdot \alpha = (\gamma \cdot \pi_{A[\gamma]} \cdot \alpha, \pi'_{A[\gamma]}[\alpha])$ so in particular $\pi'_{A[\gamma]}[\alpha] = \pi'_A[\gamma_A \cdot \alpha] = \pi'_A[\xi]$, and this equation in combination with $\pi_{A[\gamma]} \cdot \alpha = \delta$ fully determines the unique factorisation $\alpha = (\delta, \pi'_A[\xi])$. □

This completes the construction of the interpretation of $\mathscr{L}$ in $\mathscr{C}(\mathscr{L})$. This gives us a category generated from the language together with a canonical interpretation of the language in the category.

Local Morphisms

We can take this construction still further to construct a category $\mathscr{C}_\Gamma(\mathscr{L})$ over each context Γ in $\mathscr{L}$ by taking all types of the form $\Gamma \vdash A$ to be objects and defining a "local morphism" $f : (\Gamma \vdash A) \to (\Gamma \vdash B)$ to be a term of the form $\Gamma, a : A \vdash f : B$. In this case where B does not depend on $a : A$ it can be helpful to write B^+ as a reminder of this: $\Gamma, A \vdash B = B[\pi_A] = B^+$.

Proposition 2.3.2 *The system $\mathscr{C}_\Gamma \equiv \mathscr{C}_\Gamma(\mathscr{L})$ of types in contexts and local morphisms described above forms a category, and each context morphism $\gamma : \Delta \to \Gamma$ induces a substitution functor $\gamma^* : \mathscr{C}_\Gamma \to \mathscr{C}_\Delta$ with $\mathrm{id}_\Gamma^* = \mathrm{id}_{\mathscr{C}_\Gamma}$ and $(\gamma \cdot \delta)^* = \delta^* \gamma^*$.*

Proof. Take $\Gamma, A \vdash \pi'_A : A^+$ to be the identity and define the composite of $\Gamma, A \vdash f : B^+$ and $\Gamma, B \vdash g : C^+$ to be $\Gamma, A \vdash g \cdot f \equiv g[(\pi_A, f)]$.

To verify the category equations calculate $f \cdot \pi'_A = f[(\pi_A, \pi'_A)] = f[\mathrm{id}_{\Gamma,A}] = f$ and $\pi'_B \cdot f = \pi'_B[(\pi_A, f)] = f$; for associativity calculate

$$\begin{aligned}(h \cdot g) \cdot f &= h[(\pi_B, g)][(\pi_A, f)] = h[(\pi_B, g) \cdot (\pi_A, f)] \\ &= h[(\pi_B \cdot (\pi_A, f), g[(\pi_A, f)])] = h[(\pi_A, g[(\pi_A, f)])] = h \cdot (g \cdot f) \ .\end{aligned}$$

Given $\gamma : \Delta \to \Gamma$ define γ^* to take $\Gamma \vdash A$ to $\Delta \vdash A[\gamma]$ and $\Gamma, A \vdash f : B^+$ to the local morphism $\Delta, A[\gamma] \vdash f[\gamma] : B^+[\gamma] = B[\gamma]^+$. This clearly defines a collection of functors satisfying the given equations. □

Note that for each Γ there is a full and faithful functor $\mathscr{C}_\Gamma \to \mathscr{C}/\Gamma$ taking $\Gamma \vdash A$ to π_A, but that for general $\mathscr{L}$ this functor is *not* an equivalence. In this thesis we'll be concentrating on $\mathscr{L} = \mathscr{L}_{\mathbb{C}}$ where $\mathscr{C}_\Gamma \simeq \mathscr{C}/\Gamma$ *does* hold.

2.4 Dependent Type Constructions

In practice we will need six type constructions in the language, namely products, coproducts, exponentials, Sigma types, Pi types and equality types together with two constant types 0 and 1. For conciseness of presentation we can define products using Sigma types and exponentials using Pi types, so given $\Gamma \vdash A, B$ we can define $\Gamma \vdash A \times B \equiv \sum a : A.B$ and $\Gamma \vdash A \Rightarrow B \equiv \prod a : A.B$.

These can all be defined in the same concise style presented as type forming rules, term forming rules and equations. These rules can all be incorporated into the dependently typed language as suitable type and term forming operations combined with equations as shown in figures 2.2 and 2.3.

Note that the rather clumsy term $(x \mapsto \kappa a.t \,[\!]\, \kappa' b.u)$ is not used elsewhere here; instead, wherever possible I use the categorical notation of writing $[f; g] : A + B \to C$ for $f : A \to C$ and $g : B \to C$, and indeed $(x \mapsto \kappa a.t \,[\!]\, \kappa' b.u) \equiv [\lambda a : A.t; \lambda b : B.u]x$.

These type constructions all generate categorical structure on each $\mathscr{C}_\Gamma$ as follows.

Type Forming Rules

$$\frac{\Gamma,\, a{:}A \vdash B}{\Gamma \vdash \sum a{:}A.\ B} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A\times B} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A+B} \qquad \frac{\Gamma \vdash}{\Gamma \vdash 1}$$

$$\frac{\Gamma,\, a{:}A \vdash B}{\Gamma \vdash \prod a{:}A.\ B} \qquad \frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A\Rightarrow B} \qquad \frac{\Gamma \vdash A}{\Gamma,\, a,a'{:}A \vdash \mathrm{Eq}_A(a,a')} \qquad \frac{\Gamma \vdash}{\Gamma \vdash 0}$$

Introduction Rules

$$\frac{\Gamma,\, a{:}A \vdash b{:}B}{\Gamma \vdash (a,b){:}\sum a{:}A.\ B} \qquad \begin{array}{c}\Gamma,\, a{:}A \vdash \kappa a{:}A+B \\ \Gamma,\, b{:}B \vdash \kappa' b{:}A+B\end{array} \qquad \frac{\Gamma \vdash}{\Gamma \vdash\, !_\Gamma{:}1}$$

$$\frac{\Gamma,\, a{:}A \vdash bB}{\Gamma \vdash (\lambda a{:}A.\ b){:}\prod a{:}A.\ B} \qquad \frac{\Gamma \vdash A}{\Gamma,\, a{:}A \vdash \mathrm{refl}_A(a){:}\mathrm{Eq}_A(a,a)}$$

Elimination Rules

$$\begin{array}{c}\Gamma,\, c{:}\sum a{:}A.B \vdash \pi c{:}A \\ \Gamma,\, c{:}\sum a{:}A.B \vdash \pi' c{:}B[\pi c/a]\end{array} \qquad \frac{\begin{array}{c}\Gamma,\, x{:}A+B,\, \Delta \vdash C \\ \Gamma,\, a{:}A,\, \Delta[\kappa a/x] \vdash t{:}C[\kappa a/x] \\ \Gamma,\, b{:}B,\, \Delta[\kappa' b/x] \vdash u{:}C[\kappa' b/x]\end{array}}{\Gamma,\, x{:}A+B,\, \Delta \vdash (x \Mapsto \kappa a.\ t \mathbin{[\!]} \kappa' b.\ u){:}C}$$

$$\frac{\Gamma \vdash t{:}A \quad \Gamma \vdash f{:}\prod a{:}A.\ B}{\Gamma \vdash ft{:}B[t/a]} \qquad \frac{\begin{array}{c}\Gamma \vdash t{:}A \quad \Gamma \vdash u{:}A \\ \Gamma \vdash p{:}\mathrm{Eq}_A(t,u)\end{array}}{\Gamma \vdash t = u{:}A}$$

$$\frac{\Gamma \vdash t{:}0 \quad \Gamma \vdash A}{\Gamma \vdash ?_{A,t}{:}A}$$

Figure 2.2: Term Introduction and Elimination Rules

- If $\mathscr{L}$ has Sigma types then for each $\Gamma \vdash A$ the substitution functor $\pi_A^*{:}\mathscr{C}_\Gamma \to \mathscr{C}_{\Gamma,A}$ has a left adjoint $\sum_A \dashv \pi_A^*$ which commutes with each substitution $\gamma{:}\Delta \to \Gamma$ so that $(\sum_A B)[\gamma] = \sum_{A[\gamma]} B[\gamma_A]$, and such that the map constructing a Sigma type from its context $(\pi_A \cdot \pi_B, (\pi'_A[\pi_B], \pi' B)){:}\Gamma, A, B \to \Gamma, \sum_A B$ is an isomorphism.

  The first of these is the Beck-Chevalley condition and the second tells us that the coproducts are "strong" (Jacobs, 1999, §10.5.2), in other words, strong Sigma types correspond to composition of display maps.

- If $\mathscr{L}$ has Sigma types and the type 1 then each $\mathscr{C}_\Gamma$ has finite products preserved "on the nose" by every substitution functor.

$$\pi(a,b) = a \qquad\qquad (\pi c, \pi' c) = c$$

$$\pi'(a,b) = b$$

$$(\kappa c \mapsto \kappa a.\ t \ [\!]\ \kappa' b.\ u) = t[c/a] \qquad (c \mapsto \kappa a.\ t[\kappa a/x] \ [\!]\ \kappa' b.\ t[\kappa' b/x]) = t[c/x]$$

$$(\kappa' c \mapsto \kappa a.\ t \ [\!]\ \kappa' b.\ u) = u[c/b]$$

$$(\lambda a : A.\ b)t = b[t/a] \qquad\qquad \lambda a : A.\ fa = f$$

$$\Gamma, A, A, p : \mathrm{Eq}_A, q : \mathrm{Eq}_A \ \vdash\ p = q \qquad\qquad t = !_\Gamma$$

Figure 2.3: Equations for Types

- If $\mathscr{L}$ has coproduct types and the type 0 then each $\mathscr{C}_\Gamma$ has finite distributive coproducts which are preserved by substitution functors.
- If $\mathscr{L}$ has Pi types then for each $\Gamma \vdash A$ the substitution functor π_A^* has a right adjoint $\pi_A^* \dashv \prod_A$ such that $(\prod_A B)[\gamma] = \prod_{A[\gamma]} B[\gamma_A]$.
- If $\mathscr{L}$ has Pi types and products then each $\mathscr{C}_\Gamma$ is cartesian closed with exponentials preserved by substitution.

Chapter 3

Fibrations and Families

Here we develop the categorical framework required for a strictly formal interpretation of the internal language and complete the development started in the previous chapter.

3.1 An Introduction to Fibrations

In chapter 2 we developed an internal language suitable for interpretation in a locally cartesian closed category. Here we develop the categorical framework required to complete this interpretation.

This thesis extensively uses the machinery of fibrations and indexed categories (Bénabou, 1975, 1985; Jacobs, 1999; Paré and Schumacher, 1978; Borceux, 1994) to develop the key properties of container categories. In particular the fullness of the functor T taking each container to its extension as a functor relies on the use of fibred natural transformations between fibred functors.

We therefore begin with rapid overview of fibrations over a fixed base category $\mathbb{C}$ (it will not be necessary in this thesis to consider changes of base between fibrations, but of course this is an essential part of the complete development of the theory).

We begin with the notion of an "indexed category": this is, as we will see in theorem 3.1.7, essentially the presentation of a fibration.

Definition 3.1.1 *An* indexed category *over a base category* $\mathbb{C}$ *is given by a pseudo-functor* $\mathscr{E} : \mathbb{C}^{\mathrm{op}} \to \mathbf{Cat}$. *This assigns to each* $\Gamma \in \mathbb{C}$ *a category* $\mathscr{E}_\Gamma$ *called the* fibre of $\mathscr{E}$ over Γ *and to each* $\gamma : \Delta \to \Gamma$ *in* $\mathbb{C}$ *a* reindexing functor $\gamma^* : \mathscr{E}_\Gamma \to \mathscr{E}_\Delta$ *together with* *canonical isomorphisms* $\mathrm{id}_\Gamma^* \cong \mathrm{id}_{\mathscr{E}_\Gamma}$ *and* $\delta^*\gamma^* \cong (\gamma \cdot \delta)^*$*, satisfying certain coherence equations.*

When the canonical isomorphisms are identities $\mathrm{id}_\Gamma^* = \mathrm{id}_{\mathscr{E}_\Gamma}$ *and* $\gamma^*\delta^* = (\delta \cdot \gamma)^*$ *the pseudo functor* $\mathscr{E}$ *is strict and is called a* split indexed category.

An object $X \in \mathscr{E}_\Gamma$ of a fibre category can usefully be thought of as a Γ-indexed collection of elements of $\mathscr{E}$. Thus for each generalised element $\gamma : \Delta \to \Gamma$ there is an object $\gamma^* X \in \mathscr{E}_\Delta$ and so we can in general write $X = (X_\gamma)_{\gamma:\Gamma}$.

The indexed categories over a given base category can be made into a 2-category by the appropriate definition of "indexed functor" and "indexed natural transformation".

Definition 3.1.2 *An* indexed functor *between indexed categories* $\mathscr{E}$ *and* $\mathscr{E}'$ *over* $\mathbb{C}$ *is given by a pseudo-natural transformation* $F : \mathscr{E} \to \mathscr{E}'$*, or equivalently for each* $\Gamma \in \mathbb{C}$ *a functor* $F_\Gamma : \mathscr{E}_\Gamma \to \mathscr{E}'_\Gamma$ *such that for each* $\gamma : \Delta \to \Gamma$ *there is a canonical isomorphism* $\gamma^* F_\Gamma \cong F_\Delta \gamma^*$*. A* split indexed functor *between split indexed categories is an indexed functor where these isomorphisms are identities.*

An indexed natural transformation $\alpha : F \to G$ *assigns to each* Γ *a natural transformation* $\alpha_\Gamma : F_\Gamma \to G_\Gamma$ *such that* $\gamma^* \alpha_\Gamma \cong \alpha_\Delta \gamma^*$ *(this isomorphism refers to identity after composition with the canonical isomorphisms of the functor). If* F *and* G *are split this is an ordinary equality.*

The coherence equations on the isomorphisms $\mathrm{id}_\Gamma^* \cong \mathrm{id}_{\mathscr{C}_\Gamma}$ and $\gamma^*\delta^* \cong (\delta \cdot \gamma)^*$ alluded to above are similarly mentioned in passing in Paré and Schumacher (1978) and are similar to those described in detail for monoidal categories in MacLane (1971). An explicit presentation can be found in Jacobs (1999) or Borceux (1994).

We make some effort to avoid explicit presentation of these coherence equations as taking them into account complicates the presentation and the algebra considerably. This can be done in two ways: firstly, in the definition of a *fibration* below we have a system which is equivalent (modulo observations on choice) to an indexed category but in which the coherence isomorphisms are implicit; secondly, we will present a construction which constructs a split fibration (equivalent to a split indexed category) from any fibration, and which is equivalent to the original fibration.

This effort is directly related to the problems of the naïve interpretation of the internal language discussed in section 2.3. In particular, we will show how the language is interpreted in a framework obtained by splitting a fibration.

A fibration over $\mathbb{C}$ is defined to be a functor into $\mathbb{C}$ satisfying a particular condition involving "cartesian morphisms" (also referred to as "prone" in Johnstone, 2002 and Taylor, 1999), so first we must define what a cartesian morphism is.

Definition 3.1.3 *Given a functor $p : \mathbb{E} \to \mathbb{C}$, for each $\Gamma \in \mathbb{C}$ write $\mathbb{E}_\Gamma$, called* the fibre of p over Γ, *for the subcategory of $\mathbb{E}$ mapped by p onto Γ. In other words, $X \in \mathbb{E}_\Gamma$ iff $pX = \Gamma$ and $\mathbb{E}_\Gamma(X,Y) = \{ f : X \to Y \text{ in } \mathbb{E} \mid pf = \mathrm{id}_\Gamma \}$. Similarly, for $\gamma : \Delta \to \Gamma$ in $\mathbb{C}$ and $X \in \mathbb{E}_\Delta$, $Y \in \mathbb{E}_\Gamma$ write $\mathbb{E}_\gamma(X,Y) \equiv \{ f : X \to Y \mid pf = \gamma \}$ for the set of morphisms in $\mathbb{E}$* over γ.

The morphisms in $\mathbb{E}_\Gamma$ are over id_Γ and are called "vertical".

Definition 3.1.4 *Given a functor $p : \mathbb{E} \to \mathbb{C}$ a morphism $f : X \to Y$ in $\mathbb{E}$ is said to be* cartesian *(with respect to p) iff for each $Z \in \mathbb{E}$ and each $\gamma : pZ \to pX$ in $\mathbb{C}$ the function $f \cdot - : \mathbb{E}_\gamma(Z,X) \to \mathbb{E}_{pf\cdot\gamma}(Z,Y)$ taking g to $f \cdot g$ is an isomorphism.*

This definition is illustrated by the figure below: if f is cartesian then every $h : Z \to Y$ over $pf \cdot \gamma$ has a factorisation as $h = f \cdot g$ for a unique g over γ thus:

$$\begin{tikzcd} Z \arrow[r, dashed, "g"'] \arrow[rr, bend left, "h"] & X \arrow[r, "f"'] & Y & & \mathbb{E} \arrow[d, "p"] \\ pZ \arrow[r, "\gamma"] & pX \arrow[r, "pf"] & Y & & \mathbb{C} \end{tikzcd} .$$

Note in particular that if there is a cartesian morphism into X over every map γ into pX in $\mathbb{C}$ then *every* map into X has a unique (up to isomorphism) factorisation as a vertical morphism followed by a cartesian morphism.

Now we can define a fibration to be a functor equipped with "enough" cartesian morphisms.

Definition 3.1.5 *A functor $p : \mathbb{E} \to \mathbb{C}$ is a* fibration *iff for every $X \in \mathbb{E}$ and every map γ into pX in $\mathbb{C}$ there exists a cartesian morphism into X over γ.*

A cleavage *on p is a* choice *of cartesian morphism $\overline{\gamma}_X : \gamma^* X \to X$ for each X and γ. A cleavage is* split *iff it satisfies equations $\overline{(\mathrm{id}_\Gamma)}_X = \mathrm{id}_X$ and $\overline{(\gamma \cdot \delta)}_X = \overline{\gamma}_X \cdot \overline{\delta}_{\gamma^* X}$.*

A split fibration *is a fibration together with a split cleavage (the* splitting *of p).*

A fibration with a cleavage is also said to be *cloven*. We will assume throughout this thesis that all of our fibrations are cloven: this cleavage can either be assumed to already exist, or else to arise from a suitable choice principle in the meta-theory. We won't be looking at constructions where the construction of a cleavage is problematic.

We can now define a 2-category of fibrations, fibred functors and fibred natural transformations over $\mathbb{C}$.

Definition 3.1.6 *Given fibrations* $p:\mathbb{E}\to\mathbb{C}$ *and* $q:\mathbb{F}\to\mathbb{C}$ *a* fibred functor $F:p\to q$ *is a functor* $F:\mathbb{E}\to\mathbb{F}$ *such that* $qF=p$ *which takes cartesian morphisms in* $\mathbb{E}$ *to cartesian morphisms in* $\mathbb{F}$*. A fibred functor between split fibrations is a* split fibred functor *iff it preserves the splitting, ie* $F\overline{\gamma}_X=\overline{\gamma}_{FX}$*.*

A fibred natural transformation $\alpha:F\to G$ *between fibred functors is a natural transformation with vertical components, ie* $p\alpha X=\mathrm{id}_{pX}$ *for each* $X\in\mathbb{E}$*.*

Write $\mathbf{Fib}_{\mathbb{C}}$ for the 2-category of fibrations, fibred functors and fibred natural transformations over $\mathbb{C}$ and $\mathbf{Fib}_{\mathbb{C}}^{\mathrm{split}}$ for the 2-category of split fibrations, split fibred functors and fibred natural transformations.

We now have enough machinery in place to state the following theorem which tells us that fibrations and indexed categories are essentially the same thing.

Theorem 3.1.7 *The 2-categories of indexed categories and of fibrations over a common base are weakly equivalent. Similarly the 2-categories of split indexed categories and split fibrations are equivalent.*

To be precise, the 2-functors between indexed categories and fibrations establish an equivalence between each fibration and a class of corresponding indexed categories (one indexed category for each cleavage on the fibration). On the other hand, the 2-functors between split indexed categories and split fibrations establish an isomorphism between each fibration and its corresponding indexed category.

Proof. I will sketch the construction of an indexed category from a fibration and vice versa, but for further details see Jacobs (1999) or Borceux (1994).

To construct an indexed category from a fibration observe that each cleavage on a fibration $p:\mathbb{E}\to\mathbb{C}$ allows us to construct an indexed category. First, to each Γ assign the fibre $\mathbb{E}_\Gamma$. The construction of $(\gamma,X)\mapsto(\overline{\gamma}_X:\gamma^*X\to X)$ given by the cleavage is sufficient to construct functors $\gamma^*:\mathbb{E}_\Gamma\to\mathbb{E}_\Delta$ for each $\gamma:\Delta\to\Gamma$, and it is a straightforward calculation to discover that the necessary isomorphisms exist and satisfy the coherence equations.

Conversely, given an indexed category $\mathscr{E} : \mathbb{C}^{\mathrm{op}} \to \mathbf{Cat}$ a total category $\int_{\mathbb{C}} \mathscr{E}$ called the *Grothendieck completion* of $\mathscr{E}$ can be constructed with a projection $\int_{\mathbb{C}} \mathscr{E} \to \mathbb{C}$ making this into a fibration. The objects of $\int_{\mathbb{C}} \mathscr{E}$ are the pairs $(\Gamma \in \mathbb{C}, X \in \mathscr{E}_\Gamma)$ and a morphism $(\Delta, Y) \to (\Gamma, X)$ is given by a pair of morphisms $(\gamma : \Delta \to \Gamma, f : Y \to \gamma^* X)$. The projection $\int_{\mathbb{C}} \mathscr{E} \to \mathbb{C}$ is obvious, and each morphism of the form $(\gamma, \mathrm{id}_{\gamma^* X})$ is cartesian. The canonical isomorphisms and their coherence equations play an essential role in this construction.

If p is a split fibration then the reindexing functors γ^* generated from the splitting for a split indexed category. Conversely, the Grothendieck completion of a split indexed category generates a split fibration. $\square$

In fact, this theorem establishes a correspondence between indexed categories and fibrations with a specified cleavage: the reindexing functors between fibres correspond precisely to a cleavage.

The point of this theorem is that we can work with the fibration representation of a system or its corresponding indexed category representation, depending on which is more convenient. In practice it is often easier to work with indexed categories when the fibration is split, but otherwise working with the fibration can provide a more uniform perspective. Indeed: "an indexed category is just a presentation of a fibred category" (Bénabou, 1985).

3.2 Examples of Fibrations

The most important fibration, and the one central to the development of this thesis, is given by the slice categories of a pullback complete category.

Example 3.2.1 *If $\mathbb{C}$ is pullback complete then the assignment $\Gamma \mapsto \mathbb{C}/\Gamma$ is the object part of an indexed category with reindexing functors given by pullbacks.*

To be more precise, given $\gamma : \Delta \to \Gamma$ in $\mathbb{C}$ the reindexing functor $\gamma^* : \mathbb{C}/\Gamma \to \mathbb{C}/\Delta$ takes each object $(f : A \to \Gamma) \in \mathbb{C}/\Gamma$ to the left hand side of the pullback square below

$$\begin{tikzcd} \gamma^* A \arrow[r] \arrow[d, "\gamma^* f"'] \arrow[dr, phantom, "\lrcorner", very near start] & A \arrow[d, "f"] \\ \Delta \arrow[r, "\gamma"'] & \Gamma \end{tikzcd}$$

regarded as an object of $\mathbb{C}/\Delta$. This can be referred to as the slice indexed category.

The corresponding fibration is the *codomain fibration* $\mathrm{cod}_{\mathbb{C}} : \mathbb{C}^{\to} \to \mathbb{C}$: objects of $\mathbb{C}^{\to}$ are arrows in $\mathbb{C}$, morphisms in $\mathbb{C}^{\to}$ are commutative squares in $\mathbb{C}$, and the projection $\mathrm{cod}_{\mathbb{C}}$ takes $f : X \to Y \in \mathbb{C}^{\to}$ to $\mathrm{cod}_{\mathbb{C}} f \equiv Y$.

In the 2-category of fibrations the codomain fibration effectively plays the role of the base category $\mathbb{C}$. In particular in definition 3.2.5 below we make explicit use of this.

The objects of a slice category $\mathbb{C}/\Gamma$ can usefully be regarded as "Γ-indexed families of objects in $\mathbb{C}$", and in the case of $\mathbb{C} \equiv \mathbf{Set}$ this can be made more explicit.

Example 3.2.2 *The assignment sending each set $I \in \mathbf{Set}$ to the category of I-indexed sets and functions is an indexed category, referred to as the* family fibration **Fam***.*

The objects of $\mathbf{Fam}_I$ *are tuples $\vec{X} = (X_i)_{i\in I}$ with morphisms $\vec{X} \to \vec{Y}$ given by tuples of maps $(f_i : X_i \to Y_i)_{i\in I}$. A function $u : J \to I$ induces a pullback functor u^* defined by $u^*(X_i)_{i\in I} \equiv (X_{uj})_{j\in J}$.*

The following proposition makes it clear that, for **Set** at least, families and slices are equivalent.

Proposition 3.2.3 *The slice indexed category for* **Set** *is equivalent to* **Fam***.*

Proof. It is sufficient to show an equivalence $\mathbf{Set}/I \simeq \mathbf{Fam}_I$ for each I compatible with pullbacks and substitution. Assign the family of sets $(\{\, a \in A \mid f(a) = i \,\})_{i\in I}$ to each map $f : A \to I$ in $\mathbf{Set}/I$; conversely, given $(X_i)_{i\in I}$ assign to this the function $\pi : \sum_{i\in I} X_i \to I$ taking each tuple $(i,x) \in \sum_{i\in I} X_i$ to i. These assignments clearly extend to functions and can easily be seen to define an equivalence of categories which extends to an equivalence of indexed categories. $\square$

A construction of interest is the I-indexed power of a fibration.

Example 3.2.4 *Given a fibration $p : \mathbb{E} \to \mathbb{C}$ the fibration p^I is constructed as follows. Define the objects of a category $\mathbb{E}^{(I)}$ to be pairs $(\Gamma \in \mathbb{C}, (X_i \in \mathbb{E}_\Gamma)_{i\in I})$ with morphisms $(\Delta, \vec{Y}) \to (\Gamma, \vec{X})$ given by pairs $(\gamma : \Delta \to \Gamma, (f_i : Y_i \to X_i)_{i\in I})$ such that $pf_i = \gamma$ for all $i \in I$. The projection taking $(\gamma, \vec{f})$ to γ is the fibration p^I.*

The corresponding indexed category description is perhaps clearer: the fibre over Γ of p^I is $\mathbb{E}^I_\Gamma$ and the reindexing functor over $\gamma : \Delta \to \Gamma$ is $(\gamma^*)^I : \mathbb{E}^I_\Gamma \to \mathbb{E}^I_\Delta$.

Of particular interest is the fibration of I-indexed powers of $\mathbb{C}$: in this case the fibre of $\mathrm{cod}_{\mathbb{C}}^I$ is $(\mathbb{C}/\Gamma)^I$. When we talk in this thesis about functors to and from $\mathbb{C}^I$ we will really mean fibred functors to and from $\mathrm{cod}_{\mathbb{C}}^I$. In particular, we have the following definition.

Definition 3.2.5 *Write* $[\mathbb{C}^I, \mathbb{C}^J]$ *for the category of fibred functors and fibred natural transformations* $\mathrm{cod}_{\mathbb{C}}^I \to \mathrm{cod}_{\mathbb{C}}^J$*:*

$$[\mathbb{C}^I, \mathbb{C}^J] \equiv \mathbf{Fib}_{\mathbb{C}}(\mathrm{cod}_{\mathbb{C}}^I,\ \mathrm{cod}_{\mathbb{C}}^J) \ .$$

Note that $[\mathbb{C}^I, \mathbb{C}^J] \cong [\mathbb{C}^I, \mathbb{C}]^J$, since p^I is the I-fold product of p in the 2-category $\mathbf{Fib}_{\mathbb{C}}$.

In the case $\mathbb{C} = \mathbf{Set}$ the categories $[\mathbf{Set}, \mathbf{Set}]$ and $\mathbf{Cat}(\mathbf{Set}, \mathbf{Set})$ are isomorphic: every functor $F : \mathbf{Set} \to \mathbf{Set}$ automatically lifts to a fibred functor $F : \mathbf{Fam} \to \mathbf{Fam}$, which is equivalent to a functor $\mathrm{cod}_{\mathbf{Set}} \to \mathrm{cod}_{\mathbf{Set}}$. It is easy enough to see this: simply define $F(X_i)_{i\in I} \equiv (FX_i)_{i\in I}$.

More generally $[\mathbf{Set}^I, \mathbf{Set}] \simeq \mathbf{Cat}(\mathbf{Set}^I, \mathbf{Set})$, and indeed if we restrict our attention to the split fibred functors between the equivalent family fibrations we get an isomorphism $\mathbf{Fib}_{\mathbf{Set}}^{\mathrm{split}}(\mathbf{Fam}^I, \mathbf{Fam}) \cong \mathbf{Cat}(\mathbf{Set}^I, \mathbf{Set})$. This observation reinforces the importance of the role of fibrations when dealing with a base category $\mathbb{C}$ other than **Set**.

Splitting a Fibration

The observations in this section come from Bénabou (1975, 1985) and Jacobs (1999). Another important fibration is given by the following.

Example 3.2.6 *The functor* $\mathrm{dom}_\Gamma : \mathbb{C}/\Gamma \to \mathbb{C}$ *taking* $\gamma : \Delta \to \Gamma$ *to* Δ *is a split fibration.*

This fibration plays the role analogous to the discrete category generated by a set: the fibration dom_Γ is the discrete fibration generated by an object of $\mathbb{C}$. For instance, the fibre of dom_Γ over Δ is the discrete category generated by $\mathbb{C}(\Delta, \Gamma)$ — we should think here of maps $\gamma : \Delta \to \Gamma$ as "local elements" of Γ in context Δ.

In analogy with the theory of presheaves, where the *representable functors* are functors of the form $\mathbb{C}(-, X) \in \mathbf{Set}^{\mathbb{C}^{\mathrm{op}}}$, so also the fibrations dom_Γ are called the *representable fibrations*. Furthermore, just as the indexed category representation allows us to regard a fibration as an object of a generalised "presheaf" (2-)category $\mathbf{Cat}^{\mathbb{C}^{\mathrm{op}}}$, we do in indeed have a generalisation of the Yoneda lemma to this context.

Theorem 3.2.7 (Fibred Yoneda) *For each fibration $p : \mathbb{E} \to \mathbb{C}$ and each object $\Gamma \in \mathbb{C}$ there is an equivalence of categories*

$$\mathbb{E}_\Gamma \simeq \mathbf{Fib}_{\mathbb{C}}(\mathrm{dom}_\Gamma, p)$$

natural (up to isomorphism) in Γ and p. When p is a split fibration this equivalence restricts to an isomorphism $\mathbb{E}_\Gamma \cong \mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}(\mathrm{dom}_\Gamma, p)$.

Proof. Given $F : \mathrm{dom}_\Gamma \to p$ compute $F\,\mathrm{id}_\Gamma \in \mathbb{E}_\Gamma$. Conversely, given $X \in \mathbb{E}_\Gamma$ construct a functor $\widetilde{X}$ taking $\gamma \in \mathbb{C}/\Delta$ to $\gamma^* X \in \mathbb{E}_\Delta$. Note that this construction of $\widetilde{X}$ depends on a choice of cleavage on p.

Clearly $\widetilde{X}\,\mathrm{id}_\Gamma = \mathrm{id}_\Gamma^* X \cong X$, and conversely, given $F : \mathrm{dom}_\Gamma \to p$ construct $(\widetilde{F\,\mathrm{id}_\Gamma})\gamma = \gamma^* F\,\mathrm{id}_\Gamma \cong F\gamma^*\,\mathrm{id}_\Gamma = F\gamma$; this establishes the equivalence for each Γ, p.

Each morphism $\gamma : \Delta \to \Gamma$ extends to a fibred functor $\mathrm{dom}_\gamma : \mathrm{dom}_\Delta \to \mathrm{dom}_\Gamma$ taking each $\delta : \Xi \to \Delta$ to $\gamma \cdot \delta$, and so we can define $\gamma^* \widetilde{X} \equiv \widetilde{X} \cdot \gamma$. It is now easy to calculate $\mathrm{dom}_\gamma F\,\mathrm{id}_\Gamma = F\gamma = F\gamma^*\,\mathrm{id}_\Gamma \cong \gamma^* F\,\mathrm{id}_\Gamma$. This is enough to show naturality of the bijection in Γ.

Similarly, each functor $F : p \to q$ induces for each $\Gamma \in \mathbb{C}$ a composition functor $\widetilde{F}_\Gamma : \mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}(\mathrm{dom}_\Gamma, p) \to \mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}(\mathrm{dom}_\Gamma, p)$ with $\widetilde{F}_\Gamma X \equiv F \circ X$.

$$\begin{tikzcd}
\mathbb{E}_\Gamma \arrow[r, "F_\Gamma"] \arrow[d, "\gamma^*"'] & \mathbb{D}_\Gamma \arrow[d, "\gamma^*"] \\
\mathbb{E}_\Delta \arrow[r, "F_\Delta"'] & \mathbb{D}_\Delta
\end{tikzcd}
\qquad
\begin{tikzcd}
\Gamma \arrow[r, "\widetilde{X}"] \arrow[dr, "\widetilde{F}_\Gamma \widetilde{X}" near end] & p \arrow[d, "F"] \\
\Delta \arrow[u, "\gamma"] \arrow[ur, "\gamma^* \widetilde{X}" near start] \arrow[r, "\widetilde{F}_\Delta \gamma^* \widetilde{X} = \gamma^* \widetilde{F}_\Gamma \widetilde{X}"'] & q
\end{tikzcd}$$

The diagram above illustrates this situation and shows both naturality of the bijection of the theorem and the fact that $F : p \to q$ induces what can be regarded as a split functor $\mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}(\mathrm{dom}_-, p) \to \mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}(\mathrm{dom}_-, q)$. □

The following corollary follows from $\mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}(\mathrm{dom}_\Delta, \mathrm{dom}_\Gamma) \cong (\mathbb{C}/\Gamma)_\Delta \cong \mathbb{C}(\Delta, \Gamma)$.

Corollary 3.2.8 $\Gamma \mapsto \mathrm{dom}_\Gamma$ *extends to a full and faithful functor* $\mathrm{dom}_- : \mathbb{C} \to \mathbf{Fib}^{\mathrm{split}}_{\mathbb{C}}$.

The functor dom_γ taking $\delta : \Xi \to \Delta$ to $\gamma \cdot \delta$ is more familiarly known as $\sum_\gamma$ in its context as a functor $\sum_\gamma : \mathbb{C}/\Delta \to \mathbb{C}/\Gamma$ left adjoint to γ^*.

We conclude with another corollary of 3.2.7. This result is of particular interest, as already observed, as it hints that we can avoid the technicalities of working with coherence morphisms by working in a split fibration.

Corollary 3.2.9 *Every fibration is equivalent to a split fibration, and this equivalence is part of a pseudo-equivalence* $\mathbf{Fib}_{\mathbb{C}} \sim \mathbf{Fib}_{\mathbb{C}}^{\mathrm{split}}$.

Proof. Given a fibration $p : \mathbb{E} \to \mathbb{C}$ define a split indexed category with fibre $\mathbf{Fib}_{\mathbb{C}}(\mathrm{dom}_\Gamma, p)$ over each Γ and with reindexing functors along γ given by composition with $\mathrm{dom}_\gamma : \mathrm{dom}_\Delta \to \mathrm{dom}_\Gamma$. As we've already seen in the proof to 3.2.7 each functor $F : p \to q$ translates into a split fibred functor. □

Given the importance of this construction, a detailed review will be instructive. Let $p = \mathrm{cod}_{\mathbb{C}}$, the codomain fibration (also known as the slice indexed category), and recall that the fibres of this fibration are the slices $\mathbb{C}/\Gamma$. The problem of coherence arises because pulling back along maps into Γ cannot in general be made associative, in other words pullbacks cannot in general be assigned to make the equation $\delta^*\gamma^* = (\gamma \cdot \delta)^*$ hold.

The splitting construction works by replacing the slice category $\mathbb{C}/\Gamma$ with the equivalent (but much larger) category $\mathbf{Fib}_{\mathbb{C}}(\mathrm{dom}_\Gamma, \mathrm{cod}_{\mathbb{C}})$. An object F of this category can conveniently be thought of as assigning to *every* map γ into Γ a choice of pullback $\gamma^*\pi_F$ of a selected map $\pi_F : \Gamma.F \to \Gamma$.

In other words, each $A \in \mathbb{C}/\Gamma$ is replaced by a set of equivalent objects F in the splitting. All the objects F replacing A agree with A on the basic map $\pi_F = A$, but each F assigns its own choice of pullback $\gamma^*\pi_F$. The beauty of this construction is that because each F is a morphism in $\mathbf{Fib}_{\mathbb{C}}$ we can implement reindexing of F along a map γ into Γ by simple composition:

$$\begin{tikzcd} \mathrm{dom}_\Delta \arrow[rr, "\mathrm{dom}_\gamma"] \arrow[dr, "\gamma^* F \equiv F\,\mathrm{dom}_\gamma"'] & & \mathrm{dom}_\Gamma \arrow[dl, "F"] \\ & \mathrm{cod}_{\mathbb{C}} & \end{tikzcd}$$

and so of course reindexing is necessarily split!

3.3 Categories with Families

The following definition ties the splitting of the codomain fibration that we've just described back to the internal language described in the previous chapter. This definition is from Dybjer (1995) with some minor changes of notation and separating out the terminal object (or "global" context) as a separate condition.

Definition 3.3.1 *A* category with families *(or* cwf*) is given by the following data:*

- *A category* $\mathbb{C}$ *(of* contexts *and* context morphisms*).*
- *A functor* $\mathbb{C}^{\mathrm{op}} \to \mathbf{Fam}$ *defining for each context a set of* types in context *and for each type in context a set of* well typed terms*. For a type A in context Γ write $\Gamma \vdash A$, and for a term t of type A write $\Gamma \vdash t : A$. Functorality provides a* substitution *operation*

$$\frac{\gamma : \Gamma \longrightarrow \Delta \quad \Delta \vdash t : A}{\Gamma \vdash t[\gamma] : A[\gamma]}$$

satisfying the equations $A[\mathrm{id}_\Gamma] = A$ and $A[\gamma][\delta] = A[\gamma \cdot \delta]$ for types and similarly $t[\mathrm{id}_\Gamma] = t$ and $t[\gamma][\delta] = t[\gamma \cdot \delta]$ for terms.

- *Context comprehension: to each type $\Gamma \vdash A$ associate a morphism $\pi_A : \Gamma.A \to \Gamma$ in $\mathbb{C}$, called the* display map *of A, and a term $\Gamma.A \vdash \pi'_A : A[\pi_A]$, the* variable *of A, together with a bijection*

$$\begin{array}{c} \gamma : \Delta \longrightarrow \Gamma \quad \Delta \vdash t : A[\gamma] \\ \hline\hline \langle \gamma, t \rangle : \Delta \longrightarrow \Gamma.A \end{array}$$

satisfying equations $\pi_A \cdot \langle \gamma, t \rangle = \gamma$, $\pi'_A[\langle \gamma, t \rangle] = t$ and $\langle \pi_A, \pi'_A \rangle = \mathrm{id}_{\Gamma.A}$.

If $\mathbb{C}$ has a terminal object say that the category with families has a global context.

The following definitions and abbreviations will prove useful.

Definition 3.3.2 *For $\Gamma \vdash A$, $\Gamma \vdash B$, $\Gamma \vdash t : B$, $\gamma : \Gamma \to \Delta$, $\Gamma.A \vdash D$ define:*

$$\Gamma.A \vdash B^+ \equiv B[\pi_A]$$

$$\Gamma.A \vdash t^+ \equiv t[\pi_A] : B[\pi_A]$$

$$\gamma_A \equiv \langle \gamma \cdot \pi_{A[\gamma]}, \pi'_{A[\gamma]} \rangle : \Delta.A[\gamma] \longrightarrow \Gamma.A$$

$$\gamma_{A,D} \equiv (\gamma_A)_D : \Delta.A[\gamma].D[\gamma_A] \longrightarrow \Gamma.A.D$$

The following equations can readily be derived from the definitions

$$\langle\gamma,t\rangle\cdot\delta=\langle\gamma\cdot\delta,t[\delta]\rangle\qquad\gamma_A\cdot\langle\delta,t\rangle=\langle\gamma\cdot\delta,t\rangle\qquad(\gamma\cdot\delta)_A=\gamma_A\cdot\delta_{A[\gamma]}\ .$$

Note that substitution of display maps creates pullbacks. This result is equivalent to proposition 2.3.1, and the proof is pretty much the same.

Proposition 3.3.3 *For type in context $\Gamma\vdash A$ and context morphism $\gamma:\Gamma\to\Delta$ in a category with families the following square is a pullback:*

$$\begin{tikzcd}
\Delta.A[\gamma] \arrow[r, "\gamma_A"] \arrow[d, "\pi_{A[\gamma]}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Gamma.A \arrow[d, "\pi_A"] \\
\Delta \arrow[r, "\gamma"'] & \Gamma
\end{tikzcd}\ .$$

Proof. Given a cone $\delta:\Xi\to\Delta$ and $f:\Xi\to\Gamma.A$ with $\pi_A\cdot f=\gamma\cdot\delta$ then any factorisation $g:\Xi\to\Delta.A[\gamma]$ must satisfy equations $\pi_{A[\gamma]}\cdot g=\delta$ and $\gamma_A\cdot g=f$. We can now calculate $\pi'_{A[\gamma]}[g]=\pi'_A[\gamma_A][g]=\pi'_A[\gamma_A\cdot g]=\pi'_A[f]$ and conclude that $g=\langle\gamma\cdot\delta,\pi'_A[f]\rangle$ is the required unique factorisation. □

There is a very close relationship between the notion of a category with families and the internal language defined in section 2.2. Compare the following definition with the naïve interpretation in section 2.3.

Definition 3.3.4 *An* interpretation *of a dependently typed language $\mathscr{L}$ in a category with families $\mathscr{C}$ with category of contexts $\mathbb{C}$ and a global context is given by the following structure.*

- *Contexts in $\mathscr{L}$ are interpreted as objects of $\mathbb{C}$, types in $\mathscr{L}$ are types in context in $\mathscr{C}$, and terms are interpreted as well typed terms in $\mathscr{C}$. We'll write $[\![-]\!]$ for the interpretation function in all cases.*
- *The empty context $\diamond$ is interpreted as the terminal object in $\mathbb{C}$, ie $[\![\diamond]\!]=1$.*
- *Given $\Gamma\vdash A$ in $\mathscr{L}$ the interpretation of Γ,A is the domain of the display map $\pi_{[\![A]\!]}$, ie $[\![\Gamma,A]\!]=[\![\Gamma]\!].[\![A]\!]$ in $\mathbb{C}$.*
- *A variable $\Gamma,x:A\vdash x:A$ is interpreted as the term $[\![\Gamma,A]\!]\vdash\pi'_A:[\![A]\!][\pi_A]$. More generally, a variable $\Gamma,x:A,y_1:B_1,\dots,y_n:B_n\vdash x:A$ in $\mathscr{L}$ is interpreted in $\mathscr{C}$ as the term*

$$[\![\Gamma,A,B_1,\dots,B_n]\!]\ \vdash\ \pi'_A[\pi_{B_1}\cdot\dots\cdot\pi_{B_n}]:[\![A]\!][\pi_A\cdot\pi_{B_1}\cdot\dots\cdot\pi_{B_n}]\ .$$

- *Filling in place-holders in $\mathscr{L}$ is interpreted by substitution in $\mathscr{C}$, so we have* $[\![A(\gamma)]\!] = [\![A]\!][[\![\gamma]\!]]$ *and* $[\![f(\gamma)]\!] = [\![f]\!][[\![\gamma]\!]]$.
- *The interpretation of the empty context morphism is* $[\![\diamond_\Gamma]\!] : [\![\Gamma]\!] \to [\![\diamond]\!] = 1$.
- *Context morphisms are combined thus:* $[\![(\gamma, t)]\!] = \langle [\![\gamma]\!], [\![t]\!] \rangle$.

By checking the equations (2.1) used to define substitution and composition in $\mathscr{L}$ we can see that substitution in $\mathscr{L}$ is interpreted as substitution in $\mathscr{C}$ and context morphism composition in $\mathscr{L}$ becomes composition in $\mathbb{C}$.

In this thesis I am concerned with the categorical models of the language rather than with the language itself: the internal language is developed here as a tool with which to reason about the category $\mathbb{C}$ (and some other related structures). The development of the equational part of the theory will largely be taken for granted, so the extension of the definition of an interpretation of a language to include the equality judgements of a dependently typed theory will not be developed here.

An important point is that the internal language as described here and the structure of a category with families are very close indeed. The following proposition captures part of this.

Proposition 3.3.5 *Every category with families with a global context has an associated dependently typed theory with a canonical interpretation. Conversely, every dependently typed theory can be used to generate a category with families with associated theory equal to the original theory.*

Proof. Given a category with families $\mathscr{C}$ define the language $\mathscr{L}$ inductively in a fairly obvious way. The types of $\mathscr{L}$ in the empty context are precisely the types $1 \vdash A$ of $\mathscr{C}$; given $\Gamma \vdash A$, the types in the context Γ, A in $\mathscr{L}$ are the types $\Gamma.A \vdash B$, and so forth. As this construction proceeds (with similar constructions for terms), the interpretation according to definition 3.3.4 is simultaneously defined.

Conversely, the construction of the category of context morphisms from a language (section 2.3 can be used to construct a category with families.

Note that in passing from a language to a cwf and back again, essentially the same language (with a rather fuller presentation) is recovered. On the other hand, when passing from a cwf to a language and back again, only those contexts in $\mathbb{C}$ which can be built up inductively from types are recovered. $\square$

Informally this proposition is saying that there is a reflective inclusion of theories into categories with families thus

$$\mathit{Theory} \underset{\longleftarrow}{\overset{\hookrightarrow}{}} \mathit{Cwf} \quad .$$

However, as we have not defined an appropriate notion of morphism between theory (or cwfs) this remains informal. One thing is stopping this correspondence being an equivalence, and that is the key role of the empty context in the language.

The simplest possible example of a category with families is given by the following.

Example 3.3.6 *For any category $\mathbb{C}$ there are two trivial categories of families with $\mathbb{C}$ the category of contexts:*

- *If the set of types in context is empty then the empty cwf with no types or terms is obtained. The corresponding language is empty, and its category of contexts is the single morphism category.*
- *If the set of types in each context is a singleton $\Gamma \vdash 1_\Gamma$ with one term $\Gamma \vdash !_\Gamma : 1_\Gamma$ then a cwf is obtained by defining $\pi_{1_\Gamma} \equiv \mathrm{id}_\Gamma$. The corresponding language is non empty but trivial, and again the category of contexts of the language is the singleton category.*

The canonical example of a category with families is given by the internal language of a pullback complete category. This construction will be used implicitly throughout the rest of this thesis.

Proposition 3.3.7 *If $\mathbb{C}$ is pullback complete then a category with families with category of contexts $\mathbb{C}$ can be constructed by defining the types in context $\Gamma \vdash A$ to be the fibred functors $A : \mathrm{dom}_\Gamma \to \mathrm{cod}_\mathbb{C}$ with display maps $\pi_A \equiv A\,\mathrm{id}_\Gamma$.*

Proof. Define substitution of types by composition with γ, ie $A[\gamma] \equiv A\gamma$ (which we can also call $\gamma^* A$); this is clearly strictly functorial in γ.

Given a substitution $\gamma : \Delta \to \Gamma$ and a type $\Gamma \vdash A$ computing $A\gamma$ as a morphism in $\mathbb{C}^\to$ (regarding γ as a morphism $\gamma \to \mathrm{id}_\Gamma$ in $\mathbb{C}/\Gamma$) yields the right hand pullback square below

$$\begin{tikzcd}
\Xi.A[\gamma\cdot\delta] = \Xi.A[\gamma][\delta] \arrow[r, "\delta_{A[\gamma]}"] \arrow[d, "\pi_{A[\gamma\cdot\delta]}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Delta.A[\gamma] \arrow[r, "\gamma_A"] \arrow[d, "\pi_{A[\gamma]}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Gamma.A \arrow[d, "\pi_A"] \\
\Xi \arrow[r, "\delta"'] & \Delta \arrow[r, "\gamma"'] & \Gamma
\end{tikzcd} .$$

We will frequently use the notation γ_A for the lifting of γ as above. Note of course that $\gamma_{\mathrm{id}_\Gamma} = \gamma$, and it's easy to see above that $(\gamma\cdot\delta)_A = \gamma_A \cdot \delta_{A[\gamma]}$.

Define the well typed terms $\Gamma \vdash t : A$ to be morphisms $t : \mathrm{id}_\Gamma \to \pi_A$ in $\mathbb{C}/\Gamma$, ie splittings of π_A. Given a substitution γ define $t[\gamma] : \Delta \to \Delta.A[\gamma]$ to be the unique morphism satisfying $\pi_{A[\gamma]} \cdot t[\gamma] = \mathrm{id}_\Delta$ and $\gamma_A \cdot t[\gamma] = t \cdot \gamma$ thus:

$$\begin{tikzcd}
\Delta.A[\gamma] \arrow[r, "\gamma_A"] \arrow[d, "\pi_{A[\gamma]}"] \arrow[dr, phantom, "\lrcorner", very near start] & \Gamma.A \arrow[d, "\pi_A"'] \\
\Delta \arrow[r, "\gamma"'] \arrow[u, dashed, bend left=60, "t[\gamma]"] & \Gamma \arrow[u, bend right=60, "t"']
\end{tikzcd} .$$

Clearly $t[\mathrm{id}_\Gamma] = t$; so it remains to check the substitution of composites: $\pi_\Xi \cdot (t[\gamma][\delta]) = \mathrm{id}_\Xi = \pi_\Xi \cdot (t[\gamma\cdot\delta])$, and $\gamma_A \cdot \delta_{A[\gamma]} \cdot (t[\gamma][\delta]) = \gamma_A \cdot t[\gamma] \cdot \gamma = t \cdot \gamma \cdot \delta = (\gamma\cdot\delta)_A \cdot t[\gamma\cdot\delta]$. This is enough to show that substitution of terms as defined here is functorial.

To establish context comprehension, note first of all that for each $\gamma : \Delta \to \Gamma$ and type $\Gamma \vdash A$ there is a bijection $\mathbb{C}/\Gamma(\gamma, \pi_A) \cong \mathbb{C}/\Delta(\mathrm{id}_\Delta, \pi_{\Delta.A[\gamma]})$, since $\pi_{\Delta.A[\gamma]}$ is the pullback of π_A along γ. For a term $\Delta \vdash t : A[\gamma]$ write $\langle\gamma, t\rangle$ for the corresponding morphism $\Delta \to \Gamma.A$ and note that $\langle\gamma, t\rangle = \gamma_A \cdot t$ and clearly, by construction, $\pi_A \cdot \langle\gamma, t\rangle = \gamma$.

Finally we need to establish the role of the term $\Gamma.A \vdash \pi'_A : A[\pi_A]$. Start by defining $\pi'_A : \Gamma.A \to \Gamma.A.A[\pi_A]$ to be the unique map with $\pi_A \cdot \pi'_A = \mathrm{id}_{\Gamma.A} = (\pi_A)_A \cdot \pi'_A$. It remains to show that $\pi'_A[\langle\gamma, t\rangle] = t$, but as we've already seen, t is the unique morphism satisfying $\pi_{A[\gamma]} \cdot t = \mathrm{id}_\Delta$ and $\gamma_A \cdot t = \langle\gamma, t\rangle$, so it is enough to show $\gamma_A \cdot \pi'_A[\langle\gamma, t\rangle] = \langle\gamma, t\rangle$. First note that $\gamma_A = \gamma_A \cdot t \cdot \pi_{A[\gamma]} = \langle\gamma, t\rangle \cdot \pi_{A[\gamma]} = \pi_{A[\pi_A]} \cdot \langle\gamma, t\rangle_A$, and then calculate $\gamma_A \cdot \pi'_A[\langle\gamma, t\rangle] = \pi_{A[\pi_A]} \cdot \langle\gamma, t\rangle_A = \pi_{A[\pi_A]} \cdot \pi'_A\langle\gamma, t\rangle$. □

We will now start to use the internal language and the category of families associated with $\mathbb{C}$ interchangeably to reason about morphisms and types in $\mathbb{C}$. To start with, we will write $\Gamma \vdash A$ for $A \in \mathbb{C}/\Gamma$.

Conversely to the above result, we can construct a fibration from every category with families. In an important sense this fibration is *equivalent* to the original category with families, but it will take us too far afield to make this correspondence explicit, in particular we would first need to develop the theory of fibrations with comprehension.

Proposition 3.3.8 *Given a category with families $\mathscr{C}$ and category of contexts $\mathbb{C}$ a split fibration $\overline{\mathscr{C}} \to \mathbb{C}$ can be constructed with objects of $\overline{\mathscr{C}}$ the types of $\mathscr{C}$ and vertical morphisms terms of the form $\Gamma.A \vdash f : B[\pi_A]$ such that $\overline{\mathscr{C}}_\Gamma(A,B) \cong \mathbb{C}/\Gamma(\pi_A, \pi_B)$.*

Proof. Define the objects of each fibred $\overline{\mathscr{C}}_\Gamma$ to be the types $\Gamma \vdash A$ and morphisms $A \to B$ to be terms $\Gamma.A \vdash f : B^+$. The bijection $\overline{\mathscr{C}}_\Gamma(A,B) \cong \mathbb{C}/\Gamma(\pi_A, \pi_B)$ is through the maps

$$\begin{array}{ccc} \Gamma.A \vdash f : B^+ & \longmapsto & \Gamma.A \xrightarrow{\langle \pi_A, f \rangle} \Gamma.B \\ \pi_A \xrightarrow{f} \pi_B & \longmapsto & \Gamma.A \vdash \pi'_B[f] : B[\pi_B \cdot f] = B^+ \end{array} .$$

It is easy to see that defining $g \cdot f \equiv g[\langle \pi_A, f \rangle]$ respects composition in $\mathbb{C}$ and so each $\overline{\mathscr{C}}_\Gamma$ is a category. For $\gamma : \Delta \to \Gamma$ define $\gamma^* : \overline{\mathscr{C}}_\Gamma \to \overline{\mathscr{C}}_\Delta$ to take A to $A[\gamma]$ and t to $t[\gamma]$; this makes $\overline{\mathscr{C}}$ into a split indexed category. □

One further example is important: this is the category of families of "small types" or "types inside a universe". Given a family in $\mathbb{C}$, which we can call a "universe" and write $\mathscr{U} \vdash \mathrm{Elt}$, the families which arise as pullbacks along a map into $\mathscr{U}$ can be thought of as "small" types.

Construction 3.3.9 *Given a category with families $\mathscr{C}$ with category of contexts $\mathbb{C}$ and a designated type $\mathscr{U} \vdash \mathrm{Elt}$ in $\mathscr{C}$ then a new category of families $\mathscr{C}_\mathscr{U}$ with the same category of contexts, called the* category of families in the universe $\mathscr{U}$ *is defined by restricting the types of $\mathscr{C}_\mathscr{U}$ to types in $\mathscr{C}$ of the form $\Gamma \vdash \mathrm{Elt}[\gamma]$ for $\gamma : \Gamma \to \mathscr{U}$ in $\mathbb{C}$. Terms in $\mathscr{C}_\mathscr{U}$ are the same as terms in $\mathscr{C}$ where defined.*

For example, if $\mathbb{C}$ has a natural numbers object $\mathbb{N}$ then the morphism $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ taking (n,m) to $n+m+1$ can be regarded as a family, which we'll write $\mathbb{N} \vdash \mathrm{Fin}$. For $n : \mathbb{N}$ the object Fin_n can be regarded as the "finite cardinal with n elements" (Johnstone, 1977, §6.21) and Fin is the "object of finite cardinals in $\mathbb{C}$". Then given $\gamma : \Gamma \to \mathbb{N}$ a type $\Gamma \vdash A \equiv \gamma^*\mathrm{Fin}$ has the property that each fibre of A is a finite cardinal, and we can say that an object of this form is *discretely finite*.

We will see in section 4.7 that the shapely types of Jay (1995) are the containers constructed in the universe of finite cardinals.

Context Morphisms Revisited

Note that the internal language associated with a category with families only allows us to build contexts out of "small" types, so for example the internal language associated with the family of finite cardinals $\mathbb{N} \vdash \mathrm{Fin}$ only allows us to talk about finite contexts. As the categories with families are the basic framework in which we wish to work, this seems to be a weakness in the definition of the internal language.

A note on the treatment of contexts in the language: it seems to be usual (eg, Hofmann, 1994) to syntactically build up contexts from the empty context using only types in context in the language. However, this approach is too restrictive in three ways.

- In the context of this thesis we want to talk about functors of the form $\mathbb{C}^I \to \mathbb{C}$, and in order to do this we need to develop the internal language of $\mathbb{C}^I$. To ensure that the language is close to the interpretation the languages for $\mathbb{C}$ and $\mathbb{C}^I$ need to share a common category of contexts.

- The internal language has been used to construct categories with families. If we want to show an equivalence between the language and this semantic structure we need to extend the contexts of the language.

- In practice all of the constructions we describe using the language are in the context of an *arbitrary* context Γ, not necessarily a finite context built up inductively. The language should reflect this practice.

One solution is to extend the internal language so that there is a more explicit distinction between "small" types, which can appear either side of a type judgement, and "large" types which can only appear in the context. To do this explicitly, we would need to revisit the language in detail; one approach might begin as follows.

A category $\mathbb{C}_0$ of *basic contexts* is given. The rules for creating contexts and context morphisms in section 2.2 are replaced by the following three rules:

$$\frac{\Gamma \in \mathbb{C}_0}{\Gamma \vdash} \qquad \frac{\gamma : \Delta \longrightarrow \Gamma \text{ in } \mathbb{C}_0}{\underline{\gamma} : \Delta \longrightarrow \Gamma} \qquad \frac{\Gamma \vdash A}{\pi_A : \Gamma, A \longrightarrow \Gamma} \ .$$

The introduction of a new special rule to form π_A is required to accommodate the fact that we can no longer inductively construct π_A as described in section 2.3. If contexts and context morphisms are constructed inductively as before, the following rules for computing context composition suffice:

$$\underline{\gamma} \cdot \underline{\delta} \equiv \underline{\gamma \cdot \delta} \qquad (\gamma, t) \cdot \delta \equiv (\gamma \cdot \delta, t[\delta]) \qquad \pi_A \cdot (\gamma, t) \equiv \gamma \ .$$

and the analysis in section 2.3 continues to go through.

However, this thesis is not about syntactic structure, and here is not the place to address any of the issues arising from the discussion above. Instead, we will take the category of families as our implicit internal language, except that we will use variables and implicit weakening as provided by the original internal language.

3.4 Types in Fibrations

All the types of the basic language have their expression in the more general fibred framework. We will develop most of the theory simultaneously in a general fibration and in the context of a category with families, but we'll focus on the the structure specific to a locally cartesian closed category $\mathbb{C}$.

Proposition 3.4.1 *If $\mathbb{C}$ is closed under pullbacks then for each $\gamma : \Delta \to \Gamma$ in $\mathbb{C}$ the reindexing functor $\gamma^* : \mathbb{C}/\Gamma \to \mathbb{C}/\Delta$ has a left adjoint $\sum_\gamma \dashv \gamma^*$. When $\gamma = \pi_X$ is a display map, write $\sum_X \equiv \sum_{\pi_X}$ (indeed, every map can be regarded as a display map).*

These adjoint functors satisfy the Beck-Chevalley *condition: for each $X \in \mathbb{C}/\Gamma$ and $\gamma : \Delta \to \Gamma$ the canonical morphism $\sum_{\gamma^* X} \gamma_X^* \to \gamma^* \sum_X$ is iso; also a* strength *condition is satisfied: the canonical morphism $\pi_A \cdot \pi_B \to \pi_{\sum_A B}$ in $\mathbb{C}/\Gamma$ is iso.*

Proof. For $f : X \to \Delta$ in $\mathbb{C}/\Delta$ define $\sum_\gamma f \equiv \gamma \cdot f \in \mathbb{C}/\Gamma$. All the properties of the proposition are a routine verification of properties of pullbacks.

The "canonical" morphism $\sum_{\gamma^* X} \gamma_X^* \to \gamma^* \sum_X$ referred to in the proposition arises as the transpose of

$$\gamma_X^* \xrightarrow{\gamma_X^* \eta^{\sum_X}} \gamma_X^* \pi_X^* \sum_X \cong \pi_{\gamma^* X} \gamma^* \sum_X \quad ,$$

and similarly $\pi_A \cdot \pi_B \to \pi_{\sum_A B}$ arises from the term $\Gamma, a : A, b : B \vdash (a, b) : \sum_A B$. $\square$

These left adjoints to substitution correspond directly to Sigma types in the language, the Beck-Chevalley condition corresponds to the preservation of $\sum$ by substitution, and strength allows us to treat the contexts Γ, A, B and $\Gamma, \sum_A B$ interchangeably.

Definition 3.4.2 *A category with families* has Sigma types *iff there is a construction*

$$\frac{\Gamma \vdash A \quad \Gamma.A \vdash B}{\Gamma \vdash \sum_A B}$$

with equations $\pi_{\sum_A B} = \pi_A \cdot \pi_B$ and $(\sum_A B)[\gamma] = \sum_{A[\gamma]} B[\gamma_A]$ for each $\gamma : \Delta \to \Gamma$.

In fact, the equation $\pi_{\sum_A B} = \pi_A \cdot \pi_B$ is stronger than strictly necessary, but as the proposition below shows this doesn't matter.

Proposition 3.4.3 *The category with families for $\mathbb{C}$ has Sigma types.*

Proof. Given $A : \mathrm{dom}_\Gamma \to \mathrm{cod}_\mathbb{C}$ and $B : \mathrm{dom}_{\Gamma.A} \to \mathrm{cod}_\mathbb{C}$ define $\sum_A B : \mathrm{dom}_\Gamma \to \mathrm{cod}_\mathbb{C}$ by defining $(\sum_A B)\gamma \equiv \sum_{A\gamma} B\gamma_A$ in $\mathbb{C}$ for each $\gamma : \Delta \to \Gamma$. The equations for Sigma types are automatically satisfied by this construction. □

To relate this to the Sigma types in the language define

$$\Gamma, \textstyle\sum_A B \vdash \pi \equiv \pi'_A[\pi_A] : A[\pi_{\sum_A B}] \qquad \Gamma, \textstyle\sum_A B \vdash \pi' \equiv \pi'_B : B[\langle \pi_{\sum_A B}, \pi \rangle]$$

and observe that $A[\pi_{\sum_A B}]$ is just A weakened and similarly $B[\langle \pi_{\sum_A B}, \pi\rangle]$ is in fact just B with π substituted for A; we'll write this more conveniently as $\pi^* B$.

One useful consequence of this result is that in the codomain fibration products in a fibre can be described using the $\sum$ construction: given $X, Y \in \mathbb{C}/\Gamma$ then

$$X \times Y \cong \textstyle\sum_X \pi_X^* Y \cong \sum_Y \pi_Y^* X \quad .$$

Another important construction which is effectively lifting a familiar categorical construction into the fibred framework is the "equality type". Just as the equaliser $\mathrm{Eq}(f,g)$ of parallel pairs $f, g : X \rightrightarrows Y$ can be constructed as a pullback of the diagonal $\delta_Y \equiv (\mathrm{id}_Y, \mathrm{id}_Y)$ thus

$$\begin{tikzcd} \mathrm{Eq}(f,g) \arrow[r, tail] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] \arrow[d, "{(f,g)}"] & Y \\ Y \arrow[r, tail, "\delta_Y"'] & Y \times Y & \end{tikzcd}$$

then similarly we can regard the type $X \vdash \mathrm{Eq}(f,g)$ (which is, incidentally, a subobject of 1 in $\mathbb{C}/X$, and thus takes the role of a proposition) as a substitution instance $(f,g)^*\mathrm{Eq}$ of the "equality type" $Y \vdash \mathrm{Eq}_Y$.

Definition 3.4.4 *A category with families* has equality types *when for each type $\Gamma \vdash A$ there exist type and term*

$$\Gamma.A.A^+ \vdash \mathrm{Eq}_A \qquad\qquad \Gamma.A \vdash \mathrm{refl}_A : \mathrm{Eq}_A[\langle \mathrm{id}_{\Gamma.A}, \pi'_A \rangle]$$

together with a deduction rule

$$\frac{\Gamma \vdash t:A \quad \Gamma \vdash u:A \quad \Gamma \vdash e:\mathrm{Eq}(t,u)}{\Gamma \vdash t = u:A} \ ,$$

writing $\mathrm{Eq}(t,u) \equiv \mathrm{Eq}[\langle\langle \mathrm{id}_\Gamma, t\rangle, u\rangle]$, *such that* $\mathrm{Eq}_{A[\gamma]} = \mathrm{Eq}_A[\gamma_{A,A^+}]$, *or equivalently,* $\mathrm{Eq}(t,u)[\gamma] = \mathrm{Eq}(t[\gamma], u[\gamma])$*; this is the Beck-Chevalley condition.*

Of course, the map $\langle\langle \mathrm{id}_\Gamma, t\rangle, u\rangle : \Gamma \to \Gamma.A.A^+$ is the same thing as a term $(t,u) : A \times A$ over Γ, and so we can write $\mathrm{Eq}(t,u) = (t,u)^* \,\mathrm{Eq}_A$. As one might expect, we have equality types for $\mathbb{C}$.

Proposition 3.4.5 *The category of families for* $\mathbb{C}$ *has equality types.*

Proof. Clearly the diagonal $\delta_A \equiv \langle \mathrm{id}_A, \pi'_A\rangle : \Gamma.A \to \Gamma.A.A^+$ is a suitable display map for the type Eq_A, but the problem is to construct $\mathrm{Eq}_A : \mathrm{dom}_{\Gamma.A.A^+} \to \mathrm{cod}_{\mathbb{C}}$ in such a way that Beck-Chevalley is satisfied.

Observe that any map $\gamma : \Delta \to \Gamma.A.A^+$ can be uniquely decomposed into components $\gamma = \langle\langle \gamma', t\rangle, u\rangle$ for $\gamma' = \pi_A \cdot \pi_{A^+} \cdot \gamma$, $t = \pi'_A[\pi_{A^+} \cdot \gamma]$ and $u = \pi'_{A^+}[\gamma]$. Now for each context Δ and each pair of terms $\Delta \vdash t : A$ and $\Delta \vdash u : A$ let $\mathrm{eq}_\Delta(t,u)$ be chosen making the following square a pullback

$$\begin{tikzcd} \bullet \arrow[r] \arrow[d, "\mathrm{eq}_\Delta(t,u)"'] \arrow[dr, phantom, "\lrcorner", very near start] & \Delta.A = \Delta.A.A^+.\mathrm{Eq}_A \arrow[d, "\langle \mathrm{id}_A, \pi'_A\rangle = \pi_{\mathrm{Eq}_A}"] \\ \Delta \arrow[r, "(t,u)"'] & \Delta.A.A^+ \end{tikzcd} \ ,$$

and define $\mathrm{Eq}_A\langle\langle \gamma, t\rangle, u\rangle \equiv \mathrm{eq}_\Delta(t,u)$. This is a valid type in the category with families for $\mathbb{C}$, and the Beck-Chevalley condition is easy to verify: $\mathrm{Eq}_A[\gamma_{A,A^+}]\langle\langle \delta, t\rangle, u\rangle = \mathrm{Eq}_A(\gamma_{A,A^+} \cdot \langle\langle \delta, t\rangle, u\rangle) = \mathrm{Eq}_A\langle\langle \gamma \cdot \delta, t\rangle, u\rangle = \mathrm{eq}_\Xi(t,u) = \mathrm{Eq}_{A[\gamma]}\langle\langle \delta, t\rangle, u\rangle$.

The term $\mathrm{refl}_A : \mathrm{id}_{\Gamma.A} \to \mathrm{eq}_{\Gamma.A}(\pi'_A, \pi'_A)$ is obtained by factorisation of $(\mathrm{id}_{\Gamma.A}, \mathrm{id}_{\Gamma.A})$ through the pullback defining eq.

Any term $\Gamma \vdash e : \mathrm{Eq}(t,u)$ implies the existence of a morphism $f : \Gamma \to \Gamma.A$ with $\langle \mathrm{id}_A, \pi'_A\rangle \cdot f = (u,v)$; this implies $t = u$. ☐

The existence of Sigma types and equality types together has some quite important consequences, perhaps the most important of which is that every term $\Gamma.A \vdash f : B^+$ can be interpreted as a type by defining the type

$$\Gamma,\, b:B \ \vdash\ A \restriction f \equiv \sum a:A.\ \mathrm{Eq}(fa, b) \ ,$$

and note that $\pi_{A\restriction f} = f$.

The appropriate notion of structure in a fibration is that it be defined in each fibre and are preserved by reindexing, so we have the following definition of fibred exponentials in a fibration.

Definition 3.4.6 *Say that a fibration with fibred binary products is* fibred cartesian closed *iff each fibre is cartesian closed and the exponential structure is preserved by reindexing, ie* $\gamma^*(A^B) \cong (\gamma^* A)^{\gamma^* B}$.

Recall now that $\mathbb{C}$ is *locally cartesian closed* iff each slice category $\mathbb{C}/\Gamma$ is cartesian closed. We have the following equivalent formulations of this property.

Proposition 3.4.7 *If $\mathbb{C}$ is pullback complete then the following are equivalent:*

1. *the codomain fibration* $\mathrm{cod}_{\mathbb{C}} : \mathbb{C}^{\to} \to \mathbb{C}$ *is fibred cartesian closed;*
2. $\mathbb{C}$ *is locally cartesian closed;*
3. *each reindexing functor γ^* has a right adjoint $\gamma^* \dashv \prod_\gamma$ satisfying the Beck-Chevalley condition:* $\gamma^* \prod_A \cong \prod_{\gamma^* A} \gamma_A^*$.

Proof. **($1 \Longrightarrow 2$)** This is immediate by definition.

($2 \Longrightarrow 3$) For $\Gamma.A \vdash B$ define $\Gamma \vdash \prod_A B \equiv \sum f : (\sum_A B)^A . \mathrm{Eq}(\pi \cdot f, \mathrm{id}_A)$ then

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash (f,e) : U \longrightarrow \sum f : (\sum_A B)^A.\ \mathrm{Eq}(\pi \cdot f, \mathrm{id}_A)}{\Gamma \vdash f : U \longrightarrow (\sum_A B)^A \qquad \Gamma, U \vdash e : \mathrm{Eq}(\pi \cdot f, \mathrm{id}_A)}}{\Gamma \vdash f_1 : A \times U \longrightarrow A \qquad \Gamma, A \times U \vdash f_2 : f_1^* B \qquad f_1 = \pi}}{\Gamma, A, \pi_A^* U \vdash f_2 : \pi_A^* B}}{\Gamma, A \vdash \pi_A^* U \longrightarrow B}$$

showing that $\pi_A^* \dashv \prod_A$. To verify the Beck-Chevalley condition recall that $\pi_A^* \Sigma_\gamma \cong \Sigma_{\gamma_A} \pi_{A[\gamma]}^*$ and compute: $\mathbb{C}/\Delta(U, \gamma^* \prod_A B) \cong \mathbb{C}/\Gamma.A(\pi_A^* \Sigma_\gamma U, B) \cong \mathbb{C}/\Gamma.A(\Sigma_{\gamma_A} \pi_{A[\gamma]}^* U, B) \cong \mathbb{C}/\Delta(U, \prod_{A[\gamma]} \gamma_A^* B)$.

($3 \Longrightarrow 1$) For $X, Y \in \mathbb{C}/\Gamma$ define $X^Y \equiv \prod_Y \pi_Y^* X$ and verify $\mathbb{C}/\Gamma(U, \prod_Y \pi_Y^* X) \cong \mathbb{C}/\Gamma.Y(\pi_Y^* U, \pi_Y^* X) \cong \mathbb{C}/\Gamma(X \times U \cong \Sigma_Y \pi_Y^* U, X)$ showing that each $\mathbb{C}/\Gamma$ is cartesian closed. Preservation of exponentials follows immediately from Beck-Chevalley. $\square$

As for Sigma types, we can define Pi types in the category with families for a locally cartesian closed $\mathbb{C}$.

Proposition 3.4.8 *If $\mathbb{C}$ is locally cartesian closed then for types $\Gamma.A \vdash B$ there exists a type $\Gamma \vdash \prod_A B$ forming a right adjoint to substitution and satisfying the Beck-Chevalley condition $(\prod_A B)[\gamma] = \prod_{A[\gamma]} B[\gamma_A]$.*

Proof. As for Sigma types, define $(\prod_A B)\gamma \equiv \prod_{A\gamma} B\gamma_A$. □

Limits and Colimits in Fibrations

By a $\mathbb{J}$-limit in $\mathbb{D}$ is meant the limit of a diagram of the form $D : \mathbb{J} \to \mathbb{D}$, which we'll write in any of the forms $\varprojlim D = \varprojlim_{\mathbb{J}} D = \varprojlim_{j\in\mathbb{J}} Dj \in \mathbb{D}$. The limiting cone is written $(\pi_j : \varprojlim D \to Dj)_{j\in\mathbb{J}}$ and each cone $\vec{\alpha} = (\alpha_j : X \to D_j)_{j\in\mathbb{J}}$ uniquely factors through $\langle\alpha_j\rangle_{j\in\mathbb{J}} : X \to \varprojlim D$ satisfying the equation $\pi_j \cdot \langle\vec{\alpha}\rangle = \alpha_j$. Similarly the colimit of a diagram D is written $\varinjlim_{\mathbb{J}} D$ with colimiting cone $(\kappa_j : Dj \to \varinjlim D)_{j\in\mathbb{J}}$ and unique factorisation $[\beta_j]_{j\in\mathbb{J}}$ satisfying $[\vec{\beta}] \cdot \kappa_j = \beta_j$.

The natural notion of structure in a fibration is structure that is present in each fibre and which is preserved by reindexing. Thus we have the following definition of fibred limit and colimit.

Definition 3.4.9 *Say that a fibration $p : \mathbb{E} \to \mathbb{C}$ has* fibred $\mathbb{J}$-limits *iff each fibre has $\mathbb{J}$-limits which are preserved by reindexing functors. Dually, say that p has* fibred $\mathbb{J}$-colimits *iff each fibre has $\mathbb{J}$-colimits preserved by reindexing.*

Conveniently it turns out that limits in $\mathbb{C}$ automatically lift to fibred limits in the codomain fibration.

Proposition 3.4.10 *If $\mathbb{C}$ has pullbacks and $\mathbb{J}$-limits then* $\mathrm{cod}_{\mathbb{C}}$ *has fibred $\mathbb{J}$-limits.*

Proof. Limits in $\mathbb{C}$ lift to limits in each slice $\mathbb{C}/\Gamma$: given $D : \mathbb{D} \to \mathbb{C}/\Gamma$ construct the morphism $\langle\pi_{Dj} \cdot \pi_j\rangle_j : \varprojlim_{\mathbb{J}} \Gamma.D \to \varprojlim_{\mathbb{J}} \Gamma$ and pull it back along $\delta_{\mathbb{J}} \equiv \langle\mathrm{id}_\Gamma\rangle_j : \Gamma \to \varprojlim_{\mathbb{J}} \Gamma$

to yield an object of $\mathbb{C}/\Gamma$. Now let any $X \in \mathbb{C}/\Gamma$ be given and consider:

$$\begin{tikzcd}
\Gamma.X \arrow[rr, dashed, "\langle\vec{f}\rangle"] \arrow[dr, dashed, "f_j"'] \arrow[dd, "\pi_X"'] & & \varprojlim_{j\in\mathbb{J}} \Gamma.Dj \arrow[dl, "\pi_j"] \arrow[dd, "\langle \pi_{Dj}\cdot\pi_j\rangle_j"] \\
 & \Gamma.Dj \arrow[dd, "\pi_{Dj}"] & \\
\Gamma \arrow[rr, "\delta_{\mathbb{J}}"] \arrow[dr, "\mathrm{id}_\Gamma"'] & & \varprojlim_{j\in\mathbb{J}} \Gamma \arrow[dl, "\pi_j"] \\
 & \Gamma &
\end{tikzcd} .$$

Any cone $(f_j : X \to Dj)_j$ in $\mathbb{C}/\Gamma$ uniquely factors through $\langle\vec{f}\rangle$ as shown, and furthermore the main square commutes and so $\langle\vec{f}\rangle$ factors through the pullback along $\delta_{\mathbb{J}}$ constructed above, showing that it is the limit in $\mathbb{C}/\Gamma$ of D.

That limits are preserved by reindexing is immediately obvious since reindexing in this fibration is by taking pullbacks. $\square$

Similarly, if $\mathbb{C}$ is locally cartesian closed then we get the same result for colimits.

Proposition 3.4.11 *If $\mathbb{C}$ is locally cartesian closed and has $\mathbb{J}$-colimits then* $\mathrm{cod}_{\mathbb{C}}$ *has fibred $\mathbb{J}$-colimits.*

Proof. Given a diagram $D : \mathbb{J} \to \mathbb{C}/\Gamma$ compute $[\pi_{Dj}]_{j\in\mathbb{J}} : \varinjlim_{j\in\mathbb{J}} \Gamma.Dj \to \Gamma$ as an object of $\mathbb{C}/\Gamma$. This is the required colimit diagram, since any cone $(f_j : Dj \to X)_j$ factors as $[\vec{f}] : \varinjlim \Gamma.D \to \Gamma.X$ necessarily satisfying $\pi_X \cdot [\vec{f}] = [\pi_D] = \pi_{\varinjlim \Gamma.D}$. These colimits are preserved by pullback functors since each γ^* has a right adjoint. $\square$

Chapter 4

Categories of Containers

Here we introduce the theory of containers as representatives for a class of functors of the form

$$T_{A\triangleright B}X \equiv \sum a:A.\ \prod i:I.\ X_i^{B_i(a)} \qquad \mathbb{C}^I \xrightarrow{T_{A\triangleright B}} \mathbb{C} \quad .$$

4.1 Introducing Containers

The basic notion of a *container* is a dependent pair of types $A \vdash B$, written $(A \triangleright B)$, yielding a functor $T_{A\triangleright B}X \equiv \sum a:A.X^{B(a)}$, and it turns out that the appropriate notion of a morphism $(A \vdash B) \to (C \vdash D)$ is a pair of morphisms $(u : A \to C, f : u^*D \to B)$. With this definition of a category $\mathscr{G}$ of container generators we can construct a full and faithful functor $T : \mathscr{G} \to [\mathbb{C}, \mathbb{C}]$ and show the completeness properties discussed in the introduction. We refer to each functor of the form $T_{A\triangleright B}$ as a "container functor" or the "extension" of the container $(A \triangleright B)$.

An important part of the theory is the extension of this definition to a container in I parameters, for any set I, generating a category $\mathscr{G}_I$ of I-indexed containers with extension given by $T : \mathscr{G}_I \to [\mathbb{C}^I, \mathbb{C}]$. These containers with parameters compose to form a bicategory and play a key role in the development of the theory of fixed points, which can be thought of as being generated by iterated composition of containers.

There are two different ways of thinking of how a container functor $F(X,Y)$ in two parameters (ie, an object of $\mathscr{G}_2$) might arise. One approach is to think of $F(X,Y)$ as being an X-indexed family of containers $(A(X) \triangleright B(X))$. We therefore start with the following observation about "containers with parameters".

Proposition 4.1.1 *The following constructions of a functor $F : \mathbb{C}^n \to \mathbb{C}$ are equivalent:*

1. *If $n = 0$: an object of $\mathbb{C}$ equal to F.*

   If $n = m+1$: a container functor $S : \mathbb{C}^m \to \mathbb{C}$ together with an object $S\vec{X} \vdash P\vec{X}$ for each $\vec{X} \in \mathbb{C}^m$ satisfying the isomorphism $P\vec{Y}(S\vec{f}(s)) \cong P\vec{X}(s)$ for $\vec{f} : \vec{X} \to \vec{Y}$ and $s : S\vec{X}$. Define $F(\vec{X}, Y)$ equal to $\sum s : S\vec{X}. Y^{P\vec{X}(s)}$.

   The families $(S\vec{X} \triangleright P\vec{X}) = (S \triangleright P)\vec{X}$ should be thought of as a $\vec{X}$-indexed family of containers, so here we can think of a container in $m+1$ parameters as a m-indexed family of containers in one parameter.

2. *An object $A \in \mathbb{C}$ together with n objects over A, i.e. $A \vdash (B_i)_{i \in 1..n}$, with $F\vec{X}$ defined equal to $\sum a : A. \prod_{i \in 1..n} X_i^{B_i(a)}$.*

Proof. By induction. For $n = 0$ the two cases are clearly equivalent, so let $n = m+1$ and let $S\vec{X} = \sum a : A. \prod_{i \in 1..m} X_i^{B_i(a)}$. Observe now that $P\vec{X}$ is fully determined by $P1$, since $P\vec{X} \cong P1(S!_{\vec{X}})$, so take $B_n \equiv P1$. In particular, for $(a, f) : S\vec{X}$ note that $P\vec{X}(a, f) \cong B_n(a)$.

We can now show that the two definitions of F are equivalent:

$$\begin{aligned} F_1(\vec{X}, Y) = \sum s : S\vec{X}.\ Y^{P\vec{X}(s)} &\cong \sum a : A.\ \sum f : \textstyle\prod_{i \in 1..m} X_i^{B_i(a)}.\ Y^{P\vec{X}(a,f)} \\ &\cong \sum a : A.\ \sum f : \textstyle\prod_{i \in 1..m} X_i^{B_i(a)}.\ Y^{B_n(a)} \cong F_2(\vec{X}, Y)\ . \end{aligned}$$ □

The second of the constructions above is technically simpler and generalises more easily and we therefore take this to be our definition of a container in n parameters. For the purposes of this paper the index set n or I will generally be finite, but in fact it makes little difference.

Indeed, it is straightforward to generalise the development in this thesis to the case where containers are parameterised by *internal* index objects $I \in \mathbb{C}$; when $\mathbb{C}$ has enough coproducts nothing is lost by doing this, since $\mathbb{C}^I \simeq \mathbb{C}/\sum_{i \in I} 1$. This generalisation will be important for the development of "dependent containers"; see section 7.2.

4.2 Defining Containers

Containers are defined to capture a particular class of functor of the form $\mathbb{C}^I \to \mathbb{C}^J$; however, as $[\mathbb{C}^I, \mathbb{C}^J] \cong [\mathbb{C}^I, \mathbb{C}]^J$ it is sufficient to concentrate on the functors $\mathbb{C}^I \to \mathbb{C}$.

In this section we will define for each indexing set I the category $\mathscr{G}_I$ of "I-indexed containers" together with the extension of each container as a functor.

Definition 4.2.1 *Given a locally cartesian closed category $\mathbb{C}$ with I-indexed products (for I a set) define an I-*indexed container *to be a pair $A \in \mathbb{C}$, $B \in (\mathbb{C}/A)^I$. Such a container will be written as $(A \triangleright B)$ and the set of I-indexed containers is written $\mathscr{G}_I$.*

In the internal language of $\mathbb{C}$ we can write $(A \triangleright B)$ as

$$\vdash A \quad \text{in } \mathbb{C} \qquad a:A \vdash B(a) \quad \text{in } \mathbb{C}^I$$

Note that the way we have developed the internal language allows us to hide the I-indexing of B much of the time. When this indexing needs to be made explicit, we will write this as $i:I, a:A \vdash B_i(a)$ in $\mathbb{C}$, and the associated container may be written as

$$\big(a:A \;\triangleright\; (B_i(a))_{i\in I}\big) \qquad \text{or} \qquad \big(a:A \;\triangleright_{i:I}\; B_i(a)\big) \;\; .$$

Definition 4.2.2 *Define the* extension *of a container $(A \triangleright B) \in \mathscr{G}_I$ to be the functor $T_{A\triangleright B} : \mathbb{C}^I \to \mathbb{C}$ defined on objects $X \in \mathbb{C}^I$ by the following expression:*

$$T_{A\triangleright B}X \equiv \sum a:A.\ \prod i:I.\ X_i^{B_i(a)}$$

and on morphisms $g : X \to Y$ by

$$(T_{A\triangleright B}\,g)(a,f) \equiv (a, g\cdot f) \;\; .$$

A functor isomorphic to some $T_{A\triangleright B}$ is called a container functor.

It is clear that the above expression defines a fibred functor, but it will be helpful to look more closely at how an element of $T_{A\triangleright B}X$ in a arbitrary context $\Gamma \vdash X$ can be interpreted. We can use the rules of the internal language

$$\dfrac{\dfrac{\Gamma \vdash t : T_{A\triangleright B}X = \sum a:A.\ \prod i:I.\ X^{B_i(a)} \quad \text{in } \mathbb{C}}{\Gamma \vdash a:A \qquad \Gamma \vdash f : \prod i:I.\ X^{B_i(a)} \quad \text{in } \mathbb{C}}}{\Gamma \vdash a:A \quad \text{in } \mathbb{C} \qquad \Gamma \vdash f : B(a) \longrightarrow X \quad \text{in } \mathbb{C}^I}$$

to decompose an element of $T_{A\triangleright B}X$ into the two elements a and f as shown; we will therefore write $(a,f) : T_{A\triangleright B}X$ whenever appropriate. To be precise, $f : B(a) \to I^*X$ where I^*X is the constant family $(X)_{i\in I} \in \mathbb{C}^I$; however, such weakenings will seldom need to be made explicit.

To be precise, when we write $g \cdot f$ in definition 4.2.2, we literally mean $I^* g \cdot f$ for $I^* g : I^* X \to I^* Y$; again, such weakenings will frequently be elided.

We extend $\mathcal{G}_I$ to a category by the following definition of morphism. This definition ensures that morphisms between containers capture precisely natural transformations between container extensions, as shown by theorem 4.3.3.

Definition 4.2.3 *A* container morphism $(A \triangleright B) \to (C \triangleright D)$ *is given by a pair of morphisms $u : A \to C$ in $\mathbb{C}$ and $f : u^* D \to B$ in $(\mathbb{C}/A)^I$. The category of containers and container morphisms is written $\mathcal{G}_I$.*

The composite of two container morphisms is defined as

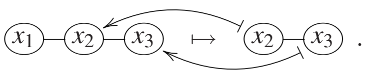
$$(v, g) \cdot (u, f) \equiv (v \cdot u,\ f \cdot u^* g) \ .$$

A morphism $(u, f) : (A \triangleright B) \to (C \triangleright D)$ can be written in type theoretic notation as

$$u : A \longrightarrow C \qquad i : I, a : A \ \vdash \ f_i(a) : D_i(ua) \longrightarrow B_i(a) \ .$$

This definition of container morphism can be understood with the help of an example. Consider the map $\mathsf{tail} : \mathrm{List} X \to 1 + \mathrm{List} X$ taking the empty list to $* \in 1$ and otherwise yielding the tail of the given list:

tail of list



This map is defined by i) a choice of shape in $1 + \mathrm{List} X$ for each shape in $\mathrm{List} X$, ie $0 \mapsto *$, $n + 1 \mapsto n$; and ii) for each position in the chosen shape a position in the original shape, ie the function $i \mapsto i + 1$.

We can now extend the construction of the extension of a container to a functor $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$.

Definition 4.2.4 *T extends to a functor $T : \mathcal{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ as follows. For a container morphism $(u, f) : (A \triangleright B) \to (C \triangleright D)$ define $T_{u,f} : T_{A \triangleright B} \to T_{C \triangleright D}$ to be the natural transformation $T_{u,f} X : T_{A \triangleright B} X \to T_{C \triangleright D} X$ thus:*

$$(a, g) : T_{A \triangleright B} X \ \vdash \ (T_{u,f} X)(a, g) \equiv (ua, g \cdot f) \ .$$

The following proposition follows more or less immediately by the construction of T.

Proposition 4.2.5 *For each container $F \in \mathcal{G}_I$ and each container morphism $\alpha : F \to G$ the functor T_F and natural transformation T_α are fibred over $\mathbb{C}$.* $\square$

4.3 A Yoneda Lemma for Containers

Just as the Yoneda lemma tells us that elements of a presheaf $F \in \mathbf{Set}^{\mathbb{C}^{\mathrm{op}}}$ are in bijection with morphisms in $\mathbf{Set}^{\mathbb{C}^{\mathrm{op}}}$ from representables to F, so we have a bijection between local elements of a fibred functor and morphisms from container functors. Thus we see container functors playing a role analogous to the representable functors. Note however that the bijection involves an essential change in base, so the analogy is not exact.

Proposition 4.3.1 *For each fibred functor $F : \mathbb{C}^I \to \mathbb{C}$ and each I-indexed container $(A \triangleright B)$ in $\mathscr{G}_I$ there is a bijection natural in F and $(A \triangleright B)$:*

$$\frac{T_{A\triangleright B} \longrightarrow F \qquad \text{in } [\mathbb{C}^I, \mathbb{C}]}{1 \longrightarrow F_A B \qquad \text{in } (\mathbb{C}/A)^I} \; .$$

Proof. We will conduct the proof entirely in the internal language of $\mathbb{C}$. In this context an arrow $x : 1 \to F_A B$ in $(\mathbb{C}/A)^I$ is regarded as a term $a : A \vdash x(a) : (FB)(a)$, and the localisation becomes almost invisible.

Given a natural transformation $\alpha : T_{A\triangleright B} \to F$ define $x_\alpha : 1 \to F_A B$ by the term

$$a : A \;\vdash\; x_\alpha(a) \equiv \alpha B(a) \cdot (a, \mathrm{id}_{B(a)}) \; .$$

Conversely, given $x : 1 \to F_A B$ define a fibred natural transformation $\alpha_x : T_{A\triangleright B} \to F$ with component $(\alpha_x)_\Gamma X : T_{A\triangleright B} X \to F_\Gamma X$ defined by the term

$$\Gamma,\, (a, f) : T_{A\triangleright B} X \;\vdash\; (\alpha_x)_\Gamma X \cdot (a, f) \equiv F f \cdot x(a) \; .$$

Verify that each $(\alpha_x)_\Gamma$ is a natural transformation by calculating for each $g : X \to Y$ in $(\mathbb{C}/\Gamma)^I$:

$$\begin{aligned}(\alpha_x)_\Gamma Y \cdot T_{A\triangleright B} g \cdot (a, f) = (\alpha_x)_\Gamma Y \cdot (a, g \cdot f) = F(g \cdot f) \cdot x(a) = F g \cdot F f \cdot x(a) \\ = F g \cdot (\alpha_x)_\Gamma X \cdot (a, f) \; .\end{aligned}$$

To see that α_x is fibred observe that the definition of $(\alpha_x)_\Gamma X$ involves variables a,f and the constants F and x which are all preserved by substitution.

Now we need to verify that these assignments are bijective. Given x it is easy to calculate (for $a : A$) that $x_{\alpha_x}(a) = (\alpha_x) B(a) \cdot (a, \mathrm{id}_{B(a)}) = F\,\mathrm{id}_{B(a)} \cdot x(a) = x(a)$.

Conversely, let α be given and calculate

$$
\begin{aligned}
(\alpha_{x_\alpha})_\Gamma X \cdot (a, f) &= Ff \cdot x_\alpha(a) = Ff \cdot \alpha B \cdot (a, \mathrm{id}_{B(a)}) = \alpha X \cdot T_{A \triangleright B} f \cdot (a, \mathrm{id}_{B(a)}) \\
&= \alpha X \cdot (a, f)
\end{aligned}
$$

showing that $\alpha_{x_\alpha} = \alpha$ and establishing the bijection.

To show naturality in F let $\beta : F \to G$ be a fibred natural transformation. It is now sufficient to verify that $x_{\beta \cdot \alpha}(a) = (\beta \cdot \alpha)B(a) \cdot (a, \mathrm{id}_{B(a)}) = \beta B(a) \cdot x_\alpha(a)$, showing that $x_{\beta \cdot \alpha} = \beta_A B \cdot x_\alpha$.

Naturality in $(A \triangleright B)$ is a little more delicate as a change of fibre is involved: given a container morphism $(u, f) : (C \triangleright D) \to (A \triangleright B)$ this induces a correspondence between

$$
\begin{tikzcd}
T_{A \triangleright B} \arrow[r, "\alpha"] & F \\
T_{C \triangleright D} \arrow[u, "T_{u,f}"] \arrow[ur, "\alpha \cdot T_{u,f}"'] &
\end{tikzcd}
\quad \text{and} \quad
\begin{tikzcd}
1 \arrow[rr, "u^* x_\alpha"] \arrow[dr, "x_{\alpha \cdot T_{u,f}}"'] & & u^* F_A B \cong F_C u^* B \arrow[dl, "F_C f"] \\
& F_C D &
\end{tikzcd}
$$

where $c : C \vdash F_C f \cdot u^* x_\alpha = F_C f \cdot \alpha B(uc) \cdot (uc, \mathrm{id}_{B(uc)}) = \alpha D(c) \cdot (T_{A \triangleright B})_C f \cdot (uc, \mathrm{id}_{B(uc)}) = \alpha D(c) \cdot T_{u,f} D(a) \cdot (c, \mathrm{id}_{D(c)}) = x_{\alpha \cdot T_{u,f}}$ (ignoring the coherence isomorphism by grace of the internal language) showing naturality as required. $\square$

The following lemma gives a bit more depth to the idea that container functors are a kind of representable functor.

Lemma 4.3.2 *There is a bijection between local elements of a container functor and morphisms in $\mathscr{G}_I$ thus:*

$$
\frac{1 \longrightarrow (T_{C \triangleright D})_A B \qquad \text{in } \mathbb{C}/A}{(A \triangleright B) \longrightarrow (C \triangleright D) \qquad \text{in } \mathscr{G}_I} \; .
$$

Proof. This is a matter of direct observation in the internal language. An element of $(T_{C \triangleright D})_A B$ can be written as

$$
a : A \vdash u(a) : C, \quad f_a : D(ua) \longrightarrow B(a)
$$

which is precisely a pair $u : A \to C$, $f : u^* D \to B$, ie a morphism in $\mathscr{G}_I$. $\square$

The following theorem is an immediate corollary and plays a key role in the development of the theory of containers in this thesis: we will frequently use reflection along T.

Theorem 4.3.3 *The functor $T : \mathscr{G}_I \to [\mathbb{C}^I, \mathbb{C}]$ is full and faithful.* $\square$

This result gives a particularly simple analysis of polymorphic functions between container functors. For example, it is easy to observe that there are precisely n^m polymorphic functions $X^n \to X^m$: the data type X^n is the container $(1 \triangleright n)$ and hence there is a bijection between polymorphic functions $X^n \to X^m$ and functions $m \to n$.

Similarly, any polymorphic function $\mathrm{List}\, X \to \mathrm{List}\, X$ can be uniquely written as a function $u : \mathbb{N} \to \mathbb{N}$ together with for each natural number $n : \mathbb{N}$ a function $f_n : un \to n$.

Container Functors as Fibred Kan Extensions

We can use proposition 4.3.1 to show that each container functor $T_{A \triangleright B}$ is a left Kan extension in the 2-category $\mathbf{Fib}_{\mathbb{C}}$.

The notion of a Kan extension is a purely 2-categorical construction, so it makes sense to talk about Kan extensions in the category of fibrations. Recall the following definition, which is a natural generalisation of MacLane (1971, §X.3).

Definition 4.3.4 *Let $\mathscr{C}$ be a 2-category and let $F : A \to B$ and $G : A \to C$ be 1-cells in $\mathscr{C}$. The* left Kan extension of G along F *is a 1-cell* $\mathrm{Lan}_F G : B \to C$ *together with a 2-cell $\eta : G \to (\mathrm{Lan}_F G) \circ F$ forming a universal arrow to $- \circ F : \mathscr{C}(B, C) \to \mathscr{C}(A, C)$.*

We can now make the following observation.

Proposition 4.3.5 *Each container functor $T_{A \triangleright B}$ is the Kan extension in* $\mathbf{Fib}_{\mathbb{C}}$ *of the constant functor $1 : \mathrm{dom}_A \to \mathrm{cod}_{\mathbb{C}}$ (taking each $f : \Gamma \to A$ to $\mathrm{id}_\Gamma \in \mathbb{C}/\Gamma$) along the functor $\widetilde{B}$ derived via theorem 3.2.7 from $B \in (\mathbb{C}/A)^I$:*

$$\begin{tikzcd} \mathrm{dom}_A \arrow[r, "1"] \arrow[d, "\widetilde{B}"'] & \mathrm{cod}_{\mathbb{C}} \\ \mathrm{cod}_{\mathbb{C}}^I \arrow[ur, "T_{A \triangleright B} \cong \mathrm{Lan}_{\widetilde{B}} 1"'] & \end{tikzcd} \qquad \textit{in } \mathbf{Fib}_{\mathbb{C}} \ .$$

Proof. From proposition 4.3.1 we see that for each $F : \mathrm{cod}_{\mathbb{C}}^I \to \mathbb{C}$ there is an isomorphism $\mathbf{Fib}_{\mathbb{C}}(\mathrm{cod}_{\mathbb{C}}^I, \mathrm{cod}_{\mathbb{C}})(T_{A \triangleright B}, F) \cong (\mathbb{C}/A)^I(1, F_A B)$, and using theorem 3.2.7 we can write $(\mathbb{C}/A)^I(1, F_A B) \cong \mathbf{Fib}_{\mathbb{C}}(\mathrm{dom}_A, \mathrm{cod}_{\mathbb{C}}^I)(1, F \circ \widetilde{B})$. This is natural in F and so $T_{A \triangleright B} \cong \mathrm{Lan}_{\widetilde{B}} 1$. $\square$

The significance of this observation is more evident in the development of quotient containers, not treated in this thesis (Abbott, 2003).

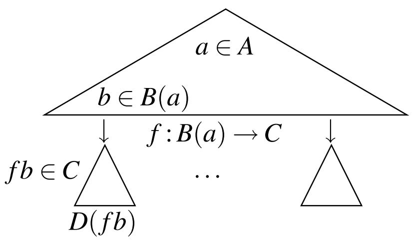
Comparing this construction of $T_{A \triangleright B}$ as a Kan extension in $\mathbf{Fib}_{\mathbb{C}}$ with the corresponding construction when $\mathbb{C} \equiv \mathbf{Set}$ may usefully illuminate the role of fibrations in this thesis. As we have already observed, functors between powers of **Set** *are* fibred as functors over **Set**, so we can translate proposition 4.3.5 into the construction

$$\begin{tikzcd} A \arrow[r, "1"] \arrow[d, "B"'] & \mathbf{Set} \\ \mathbf{Set}^I \arrow[ur, "T_{A \triangleright B} \cong \mathrm{Lan}_B 1"'] & \end{tikzcd} \qquad \text{in } \mathbf{Cat} \;.$$

Here A is the set A regarded as a discrete category and B is the functor $B : A \to \mathbf{Set}$ corresponding to the family $(B_a)_{a \in A}$. So we see that $\mathbf{Fib}_{\mathbb{C}}$ is, in effect, standing in for the category of categories, an object $A \in \mathbb{C}$ as a discrete object is represented by its representable fibration dom_A and the correspondence between families $A \vdash B$ and functors $B : A \to \mathbb{C}$ is precisely the fibred Yoneda lemma (theorem 3.2.7).

4.4 The Bicategory of Containers

Given two containers $(A \triangleright B)$ and $(C \triangleright D)$ in one parameter their extensions can be composed to yield a functor which we'll see is also a container functor. This composition can be explained with reference to the diagram



$$\Big(a : A, f : C^{B(a)} \triangleright \textstyle\sum b : B(a).\ D(fb)\Big)$$

as follows: a value in $T_{A \triangleright B} T_{C \triangleright D} X$ is an $a \in A$ together with a function $B(a) \to T_{C \triangleright D} X$; this function can in turn be decomposed into two parts, a constant part $f : B(a) \to C$ and the data dependent part, $D(fb) \to X$. This gives precisely a decomposition of the composite into shapes and positions.

Just as an I-indexed container $(A \triangleright B) \in \mathscr{G}_I$ yields a functor $\mathbb{C}^I \to \mathbb{C}$ and so can be regarded as a morphism $I \dashrightarrow 1$, so a J-indexed family of containers $(C_j \triangleright D_j)_{j \in J} \in \mathscr{G}_I^J$ with extension a functor $\mathbb{C}^I \to \mathbb{C}^J$ can be regarded as a morphism $I \dashrightarrow J$. We can extend the composite of containers to this case.

Definition 4.4.1 *Given a container* $(A \triangleright B) \in \mathscr{G}_J$ *and a* J*-indexed family of containers* $(C_j \triangleright D_j)_{j \in J}$ *define the composite container*

$$(A \ \triangleright \ B) \circ (C \ \triangleright \ D) \equiv \\ \left(a\!:\!A,\ f\!:\!\prod j\!:\!J.C_j^{B_j(a)} \ \triangleright_{i\,:\,I} \ \sum j\!:\!J.\ \sum b\!:\!B_j(a).\ D_{j,i}(f_j b)\right) \ .$$

This definition is, as might be hoped, compatible with composition of container functors.

Proposition 4.4.2 *Container composition extends to a functor* $-\circ- : \mathscr{G}_I^J \times \mathscr{G}_J \to \mathscr{G}_J$ *which commutes with composition of container functors:* $T_{(A\triangleright B)\circ(C\triangleright D)} \cong T_{A\triangleright B}T_{C\triangleright D}$.

Proof. On objects calculate using intensional choice $\prod_A \sum_B C \cong \sum_{\prod_A B} \varepsilon^* C$ and currying $C^{\sum_A B} \cong \prod_A C^B$ where shown and abbreviating the types as far as possible:

$$\begin{aligned} T_{A\triangleright B}T_{C\triangleright D}X &= \textstyle\sum_A \prod_J \left(\sum_C \prod_I X^D\right)^B \\ &\cong \textstyle\sum_A \prod_J \sum_{C^B} \prod_B \prod_I X^{\varepsilon^* D} && \text{(choice)} \\ &\cong \textstyle\sum_A \sum_{\prod_J C^B} \prod_J \prod_B \prod_I X^{\varepsilon^* D} && \text{(choice)} \\ &\cong \textstyle\sum_A \sum_{\prod_J C^B} \prod_I \prod_J \prod_B X^{\varepsilon^* D} \\ &\cong \textstyle\sum_A \sum_{\prod_J C^B} \prod_I X^{\sum_J \sum_B \varepsilon^* D} && \text{(curry)} \\ &\cong \textstyle\sum_{\sum_A \prod_J C^B} \prod_I X^{\sum_J \sum_B \varepsilon^* D} = T_{(A\triangleright B)\circ(C\triangleright D)}X \ . \end{aligned}$$

By lemma 1.4.7 we can now extend composition of containers to morphisms. □

Given container morphisms $(u,f) : (A \triangleright B) \to (A' \triangleright B')$ and $(v,g) : (C \triangleright D) \to (C' \triangleright D')$ the composite $(u,f) \circ (v,g)$ can be explicitly computed as

$$(u,f)\circ(v,g) = \begin{pmatrix} \lambda a\!:\!A,\ k\!:\!\prod_J C^{B(a)}.\ (ua,\ \lambda j\!:\!J.\ v \cdot k_j \cdot f_{j,a}), \\ i\!:\!I,\ a\!:\!A,\ k\!:\!\prod_J C^{B(a)} \ \vdash\ \lambda j\!:\!J,\ b\!:\!B'_j(ua),\ d\!:\!D'_{i,j}(vk_j f_{j,a} b). \\ \quad (j,\ f_{j,a}b,\ g_{j,k_j f_{j,a}b} d) \end{pmatrix} \ .$$

The projection and identity container morphisms are worth noting.

Proposition 4.4.3 *The projection functors* $\pi_i : \mathbb{C}^I \to \mathbb{C}$ *for* $i \in I$ *are container functors. The container* $\vec{\pi} \equiv (\pi_i)_{i\in I} \in \mathscr{G}_I^I$ *is an identity (up to isomorphism) for composition.*

Proof. Define $\pi_i \equiv (1 \triangleright_{j\in I} \mathrm{Eq}(i,j))$, in other words the shape of π_i is 0 for every parameter in I, except at i where it is equal to 1. Then $T_{\pi_i}X \cong \prod_{j\in I} X_j^{\mathrm{Eq}(i,j)} \cong X_i$.

That $F \circ \vec{\pi} \cong F$ for any $F \in \mathscr{G}_I$ follows by reflection along T from the observation that $T_{\vec{\pi}} \cong \mathrm{id}_{\mathbb{C}^I}$. □

We're now in a position to observe that containers and their composition form a bicategory.

Proposition 4.4.4 *The system of containers over $\mathbb{C}$ and container morphisms extends to a bicategory equipped with a 2-functor into the 2-category of fibred functors and natural transformations between set-indexed powers of $\mathbb{C}$.*

Proof. For sets I,J define $\mathscr{G}(I,J) \equiv \mathscr{G}_I^J$ with $T : \mathscr{G}(I,J) \to [\mathbb{C}^I, \mathbb{C}^J]$. That $- \circ -$ is a bicategory composition operation follows by reflection along each T: full and faithfulness of this functor ensures that since composition of containers is transformed into ordinary functor composition, all the coherence equations of a bicategory are automatically satisfied. □

A special case of container composition is of particular interest when constructing fixed points of types. Given a type $F(\vec{X},Y)$ corresponding to a a functor $F : \mathbb{C}^{I+1} \to \mathbb{C}$ we will be interested in finding a type $G(\vec{X})$ satisfying the isomorphism $F(\vec{X}, G(\vec{X})) \cong G(\vec{X})$. Such a functor is referred to a *fixed point* of $F(\vec{X}, -)$.

A container $F \in \mathscr{G}_{I+1}$ can be written as $(A \triangleright B, E)$ for $B \in (\mathbb{C}/A)^I$ and $E \in \mathbb{C}/A$. Given $G \equiv (C \triangleright D) \in \mathscr{G}_I$ define

$$(A \ \triangleright \ B,E)[(C \ \triangleright \ D)] \equiv \Big(a : A,\ f : C^{E(a)} \ \triangleright_{i \in I} \ B_i(a) + \sum e : E(a).\ D_i(fe)\Big) \quad .$$

This can be seen to be equal to the composite container $F \circ (\vec{\pi}, G)$ and so $F[G]$ satisfies $T_{F[G]} \cong T_F[T_G] = T_F(\mathrm{id}_{\mathbb{C}^I}, T_G)$ and thus $T_{F[G]}X \cong T_F(X, T_G X)$ and as a shorthand we can simply write $(F[G])X = F(X, GX)$.

4.5 Limits and Colimits of Containers

It turns out that each $\mathscr{G}_I$ inherits completeness and cocompleteness from $\mathbb{C}$, and that T preserves completeness. Preservation of cocompleteness is more complex, and only a limited class of colimits is preserved by T.

Proposition 4.5.1 *If $\mathbb{C}$ has limits and colimits of shape $\mathbb{J}$ then $\mathscr{G}_I$ has limits of shape $\mathbb{J}$ and T preserves these limits.*

Proof. We'll proceed by appealing to the fact that T reflects limits (since it is full and faithful), and the proof will proceed separately for products and equalisers.

Products. Let $(A_k \triangleright B_k)_{k\in K}$ be a family of containers in $\mathscr{G}_I$ and compute

$$\begin{aligned}
\prod_{k\in K} T_{A_k \triangleright B_k} X &= \prod_{k\in K} \sum a : A.\ \prod_{i\in I} X_i^{B_{k,i}(a)} \\
&\cong \sum a : \prod_{k\in K} A_k.\ \prod_{k\in K} \prod_{i\in I} X_i^{B_{k,i}(\pi_k a)} && \text{(choice)} \\
&\cong \sum a : \prod_{k\in K} A_k.\ \prod_{i\in I} X_i^{\sum_{k\in K} B_{k,i}(\pi_k a)} && \text{(curry)} \\
&= T_{\prod_{k\in K} A_k \triangleright (\sum_{k\in K} \pi_k^* B_{k,i})_{i\in I}} X
\end{aligned}$$

showing by reflection along T that

$$\prod_{k\in K} (A_k \ \triangleright\ B_k) \cong \left(\prod_{k\in K} A_k \ \triangleright\ \sum_{k\in K} \pi_k^* B_k \right) \ .$$

Equalisers. Given parallel maps $(u,f),(v,g) : (A \triangleright B) \rightrightarrows (C \triangleright D)$ construct

$$(E \ \triangleright\ Q) \rightarrowtail^{(e,q)} (A \ \triangleright\ B) \underset{(v,g)}{\overset{(u,f)}{\rightrightarrows}} (C \ \triangleright\ D)$$

where e is the equaliser in $\mathbb{C}$ of u,v and q is the coequaliser in $(\mathbb{C}/E)^I$ of e^*f, e^*g. To show that $T_{e,q}$ is the equaliser of $T_{u,f}, T_{v,g}$ fix $X \in \mathbb{C}^I$, $U \in \mathbb{C}$ and let $\alpha : U \to T_{A\triangleright B}X$ be given equalising this parallel pair at X.

For $x : U$ write $\alpha(x) = (a,h)$ where $a : A$, $h : \prod_{i\in I} X_i^{B_i(a)}$. The condition on α tells us that $u(a) = v(a)$ and so there is a unique $y : E$ with $a = e(y)$. Similarly we know that $h \cdot f(ey) = h \cdot g(ey)$ and in particular there is a unique $k : Q(y) \to X$ with $h = k \cdot q$.

The assignment $x \mapsto (y,k)$ defines a map $\beta : U \to T_{E\triangleright Q}X$ giving a unique factorisation of α, showing that $T_{e,q}X$ is an equaliser and hence so is (e,q). □

In particular, this result tells us that the limit in $[\mathbb{C}^I, \mathbb{C}]$ of a diagram of container functors is itself a container functor. Observe that this proof works by converting a limit diagram in $\mathscr{G}$ into a combination of a limit and a colimit diagram in $\mathbb{C}$.

It's nice to see that coproducts of containers are also well behaved.

Proposition 4.5.2 *If $\mathbb{C}$ has products and disjoint coproducts of size K then $\mathscr{G}_I$ has coproducts of size K preserved by T.*

Proof. Given a family $(A_k \triangleright B_k)_{k\in K}$ of objects in $\mathscr{G}_I$ calculate (making essential use of disjoint coproducts):

$$\begin{aligned}
\sum_{k\in K} T_{A_k\triangleright B_k} X &= \sum_{k\in K} \sum a : A_k.\ \prod_{i\in I} X_i^{B_{k,i}(a)} \\
&\cong \sum_{k\in K} \sum a : A_k.\ \prod_{i\in I} \Big(\big(\textstyle\coprod^{\circ}_{k'\in K} B_{k',i}\big)(\kappa_k a) \Rightarrow X_i \Big) \\
&\cong \sum a : \textstyle\sum_{k\in K} A_k.\ \prod_{i\in I} \Big(\big(\textstyle\coprod^{\circ}_{k\in K} B_{k,i}\big)(a) \Rightarrow X_i \Big) \\
&= T_{\sum_{k\in K} A_k \triangleright (\coprod^{\circ}_{k\in K} B_{k,i})_{i\in I}} X
\end{aligned}$$

showing by reflection along T that

$$\sum_{k\in K} (A_k \ \triangleright\ B_k) \cong \Big(\sum_{k\in K} A_k \ \triangleright\ \coprod^{\circ}_{k\in K} B_k \Big) \quad . \qquad \square$$

The fate of coequalisers is more complicated. It turns out that $\mathscr{G}_I$ has coequalisers when $\mathbb{C}$ has both equalisers and coequalisers, but they are *not* preserved by T.

The following proposition is not proved here, but follows from observations arising in the abstract framework, section 7.2.

Proposition 4.5.3 *If $\mathbb{C}$ has equalisers and coequalisers then $\mathscr{G}_I$ has coequalisers.* $\square$

The following example shows that coequalisers are not preserved by T.

Example 4.5.4 *Consider the following coequaliser diagram in* $[\mathbb{C},\mathbb{C}]$

$$\begin{tikzcd} X \times X \arrow[r, shift left, "\mathrm{id}_{X\times X}"] \arrow[r, shift right, "(\pi',\pi)"'] & X \times X \arrow[r, two heads] & (X \times X)/\sim \end{tikzcd}$$

where $(x,y) \sim (y,x)$. The functor $X \mapsto X \times X$ is a container functor generated by $(1 \triangleright 2)$, and the coequaliser of the corresponding parallel pair in $\mathscr{G}_1$ is the container $(1 \triangleright 0)$. Note however that $T_{1\triangleright 0} X \cong 1 \ncong (X \times X)/\sim$.

This particular coequaliser *can* be represented as a quotient container (section 7.2), since the quotient $\sim$ is data independent.

It's worth noting that in general filtered colimits aren't preserved by T either.

Example 4.5.5 *Consider the ω-chain in $\mathscr{G}_1$ given by $n \mapsto (1 \triangleright A^n)$ (for fixed A) on objects and $(n \to n+m) \mapsto \pi_{n,m} : A^{n+m} \cong A^n \times A^m \to A^n$ on maps. The filtered colimit of this diagram can be computed in $\mathscr{G}_1$ to be $(1 \triangleright A^{\mathbb{N}})$. However, applying T to this diagram produces the ω-chain*

$$X \xrightarrow{X^{\pi_{0,1}}} X^A \xrightarrow{X^{\pi_{1,1}}} X^{A^2} \xrightarrow{X^{\pi_{2,1}}} \cdots$$

and the colimit of this chain in **Set** *is strictly smaller than $X^{A^{\mathbb{N}}}$.*

4.6 Cartesian Morphisms

An important group of container morphisms is the class of *cartesian* morphisms. These are the container morphisms which preserve the data without either discarding or duplicating positions, and can be regarded as a kind of *linear* morphism of container. We will see an application of this in the definition of the derivative of a container in section 6.4; in this section we will see that filtered colimits of diagrams of cartesian morphisms exist in $\mathscr{G}$ and are preserved by T.

Definition 4.6.1 *A morphism (u,f) in $\mathscr{G}_I$ is* cartesian *iff f is an isomorphism. Write $\widehat{\mathscr{G}_I}$ for the subcategory of $\mathscr{G}_I$ with the same objects but only cartesian morphisms.*

Note indeed that (u,f) is *cartesian* with respect to this definition precisely when it is cartesian (in the sense of section 3.1.3) with respect to the projection functor $\pi : \mathscr{G}_I \to \mathbb{C}$ taking $(A \triangleright B)$ to A.

Note also that for each map of shapes u there is a bijection between cartesian morphisms $(u,f) : (A \triangleright B) \to (C \triangleright D)$ in $\mathscr{G}_I$ and morphisms $\bar{f}$ in $\mathbb{C}^I$ making each square below a pullback:

$$\begin{tikzcd}
B_i \arrow[r, "\bar{f}_i"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & D_i \arrow[d] \\
A \arrow[r, "u"'] & C
\end{tikzcd} \quad ;$$

in other words cartesian morphisms in $\mathscr{G}$ correspond to pullback squares in $\mathbb{C}$.

We can also translate the notion of cartesian morphism into natural transformations between container functors:

Proposition 4.6.2 *A natural transformation $\alpha : T_{A \triangleright B} \to T_{C \triangleright D}$ derives from a cartesian map iff the naturality squares of α are all pullbacks.* $\square$

Such natural transformations are often also called *cartesian*; in this case these maps are cartesian in the sense of fibrations with respect to the "evaluation at 1" functor $[\mathbb{D}, \mathbb{C}] \to \mathbb{C}$.

Cartesian natural transformations into container functors are of particular importance.

Proposition 4.6.3 *Any functor $G \in [\mathbb{C}^I, \mathbb{C}]$ equipped with a cartesian natural transformation $\alpha : G \to T_F$ to a container functor is itself isomorphic to a container functor.*

Proof. Let $F \equiv (A \triangleright B)$ then $(\alpha 1, \mathrm{id}_{(\alpha 1)^* B}) : (G1 \triangleright (\alpha 1)^* B) \to (A \triangleright B)$ is a cartesian map in $\mathscr{G}_I$; this yields a cartesian natural transformation $T_{G1 \triangleright (\alpha 1)^* B} \to T_{A \triangleright B}$. It now follows from the observation that each αX makes GX the pullback along $\alpha 1$ of the map $T_{A \triangleright B} X \to A$ that $G \cong T_{G1 \triangleright (\alpha 1)^* B}$ as required. $\square$

Filtered Colimits of Cartesian Diagrams

Although $\mathscr{G}_I$ has coequalisers they are not preserved in general by T, and as seen above in example 4.5.5 this also applies to filtered colimits. However, all colimit diagrams constructed in $\widehat{\mathscr{G}}$ *are* preserved by T, at least so long as we assume that $\mathbb{C}$ is *finitely accessible* as well as locally cartesian closed.

Definition 4.6.4 (Adámek and Rosický, 1994) *A category $\mathbb{C}$ is* finitely accessible *iff it has all filtered colimits and a generating set of finitely presentable objects.*

For the rest of this section only we will assume that $\mathbb{C}$ is finitely accessible and draw on results from Adámek and Rosický (1994) as necessary to show that $\widehat{\mathscr{G}_I}$ has filtered colimits which are preserved by T (when restricted to $\widehat{\mathscr{G}_I}$), and hence also by the inclusion $\widehat{\mathscr{G}_I} \hookrightarrow \mathscr{G}_I$.

The lemma below follows directly from the corresponding result in **Set** and helps us work with maps from finitely presentable objects to filtered colimits (write $\bigvee D$ for the colimit of a filtered diagram D).

Lemma 4.6.5 *Let $D : \mathbb{J} \to \mathbb{C}$ be a filtered diagram with colimiting cone $d : D \to \bigvee D$ and let U be finitely presentable.*

1. *For each $\alpha : U \to \bigvee D$ there exists $J \in \mathbb{J}$ and $\alpha_J : U \to DJ$ such that $\alpha = d_J \cdot \alpha_J$.*

2. *Given $\alpha : U \to DI$, $\beta : U \to DJ$ such that $d_I \cdot \alpha = d_J \cdot \beta$ there exists $K \in \mathbb{J}$ and maps $f : I \to K$, $g : J \to K$ such that $Df \cdot \alpha = Dg \cdot \beta$.* $\square$

Before the main result we need a technical lemma about filtered colimits in finitely accessible categories.

Lemma 4.6.6 *Given a filtered diagram in $\mathbb{C}^{\to}$ with every edge a pullback then the arrows of the colimiting cone are also pullbacks.*

Proof. We need to show, for each $I \in \mathbb{C}$, that the square

$$\begin{tikzcd} EI \arrow[r, "e_I"] \arrow[d, "\alpha_I"'] & \bigvee E \arrow[d, "\bar{\alpha}"] \\ DI \arrow[r, "d_I"'] & \bigvee D \end{tikzcd}$$

is a pullback, where $E \xrightarrow{\alpha} D$ is the diagram, (d,e) are the components of its colimiting cone and $\bar{\alpha}$ is the factorisation of $d \cdot \alpha$ through e. So let a cone $DI \xleftarrow{a} U \xrightarrow{b} \bigvee E$ satisfying $d_I \cdot a = \bar{\alpha} \cdot b$ be given. Without loss of generality we can assume that U is finitely presentable and we can now appeal to lemma 4.6.5 above.

Construct first $b_J : U \to EJ$ such that $b = e_J \cdot b_J$; then as $d_I \cdot a = \bar{\alpha} \cdot e_J \cdot b_J = d_J \cdot (\alpha_J \cdot b_J)$ there exist $f : I \to K$, $g : J \to K$ with $Df \cdot a = Dg \cdot \alpha_J \cdot b_J = \alpha_K \cdot Eg \cdot b_J$ and so we can construct a factorisation $b_I : U \to EI$ through the pullback over f satisfying $\alpha_I \cdot b_I = a$ and $Ef \cdot b_I = Eg \cdot b_J$. This is a factorisation of (a,b) since $e_I \cdot b_I = e_K \cdot Ef \cdot b_I = e_K \cdot Eg \cdot e_J = e_J \cdot b_J = b$.

This factorisation is unique. Let $b, b' : U \rightrightarrows EI$ be given such that $e_I \cdot b = e_I \cdot b'$. Then there exist $f, f' : I \rightrightarrows J$ with $Ef \cdot b = Ef' \cdot b'$; but indeed there exists $g : J \to K$ with $h \equiv g \cdot f = g \cdot f'$ and so $Eh \cdot b = Eh \cdot b'$. As the square over h is a pullback we can conclude $b = b'$. □

Now we are in a position to state the main result, that the filtered colimit of a cartesian diagram of container functors is itself a container functor.

Proposition 4.6.7 *For each set I the category $\widehat{\mathscr{G}_I}$ has filtered colimits which are preserved by T.*

Proof. Let a diagram $(D \triangleright E) : \mathbb{J} \to \widehat{\mathscr{G}_I}$ be given, i.e. for each $K \in \mathbb{J}$ there is a container $(DK \triangleright EK)$ and for each $f : K \to L$ a cartesian container morphism (Df, Ef).

For each $f : K \to L$ in $\mathbb{J}$, write $\bar{E}f$ for the map $EK \to EL$ derived from cartesian Ef

so that we get the left hand pullback square below:

$$\begin{tikzcd} EK \arrow[r, "\bar{E}f"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & EL \arrow[r, "\bar{e}_L"] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \bigvee \bar{E} \arrow[d] \\ DK \arrow[r, "\bar{D}f"'] & DL \arrow[r, "d_L"'] & \bigvee D \end{tikzcd} .$$

After taking the colimits shown (with colimiting cones d and $\bar{e}$), we know from lemma 4.6.6 that the right hand square is also a pullback and we can interpret the right hand side as a container together with a cartesian cone $(d,e) : (D \triangleright E) \dashrightarrow (\bigvee D \triangleright \bigvee \bar{E})$.

It remains to show that $T_{\bigvee D \triangleright \bigvee \bar{E}} \cong \bigvee T_{D \triangleright E}$, so let a cone $f : T_{D \triangleright E} X \dashrightarrow U$ be given as shown below, where the map k_K takes (a,g) to $(d_K(a), g)$, using the isomorphism $(\bigvee \bar{E})_i(d_K(a)) \cong EK_i(a)$ (for $K \in \mathbb{J}$, $i : I$, $a : DK_j$) derived from (d,e) cartesian.

$$\begin{tikzcd} \sum a : DK_j.\ \prod_{i \in I}(EK_i(a) \Rightarrow X_i) \arrow[r, "k_K"] \arrow[dr, "f_K"'] & \sum a : \bigvee D.\ \prod_{i \in I}((\bigvee \bar{E})_i(a) \Rightarrow X_i) \arrow[d, dashed, "h"] \\ & U \end{tikzcd}$$

To construct h let $a : \bigvee D$ and $g : \prod_{i \in i}((\bigvee D)_i(a) \Rightarrow X_i)$ be given and choose $K \in \mathbb{J}$, $a_K \in DK$ such that $a = d_K(a_K)$, and so we have $(a_K, g) : T_{DK \triangleright EK} X$ and can compute $h(a,g) \equiv f_K(a_K, g)$; this construction of $h(a,g)$ is unique and independent of the choice of K and a_K. □

In Abbott et al. (2003a) this result is used along with proposition 5.2.7 to show that the fixed point of a container $\mu Y. F(X,Y)$ can be constructed as a container. However, this approach has quite a serious drawback: the ambient category $\mathbb{C}$ is required to be locally finitely presentable, which rules out a number of interesting and important categorical models. Instead we develop fixed points of containers using W-types in chapter 5.

4.7 Shapely Types

In Jay and Cockett (1994) and Jay (1995) "shapely types" (in one parameter) in a category $\mathbb{C}$ are defined to be strong pullback preserving functors $\mathbb{C} \to \mathbb{C}$ equipped with a strong cartesian natural transformation to the list type, where the *list type* is the initial algebra $\mu Y. 1 + X \times Y$ (we define this in 5.1.5).

To see the relationship with containers, note that Jacobs (1999, prop 2.6.11) tells us that strong pullback preserving functors are in bijection with fibred pullback preserving functors, and similarly strong natural transformations between such functors correspond to fibred natural transformations. Since the "list type" is given by the container $(\mathbb{N} \triangleright \mathrm{Fin})$, it immediately follows by proposition 4.6.3 that every shapely type is a container functor. Indeed, lemma 4.8.2 shows that the exponentials required to construct the container extensions must necessarily exist.

Conversely, a container constructed with object of positions a discretely finite object is of the form $(A \triangleright \gamma^*\mathrm{Fin})$ for some map $\gamma : A \to \mathbb{N}$ and is therefore evidently a shapely type. We can therefore identify the shapely types with the containers constructed with positions in the universe of discretely finite objects, see after construction 3.3.9.

Definition 4.7.1 *An object* $A \vdash B$ *is* discretely finite *iff a morphism* $u : A \to \mathbb{N}$ *exists such that* $B \cong u^*\mathrm{Fin}$*, i.e. each fibre* $a : A \vdash B(a)$ *is isomorphic to a finite cardinal.*

Say that a container $(A \triangleright B) \in \mathscr{G}_I$ *is* discretely finite *iff each component* B_i *for* $i \in I$ *is discretely finite.*

Note that "discretely finite" is strictly stronger than finitely presentable and other possible notions of finiteness. An immediate consequence of this definition is that the object of finite cardinals is a *generic object* for the category of discretely finite containers, and the following theorem relating shapely types and containers now follows as a corollary.

Theorem 4.7.2 *In a category with a list type the category of shapely functors and strong natural transformations is equivalent to the category of discretely finite containers.* $\square$

However, this thesis tells us more about shapely types. In particular, containers show how to extend shapely types to cover coinductive types. Finally, the representation result for containers clearly translates into a representation result classifying the polymorphic functions between shapely types.

It interesting to note that the "traversals" of Moggi et al. (1999) do not carry over to containers in general, for example the type $\mathbb{N} \Rightarrow X$ does not effectively traverse over the lifting monad $X \mapsto X + 1$.

4.8 Partial Products

An alternative perspective on the construction container types is provided through the notion of a "partial product". In Dyckhoff and Tholen (1987) and Johnstone (1991) we are introduced to the notion of the *partial product* of an object over a family (presented as morphism in the ambient category), originally attributed to Pasynkov (1965). Dyckhoff and Tholen (1987) observe that partial products exist precisely when local exponentials exist, and as noted in Johnstone (1991) the object part of a partial product is precisely the extension of a container at an object. The partial product can therefore be regarded as a building block in the construction of containers.

The following is the categorical definition presented in the references above.

Definition 4.8.1 (Dyckhoff and Tholen, 1987) *The* partial product *of an object X over a morphism $f : B \to A$ is an object P of $\mathbb{C}$ together with a pair of morphisms $p : P \to A$, $e : B \times_A P \to X$ such that given any other P' with maps $p' : P' \to A$ and $e' : B \times_A P' \to X$ there exists a unique $h : P' \to P$ over A such that $e' = e \cdot (B \times_A h)$ as show in the diagram*

$$\begin{tikzcd}
& B \times_A P \arrow[rrr] \arrow[dl, "e"'] \arrow[dd] & & & B \arrow[dd, "f"] \\
X & & B \times_A P' \arrow[ul, dotted, "B \times_A h"'] \arrow[urr] \arrow[ll, "e'"] \arrow[dd] & & \\
& P \arrow[rrr, "p"] & & & A \\
& & P' \arrow[ul, dotted, "h"] \arrow[urr, "p'"'] & &
\end{tikzcd} .$$

It is convenient to translate this directly into the language of slice categories, in which case we can say that the partial product of $X \in \mathbb{C}$ over $B \in \mathbb{C}/A$ is an object $P_{A,B}X \in \mathbb{C}/A$ equipped with an evaluation morphism

$$B \times P_{A,B}X \xrightarrow{e} \pi_A^* X \quad \text{in } \mathbb{C}/A$$

establishing a bijection

$$\mathbb{C}/A(-, P_{A,B}X) \cong \mathbb{C}/A(B \times -, \pi_A^* X) \cong \mathbb{C}\big(\textstyle\sum_A (B \times -), X\big) \ ;$$

this shows that we can regard the partial products over $A \vdash B$ as constructing a right adjoint to the functor $\sum_A (B \times -) : \mathbb{C}/A \to \mathbb{C}$.

The following lemma (which holds in any category $\mathbb{C}$ with finite limits) tells us that the notion of partial product is equivalent to the notion of a container, in particular we can define $T_{A \triangleright B} X = \sum_A P_{A,B} X$.

Lemma 4.8.2 (Dyckhoff and Tholen, 1987, Lemma 2.1) *All partial products over a family $A \vdash B$ exist in $\mathbb{C}$ iff the local exponential X^B exists in $\mathbb{C}/A$ for each $X \in \mathbb{C}/A$.* $\square$

The main interest in this result is that we can now observe that the existence of containers, which we have described here using partial products, is in effect *synonymous* with the existence of exponentials of the positions in the fibre over the shape.

Thus it is clear that the theory of containers can very naturally be generalised to take account of subsystems of types with exponentials, for example we can see that shapely types exist precisely because the exponentials of discretely finite objects exist in a category with finite limits.

Chapter 5

Initial Algebras and Final Coalgebras

In this chapter we discuss the construction of initial algebras and final coalgebras for container functors and the principles in the ambient category $\mathbb{C}$ used to construct them.

5.1 Introducing Fixed Points

Initial algebras and final coalgebras can be regarded as the fundamental building blocks used to introduce infinite data structures into type theory. Initial algebras define "well founded" structures, which can be regarded as the expression of terminating processes; final coalgebras include the possibility of infinite processes.

For example, the functor $X \mapsto X+1$ has two fixed points of interest, namely $\mathbb{N}$ (the natural numbers) and $\mathbb{N}_\infty$, which can be regarded as $\mathbb{N}$ plus an "infinite" number.

First some basic results about initial algebras and final coalgebras.

Definition 5.1.1 *An* algebra *for a functor $F : \mathbb{C} \to \mathbb{C}$ is an object $X \in \mathbb{C}$ together with a morphism $h : FX \to X$; refer to X as the* carrier *of the algebra. An* algebra morphism *$(X,h) \to (Y,k)$ is a morphism $f : X \to Y$ satisfying the identity $f \cdot h = k \cdot Ff$. An* initial algebra *for F is then an initial object in the category of algebras and algebra morphisms.*

More explicitly, an initial algebra is an algebra $\alpha : FA \to A$ such that for any other algebra $h : FX \to X$ there exists a unique *morphism $\overline{h} : A \to X$ satisfying the equation*

$\overline{h} \cdot \alpha = h \cdot F\overline{h}$ *thus:*

$$\begin{tikzcd} FA \arrow[r, "\alpha"] \arrow[d, dashed, "F\overline{h}"'] & A \arrow[d, dashed, "\overline{h}"] \\ FX \arrow[r, "h"'] & X \end{tikzcd} .$$

Similarly, a coalgebra is an algebra for F^{op} in $\mathbb{C}^{\mathrm{op}}$ and a final coalgebra is a terminal object in the category of coalgebras. Writing this out explicitly, a final coalgebra is a coalgebra $\beta : B \to FB$ such that any other coalgebra $h : X \to FX$ induces a unique $\tilde{h} : X \to B$ such that $\beta \cdot \tilde{h} = F\tilde{h} \cdot h$.

The following result tells us that initial algebras and final coalgebras for a functor F are *fixed points* of F, and indeed the initial algebra is often called the least fixed point and the final coalgebra the greatest fixed point.

Proposition 5.1.2 (Lambek's Lemma) *Initial algebras are isomorphisms.*

Proof. Given an initial algebra $\alpha : FA \to A$ construct $\beta \equiv \overline{F\alpha}$ by initiality satisfying $\beta \cdot \alpha = F\alpha \cdot F\beta$. Then $(\alpha \cdot \beta) \cdot \alpha = \alpha \cdot F(\alpha \cdot \beta)$, so by initiality $\alpha \cdot \beta = \mathrm{id}_A$, and then $\beta \cdot \alpha = F(\alpha \cdot \beta) = \mathrm{id}_{FA}$ and so $\beta = \alpha^{-1}$. □

The following useful result about initial algebras tells us that initial algebras with parameters extend to functors.

Proposition 5.1.3 *Given a functor $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ if each endofunctor $F(X, -)$ on $\mathbb{C}$ has an initial algebra $(GX, \alpha X)$ then G extends to a functor and α to a natural transformation.*

Proof. Given $f : X \to Y$ define $Gf : GX \to GY$ by initiality satisfying the equation $Gf \cdot \alpha X = (\alpha Y \cdot F(f, GY)) \cdot F(X, Gf)$. Naturality of α is immediately, and it is easy to see that this construction is functorial. □

Finally an observation to link initial algebras of fibred functors to ordinary initial algebras.

Proposition 5.1.4 *Let $\mathbb{C}$ be locally cartesian closed. If the global part of a fibred functor $F : \mathbb{C} \to \mathbb{C}$ has an initial algebra (A, α) in $\mathbb{C}$ then for each context $\Gamma \in \mathbb{C}$ the algebra $(\Gamma^* A, \Gamma^* \alpha)$ is a local initial algebra for each F_Γ.*

Proof. Let $h : F_\Gamma X \to X$ be an F_Γ-algebra in $\mathbb{C}/\Gamma$; the condition that $f : \Gamma^* A \to X$ be an algebra morphism is the equation $f \cdot \Gamma^* \alpha = h \cdot F_\Gamma f$, which can be written as

$$\begin{tikzcd}
\Gamma^* FA \cong F_\Gamma \Gamma^* A \arrow[rr, bend left, "\Gamma^* \alpha"] \arrow[d, "\Gamma^* F \widetilde{f}"'] \arrow[dr, "F_\Gamma f"] & & \Gamma^* A \arrow[dd, "f"] \\
\Gamma^* F \prod_\Gamma X \arrow[d, phantom, sloped, "\cong"] & & \\
F_\Gamma \Gamma^* \prod_\Gamma X \arrow[r, "F_\Gamma \varepsilon X"'] & F_\Gamma X \arrow[r, "h"'] & X
\end{tikzcd}$$

where $\widetilde{f}$ is the exponential transpose of f. In particular, if we transpose the outer square we get an F-algebra $\widetilde{h \cdot F_\Gamma \varepsilon X} : \Gamma^* \prod_\Gamma X \to \prod_\Gamma X$ with unique algebra morphism $\widetilde{f}$, which is enough to show that f is also a unique algebra morphism. $\square$

We can now define operations μ and ν taking the fixed points of functors. If we regard a functor $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ as a type constructor $F(X,Y)$ then we can can regard the fixed points defined below as types.

Definition 5.1.5 *Given a functor $F : \mathbb{D} \times \mathbb{C} \to \mathbb{C}$ regarded as a type constructor $F(X,Y)$ define $\mu Y.F(X,Y)$ and $\nu Y.F(X,Y)$ to be the initial algebra and final coalgebra respectively of the functor $F(X,-)$.*

We know from proposition 5.1.4 that this definition is sufficient to ensure that the notion of an initial algebra behaves appropriately in the internal language.

To extend this definition of μ types observe that for containers $F \in \mathscr{G}_{I+1}$ and $G \in \mathscr{G}_I$ the operation $G \mapsto F[G]$, with $T_{F[G]} X \cong T_F(X, T_G X)$ as noted at the end of section 4.4, is an endofunctor on $\mathscr{G}_I$.

Definition 5.1.6 *For $F \in \mathscr{G}_{I+1}$ write μF and νF for the initial algebra and final coalgebra respectively of the endofunctor $F[-] : \mathscr{G}_I \to \mathscr{G}_I$.*

We will show in this chapter that functors $\mu, \nu : \mathscr{G}_{I+1} \to \mathscr{G}_I$ exist, or equivalently, that the initial algebra and final coalgebras of a container functor are container functors.

5.2 W-Types and M-Types

In axiomatic type theory (Martin-Löf, 1974; Nordström et al., 1990) the building block for inductive constructions is the W-type. Given a family of constructors $A \vdash B$ the type $\mathrm{W}a\!:\!A.B(a)$ should be regarded as the type of "well founded trees" constructed by regarding each $b\!:\!B(a)$ as a constructor of arity a.

The standard presentation of a W-type is through one type forming rule, an introduction rule and an elimination rule, together with a pair of equations. As the type theoretic development in this thesis focuses entirely on categorical models, we take W types to be *extensionally* defined, in particular the induction elimination rule constructs a *unique* term.

Definition 5.2.1 *A type system* has W-types *iff it has a type constructor*

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash \mathrm{W}_A B} \tag{W-type}$$

together with a constructor term

$$\Gamma,\ a\!:\!A,\ f\!:\!(\mathrm{W}_A B)^{B(a)} \vdash \sup(a,b)\!:\!\mathrm{W}_A B \tag{sup}$$

and an elimination rule

$$\frac{\begin{array}{l}\Gamma,\ \mathrm{W}_A B \vdash C \\ \Gamma,\ a\!:\!A,\ f\!:\!(\mathrm{W}_A B)^{B(a)},\ g\!:\!\prod b\!:\!B(a).C(fb) \vdash h(a,f,g)\!:\!C(\sup(a,f))\end{array}}{\Gamma, w\!:\!\mathrm{W}_A B \vdash \mathrm{wrec}_h(w)\!:\!C(w)} \tag{wrec}$$

satisfying equations (for $a\!:\!A$, $f\!:\!(\mathrm{W}_A B)^{B(a)}$ and $g\!:\!\prod_{\mathrm{W}_A B} C$):

$$\mathrm{wrec}_h(\sup(a,f)) = h(a,f,\mathrm{wrec}_h \cdot f)$$

$$g(\sup(a,f)) = h(a,f,g\cdot f) \implies g = \mathrm{wrec}_h \quad .$$

The W in W-type stands for "Well-ordering", and an element of $\mathrm{W}_A B$ can usefully be thought of as a well founded tree where each node of the tree is given by an element $a \in A$ and the elements $b \in B(a)$ represent the possible descendents of (or branches from) that node. Thus a tree can be described in two parts: a choice of element $a \in A$ together with a function f assigning to each $b \in B(a)$ a descendant tree to the branch b. Thus we get the constructor term $\sup\!:\!T_{A\rhd B}(\mathrm{W}_A B) \to \mathrm{W}_A B$.

The elimination (or *induction*) rule (wrec) arises from the fact that every possible path through a tree $w \in \mathrm{W}_A B$ is finite. The constructor $h(a,f,g)$ constructs a new output over the tree $w = \sup(a,f)$ from the values available over all the descendants of w, namely fb for each $b \in B(a)$. The parameter g programs in the availability of these values.

W-types are initial algebras for a particularly familiar class of functors:

Theorem 5.2.2 W-*types are precisely the initial algebras of container functors in one parameter:*

$$\mathrm{W}_A B \cong \mu X.\ \textstyle\sum_A X^B = \mu X.\ T_{A\rhd B} X \quad .$$

Proof. First observe that the map $\sup : T_{A\rhd B}\mathrm{W}_A B \to \mathrm{W}_A B$ makes $\mathrm{W}_A B$ into an initial $T_{A\rhd B}$-algebra: given an algebra $k : T_{A\rhd B}X \to X$ define $h(a, f, g) \equiv k(a, g)$ and construct $\overline{k} \equiv \mathrm{wrec}_h$. To show that this is an algebra morphism, calculate in context $a : A$ and $f : (W_A B)^{B(a)}$

$$\overline{k}\sup(a, f) = \mathrm{wrec}_h(\sup(a, f)) = h(a, f, \mathrm{wrec}_h \cdot f) = k(a, \overline{k} \cdot f) = k(T_{A\rhd B}\overline{k})(a, f)$$

and uniqueness of this map follows immediately: if a map $g : \mathrm{W}_A B \to X$ also satisfies $g \cdot \sup = k \cdot T_{A\rhd B} g$, then in particular $g(\sup(a, f)) = k(a, g \cdot f) = h(a, f, g \cdot f)$ and so $g = \overline{k}$.

Conversely, to show that $\mu X.T_{A\rhd B}X$ is a W-type we'll need to do a little more work. For conciseness write $Z \equiv \mu X.T_{A\rhd B}X$ and $\sup : T_{A\rhd B}Z \to Z$ for the initial algebra morphism on Z. Now let $Z \vdash C$ and h be given as in the hypotheses of the rule (wrec).

The initiality of Z will only allow us to construct a map into a constant type, so define $D \equiv \sum_Z C$ and from h construct $h' : T_{A\rhd B}D \to D$ as follows.

First observe that $T_{A\rhd B}D = \sum_A (\sum_Z C)^B \cong \sum_A \sum f : Z^B . \prod b : B. C(fb)$ (by intensional choice), and so we can write the arguments of h' as $a : A$, $f : Z^B$ and $g : \prod b : B(a). C(fb)$ and define $h'(a, f, g) = (\sup(a, f), h(a, f, g))$ as the following composite map:

$$T_{A\rhd B}D = \textstyle\sum_A \left(\sum_Z C\right)^B \cong \sum_A \sum f : Z^B.\ \prod b : B.\ C(fb) \xrightarrow{(\sup, h)} \sum_Z C \quad .$$

Initiality of sup induces $\overline{h} : Z \to \sum_Z C$ uniquely satisfying $\overline{h} \cdot \sup = h' \cdot T_{A\rhd B}\overline{h}$. Write $\overline{h} = (\overline{h}_0, \overline{h}_1)$ and we can now write $T_{A\rhd B}\overline{h} \cdot (a, f) = (a, \overline{h}_0 \cdot f, \overline{h}_1 \cdot f)$ as an element of $\sum_A \sum_{Z^B} \prod_B \varepsilon^* C$. This equation can now be written as a pair of equations

$$\overline{h}_0(\sup(a, f)) = \sup(a, \overline{h}_0 \cdot f) \quad , \qquad \overline{h}_1(\sup(a, f)) = h(a, \overline{h}_0 \cdot f, \overline{h}_1 \cdot f) \quad .$$

The equation for $\overline{h}_0$ tells us (by initiality of Z) that in fact $h_0 = \mathrm{id}_Z$, and then the second equation tells us that $\overline{h}_1 = \mathrm{wrec}_h$, and so $Z \cong \mathrm{W}_A B$ as required. □

Dually we can define M-types by a system of type definitions, or as will be sufficient for this thesis, to be final coalgebras of containers.

Definition 5.2.3 *Given a family $A \vdash B$ the* M-type $\mathrm{M}_A B$ *is the final coalgebra*

$$\mathrm{M}_A B \equiv \nu X.\ T_{A\rhd B} X \quad .$$

W-Induction over Lists

One consequence of theorem 5.2.2 is the specialisation of the W-recursion rule to the following deduction rule for lists.

Corollary 5.2.4 *Induction over a list* $\mathrm{List}A \cong \mu X.1 + X \times A$ *is captured by the elimination rule*

$$\frac{\begin{array}{l}\Gamma, \mathrm{List}A \vdash C \\ \Gamma \vdash f : C(\mathsf{nil}) \\ \Gamma,\ a : A,\ l : \mathrm{List}A,\ c : C(l) \vdash g(a,l,c) : C(\mathsf{cons}(a,l))\end{array}}{\Gamma,\ l : \mathrm{List}A \vdash \mathrm{lrec}_{f,g}(l) : C(l)} \qquad (\mathrm{lrec})$$

satisfying the equations

$$\mathrm{lrec}_{f,g}(\mathsf{nil}) = f \qquad \mathrm{lrec}_{f,g}(\mathsf{cons}(a,l)) = g(a,l,\mathrm{lrec}_{f,g}(l))$$

$$k(\mathsf{nil}) = f \wedge k(\mathsf{cons}(a,l)) = g(a,l,k(l)) \implies k = \mathrm{lrec}_{f,g} \quad .$$

Note also that the induction equation $D \cong A \times D + B$ can be written as a list:

$$D = \mu X.\ A \times X + B \cong (\mathrm{List}A) \times B \quad ,$$

and so each inductively defined term $D \vdash t : C$ can be constructed from terms

$$b : B \vdash fb : C(\mathsf{nil}, b)$$

$$a : A,\ l : \mathrm{List}A,\ b : B,\ c : C(l,b) \vdash g(a,l,b,c) : C(\mathsf{cons}(a,l), b)$$

and uniquely satisfies the equations

$$t(\mathsf{nil}, b) = fb \qquad t(\mathsf{cons}(a,l), b) = g(a,l,b,t(a,l)) \quad . \tag{5.1}$$

We'll make use of this observation in the proof of proposition 5.5.1 below.

Constructing W-Types

We can either assume that $\mathbb{C}$ has W-types given axiomatically or, if $\mathbb{C}$ satisfies the necessary preconditions, derive them from theorem 5.2.8 below. Alternatively if $\mathbb{C}$ is a topos we can appeal to proposition 3.6 of Moerdijk and Palmgren (2000).

Proposition 5.2.5 (Moerdijk and Palmgren, 2000, proposition 3.6) *W-types exist in any elementary topos with a natural numbers object.* □

In the rest of this section we'll construct W-types from colimits in $\mathbb{C}$. First we need to set up some auxiliary machinery, deriving largely from Adámek and Rosický (1994). In particular, the result in this subsection as stated here relies on classical set-theoretic reasoning.

Recall that a category $\mathbb{J}$ is said to be *filtered* iff every finite and non-empty diagram in $\mathbb{J}$ has a compatible cocone in $\mathbb{J}$. In Adámek and Rosický (1994) we have the following extension of this notion to an arbitrary regular cardinal $\aleph$.

Definition 5.2.6 *Say that a category $\mathbb{J}$ is $\aleph$-*filtered *iff every subcategory of $\mathbb{J}$ with less than $\aleph$ morphisms has a compatible cocone.*

If a functor $F : \mathbb{C} \to \mathbb{C}$ preserves all $\aleph$-filtered colimits say that F has rank $\aleph$*, and say that F* has rank *iff it has rank for some $\aleph$.*

Note that an ordinary filtered category is precisely an $\aleph_0$-filtered category.

We now have the following important folklore result. A variant of this theorem is proved in Adámek and Koubek (1979), and the case for $\aleph_0$ is a standard result in computer science (eg, Poigné, 1992, §7.3).

Proposition 5.2.7 *If $\mathbb{C}$ has an initial object and colimits of all filtered diagrams then any functor $F : \mathbb{C} \to \mathbb{C}$ with rank has an initial algebra.*

Proof. A sketch proof follows. In the finite case ($\aleph = \aleph_0$) we construct the colimit of the ω-chain

$$0 \longrightarrow F0 \longrightarrow F^2 0 \longrightarrow \dots \longrightarrow \varinjlim_{n \in \aleph_0} F^n 0 \ .$$

Since F preserves this diagram, we can compute $F \varinjlim_n F^n 0 \cong \varinjlim_n F F^n 0 \cong \varinjlim_n F^n 0$; it is a straightforward calculation to verify that this is the required initial algebra morphism.

The generalisation to arbitrary $\aleph$ is a not altogether straightforward set theoretic generalisation of this result. □

We now obtain the following result.

Theorem 5.2.8 *If $\mathbb{C}$ is locally cartesian closed and locally presentable then $\mathbb{C}$ has all* W*-types.*

Proof. It will suffice to show that every one-parameter container functor $T_{A \triangleright B} : \mathbb{C} \to \mathbb{C}$ has rank, and hence has an initial algebra. Decompose $T_{A \triangleright B}$ into the chain of functors

$$\mathbb{C} \xrightarrow{A^*} \mathbb{C}/A \xrightarrow{(-)^B} \mathbb{C}/A \xrightarrow{\Sigma_A} \mathbb{C} \; .$$

We appeal to two results of Adámek and Rosický (1994) to show that all of these functors (and hence their composite) have rank. We know from their theorem 2.39 that each $\mathbb{C}/A$ is accessible, and their proposition 2.23 tells us that every functor between accessible categories with an adjoint has rank. □

Constructing M-Types

We can construct M-types using a dual form of proposition 5.2.7.

Proposition 5.2.9 *If $\mathbb{C}$ is finitely complete, locally cartesian closed and has limits of ω-chains then $\mathbb{C}$ has all M-types.*

Proof. By the dual of proposition 5.2.7 it is enough to show that $T_{A \triangleright B}$ preserves cofiltered limits. Decomposing $T_{A \triangleright B}$ as in the proof of theorem 5.2.8, it is clearly enough to show that Σ_A preserves cofiltered limits, since the remaining components of this decomposition preserve all limits. But we know, for example from Carboni and Johnstone (1995), that Σ_A preserves limits of connected diagrams. □

Note that to construct M-types we only need to make use of the limit of an ω-chain, whereas the construction of W-types can involve the construction of an arbitrarily large filtered colimit, so in some sense M-types are *easier* to construct than W-limits.

This observation certainly extends to the construction of ν-types of containers: if $\mathbb{C}$ has ω-limits and colimits (and so $\mathscr{G}$ is ω-complete) it is straightforward to show (by reflection along T) that the functor $F[-] : \mathscr{G}_I \to \mathscr{G}_I$ preserves ω limits and hence has a final coalgebra. However we do not wish to make such a strong assumption about $\mathbb{C}$ in this thesis.

I believe that it is also possible to construct M-types directly from W-types, but no proof of this is presented here.

5.3 Initial Algebras of Containers

One consequence of theorem 5.2.2 is that in the presence of W-types we can immediately construct μ types for containers in one parameter. However, the construction of a μ type for a container in multiple parameters is a more delicate matter and will require the introduction of more machinery.

Let $F : \mathbb{C}^{I+1} \to \mathbb{C}$ be a container in multiple parameters, which we can write as

$$F(X,Y) \equiv T_{S \triangleright P,Q}(X,Y) = \sum s:S.\ \left(\prod i:I.\ X_i^{P_i(s)}\right) \times Y^{Q(s)} = \sum_S \left(\prod_I X^P \times Y^Q\right) \quad .$$

The task is to compute $(A \triangleright B)$ such that $T_{A\triangleright B}X \cong \mu Y.F(X,Y)$. Clearly

$$A \cong T_{A\triangleright B}1 \cong \mu Y.\ F(1,Y) \cong \mu Y.\ \sum s:S.\ Y^{Q(s)} \cong \mathrm{W}_S Q \quad ,$$

but the construction of $\mathrm{W}_S Q \vdash B$ is more tricky.

In the rest of this chapter we will ignore the index set I and write X^P for $\prod_I X^P$. In particular, this means that the family $B \in (\mathbb{C}/\mathrm{W}_S Q)^I$ will be treated uniformly (as if $I = 1$). The required extra working to take account of I can be routinely added, but will further complicate a presentation which is quite complex enough already. We will therefore take

$$F(X,Y) \equiv \sum_S (X^P \times Y^Q) \quad .$$

To simplify the algebra of types we will write $S, A^Q \vdash P + \sum_Q \varepsilon^* B$ as an abbreviation for the type expression (where ε is an evaluation map $A^Q \times Q \to A$):

$$s:S,\ f:A^{Q(s)}\ \vdash\ P(s) + \sum q:Q(s).\ B(fq) \quad .$$

For consistency with the subsequent result on final coalgebras write the initial algebra on $A = \mathrm{W}_S Q$ as $\psi : \sum_S A^Q \to A$.

Proposition 5.3.1 *Given the notation above, if* $\mathrm{W}_S Q \vdash B$ *is equipped with an isomorphism*

$$S, A^Q\ \vdash\ \varphi : P + \sum_Q \varepsilon^* B \cong \psi^* B$$

then $T_{A\triangleright B}X \cong \mu Y.F(X,Y)$.

Proof. First we show that each $T_{A\triangleright B}X$ is an $F(X,-)$ algebra thus:

$$\begin{aligned} F(X,T_{A\triangleright B}X) &= \textstyle\sum_S \left(X^P \times (\sum_A X^B)^Q\right) \cong \sum_S \left(X^P \times \sum_{A^Q} \prod_Q X^{\varepsilon^* B}\right) \\ &\cong \textstyle\sum_S \sum_{A^Q} \left(X^P \times \prod_Q X^{\varepsilon^* B}\right) \cong \sum_S \sum_{A^Q} X^{P+\Sigma_Q \varepsilon^* B} \\ &\overset{\varphi^{-1}}{\cong} \textstyle\sum_S \sum_{A^Q} X^{\psi^* B} \overset{(\psi,\mathrm{id})}{\cong} \sum_A X^B = T_{A\triangleright B}X \quad . \end{aligned}$$

With variables $s:S$, $g:X^{P(s)}$ and $h:\left(\sum_A X^B\right)^{Q(s)}$ note that we can decompose h into components $\pi\cdot h : A^{Q(s)}$ and $\pi'\cdot h : \prod q : Q(s).X^{B(\pi h q)}$ and so the algebra morphism $in : F(X,T_{A\triangleright B}X) \to T_{A\triangleright B}X$ can be conveniently written as

$$in(s,g,h) = (\psi(s,\pi\cdot h), \left[g;\pi'\cdot h\right]\cdot \varphi^{-1}) \quad ;$$

conversely, given variables $s:S$, $f:A^{Q(s)}$ and $k:X^{B(\psi(s,f))}$ similarly note that $k\cdot\varphi\cdot\kappa'$ can be regarded as a term of type $\prod q : Q(s).X^{B(fq)}$ and so we can write

$$in^{-1}(\psi(s,f),k) = (s,\ k\cdot\varphi\cdot\kappa,\ (f, k\cdot\varphi\cdot\kappa')) \quad .$$

To show that in is an *initial* $F(X,-)$-algebra we need to construct from any algebra $\alpha : F(X,Y) \to Y$ a unique map $\overline{\alpha} : T_{A\triangleright B}X \to Y$ satisfying the algebra morphism equation $\overline{\alpha}\cdot in = \alpha\cdot F(X,\overline{\alpha})$:

$$\begin{tikzcd} F(X,T_{A\triangleright B}X) \arrow[r, "in"] \arrow[d, dashed, "F(X,\overline{\alpha})"'] & T_{A\triangleright B}X \arrow[d, dashed, "\overline{\alpha}"] \\ F(X,Y) \arrow[r, "\alpha"'] & Y \end{tikzcd} \quad .$$

The map $\overline{\alpha}$ can be transposed to a term $A \vdash \widetilde{\alpha} : X^B \Rightarrow Y$ which we will construct by induction on $A = \mathrm{W}_S Q$. Given $s:S$, $f:A^{Q(s)}$ and $k:X^{B(\psi(s,f))}$ construct $g \equiv k\cdot\varphi\cdot\kappa : X^{P(s)}$ and $h \equiv k\cdot\varphi\cdot\kappa' : \prod q : Q(s).X^{B(fq)}$. In this context define $H(s,f,\beta)(k) \equiv \alpha(s,g,\beta(h))$ and compute

$$\begin{aligned} \widetilde{\alpha}(\psi(s,f))(k) &= \overline{\alpha}(\psi(s,f),k) = \overline{\alpha}\cdot in\cdot(s,g,(f,h)) \\ &= \alpha\cdot F(X,\overline{\alpha})\cdot(s,g,(f,h)) = \alpha(s,g,\overline{\alpha}\cdot(f,h)) \\ &= \alpha(s,g,(\widetilde{\alpha}\cdot f)(h)) = H(s,f,\widetilde{\alpha}\cdot f)(k) \quad . \end{aligned}$$

This shows that $\widetilde{\alpha} = \mathrm{wrec}_H$ and thus that $T_{A\triangleright B}X$ is an $F(X,-)$-initial algebra. $\square$

Note that as a corollary of this proposition the isomorphism $P + \sum_Q \varepsilon^* B \cong \psi^* B$ over $\mathrm{W}_S Q$ defines B up to isomorphism, since the container $T_{A\triangleright B}$ is determined up to isomorphism as an initial algebra.

Of course, it remains to prove the hypothesis of the theorem above, that a family $A \vdash B$ with the given isomorphism φ exists; we do this in proposition 5.5.1.

5.4 Final Coalgebras of Containers

The corresponding result for final coalgebras requires a little more structure on B, as the isomorphism φ does *not* fully determine B over $\mathrm{M}_S Q$.

First we need to define what we mean by an initial algebra "over" another fixed point: the following definition turns out to be a special case of an ordinary initial algebra.

Definition 5.4.1 *Given a functor $F : \mathbb{C} \to \mathbb{C}$ with a fixed point $\psi : FA \cong A$ and a functor $G : \mathbb{C}/A \to \mathbb{C}/FA$, define an* algebra over ψ *to be an object $A \vdash B$ together with a morphism $\varphi : GB \to \psi^* B$ over FA. An* algebra morphism $(B, \varphi) \to (B', \varphi')$ *is a morphism $f : B \to B'$ such that $\psi^* f \cdot \varphi = \varphi' \cdot Gf$. Finally define an* initial algebra over ψ *to be an initial object in this category of algebras over ψ.*

Note that an algebra (B, φ) over a fixed point ψ is equivalent to the algebra $(B, \psi^{-1*}\varphi)$ for the functor $\psi^{-1*}G : \mathbb{C}/A \to \mathbb{C}/A$. This observation allows us to see that an initial algebra over a fixed point must be an isomorphism, and that these are just a form of ordinary algebras.

We will work with the specific case of initial algebras over fixed points of the functor $T_{S \triangleright Q}$ for the functor $GX \equiv P + \sum_Q \varepsilon^* X$. We have already used such a fixed point over the initial algebra $W_S Q$ in proposition 5.3.1, and the following proposition allows us to perform a similar construction over the final coalgebra.

Proposition 5.4.2 *Given $F(X, Y) \equiv \sum_S (X^P \times Y^Q)$ as before, define $A \equiv \mathrm{M}_S Q$ with final coalgebra $\psi^{-1} : A \to \sum_S A^Q$. If an object $A \vdash B$ exists with initial algebra*

$$S, A^Q \vdash \varphi : P + \sum\nolimits_Q \varepsilon^* B \longrightarrow \psi^* B$$

over ψ then $T_{A \triangleright B} X \cong \nu Y . F(X, Y)$.

Proof. First note that the construction of $in^{-1} : T_{A \triangleright B} X \to F(X, T_{A \triangleright B} X)$ in the proof of proposition 5.3.1 above works unchanged and so in^{-1} is a coalgebra. We now want to show that this is a final coalgebra, so let $\beta : Y \to F(X, Y)$ be a coalgebra; it will be sufficient to construct $\overline{\beta} : Y \to T_{A \triangleright B} X$ uniquely satisfying $in \cdot F(X, \overline{\beta}) \cdot \beta = \overline{\beta}$.

Start by writing $\beta = (s,g,h) : Y \to \sum_S (X^P \times Y^Q)$, which we can write as

$$Y \xrightarrow{s} S \qquad Y \vdash s^*P \xrightarrow{g} X \qquad Y \vdash s^*Q \xrightarrow{h} Y \quad .$$

The goal then is to construct $\overline{\beta} = (a,k) : Y \to \sum_A X^B$ thus:

$$Y \xrightarrow{a} A \qquad Y \vdash a^*B \xrightarrow{k} X \quad ,$$

and the equation $in \cdot F(X,\overline{\beta}) \cdot \beta = \overline{\beta}$ translates into the commutative square

$$\begin{tikzcd}
Y \arrow[r, "{(s,g,h)}"] \arrow[d, dashed, "{(a,k)}"'] & \sum_S (X^P \times Y^Q) \arrow[d, dashed, "{\sum_S (X^P \times (a,k)^Q)}"] \\
\sum_A X^B \arrow[r, "in^{-1}"'] & \sum_S \left(X^P \times (\sum_A X^B)^Q\right)
\end{tikzcd} \quad .$$

Observe that $\pi \cdot in^{-1} \cdot (a,k) = s$, in other words we can write

$$a = \psi \cdot (s,f) \quad \text{for some} \quad Y \vdash s^*Q \xrightarrow{f} A \quad .$$

Evaluating both edges of this square leads to the equation

$$(s,\ g,\ (a \cdot h,\ h^*k)) = (s,\ k \cdot \phi \cdot \kappa,\ (f,\ k \cdot \phi \cdot \kappa'))$$

which can be interpreted as the following three equations in the type theory (where the detailed dependency of each function symbol is made explicit):

$$\begin{array}{ll}
y : Y,\ p : P(sy) \ \vdash\ g_y p = k_y \varphi_{sy,f_y} \kappa p & g = k \cdot \phi \cdot \kappa \\
y : Y,\ q : Q(sy) \ \vdash\ a h_y q = f_y q & a \cdot h = f \\
y : Y,\ q : Q(sy),\ b : B(f_y q) \ \vdash\ k_{h_y q} b = k_y \varphi \kappa'(q,b) & h^*k = k \cdot \varphi \cdot \kappa' \quad .
\end{array}$$

Now the equation $a = \psi \cdot (s,f) = \psi \cdot (s, a \cdot h)$ fully determines $a : Y \to A$ by finality of A, so the problem remains to determine k. The equations for k can be captured by the following commutative triangle:

$$\begin{tikzcd}
Y \vdash \ s^*P + \sum_{s^*Q} f^*B \arrow[rr, "{(s,f)^*\varphi}"] \arrow[dr, "{[g; h^*k]}"'] & & a^*B \arrow[dl, "k"] \\
& X &
\end{tikzcd} \quad .$$

Note that $s^*P + \sum_{s^*Q} f^*B = (s,f)^*(P + \sum_Q \varepsilon^* B)$ and similarly $a^*B = (s,f)^* \psi^* B$, so we can transpose the right hand edge of this triangle to produce the top and right edges of the square below

$$\begin{tikzcd} S,A^Q \vdash & P + \sum_Q \varepsilon^* B \arrow[r, "\varphi"] \arrow[d, "\mathrm{id}_P + \sum_Q \varepsilon^* \overline{k}"'] & \psi^* B \arrow[d, "\psi^* \overline{k}"] \\ & P + \sum_Q \varepsilon^* \prod_a X \arrow[r, "\alpha"'] & \psi^* \prod_a X \end{tikzcd} \quad .$$

Here $A \vdash \overline{k} : B \to \prod_a X$ is the transpose of k. As ψ is an isomorphism we can write $\prod_{(s,f)} X \cong \psi^* \prod_a X$, and so in particular $\psi^* \overline{k} \cdot \varphi$ is the transpose of $k \cdot (s,f)^* \varphi$. If we can construct $\alpha = [\alpha_0; \alpha_1]$ such that $\alpha \cdot (\mathrm{id}_P + \sum_Q \varepsilon^* \overline{k})$ is the transpose of $[g; h^*k]$ then we can appeal to initiality of φ to conclude that $\overline{k}$ (and hence k) is uniquely determined and so $T_{A \rhd B}$ is a terminal coalgebra.

Taking α_0 to be the transpose of $g : s^*P = (s,f)^*P \to X$ it remains to construct α_1.

Start by observing that $A \vdash \prod_a X$ can be written as

$$a' : A \vdash \prod_a X \equiv \left(\sum y : Y.\ \mathrm{Eq}(ay, a')\right) \Rightarrow X = X^{\sum y:Y.\mathrm{Eq}(ay,a')} \quad ,$$

or more suggestively, as $\prod_a X \equiv \prod y : Y \restriction \mathrm{Eq}(ay, a').X$. Now for $s' : S$ and $f' : A^{Q(s')}$, we can write $\psi^* \prod_a X = \prod y : Y \restriction \mathrm{Eq}(ay, \psi(s', f')).X$, and similarly for $q : Q(s')$ we have $\varepsilon^* \prod_a X = \prod y : Y \restriction \mathrm{Eq}(ay, f'q).X$. The map $A \vdash \overline{k} : B \to \prod_a X$ can be described by the equation

$$a' : A,\ b : B(a),\ y : Y \restriction \mathrm{Eq}(ay, a') \vdash (\overline{k}_{a'} b) y \equiv k_y b \quad .$$

Now define $S, A^Q, Q \vdash \alpha_1 : \varepsilon^* \prod_a X \to \psi^* \prod_a X$ by the equation in context

$$\begin{aligned} & s' : S,\ f' : A^{Q(s')},\ q : Q(s'),\ \theta : \prod y : Y \restriction \mathrm{Eq}(ay, f'q).X,\ y : Y \restriction \mathrm{Eq}(ay, \psi(s', f')) \vdash \\ & \qquad (\alpha_1 \theta) y \equiv \theta(h_y q) \quad . \end{aligned}$$

This is well defined so long as $ay = \psi(s', f') \implies ah_y q = f'q$, but this follows from the equation $ay = \psi(sy, a \cdot h_y)$, which in particular implies that $f' = a \cdot h_y$.

It remains to verify that $\alpha_1 \cdot \sum_Q \varepsilon^* \overline{k}$ is the transpose of h^*k; this follows from the calculation

$$\begin{aligned} & s' : S,\ f' : A^{Q(s')},\ q : Q(s'),\ b : B(f'q),\ y : Y \restriction \mathrm{Eq}(ay, \psi(s', f')) \vdash \\ & \qquad (\alpha_1 \overline{k}_{f'q} b) y = (\overline{k}_{f'q} b)(h_y q) = k_{h_y q} b \quad . \end{aligned}$$

□

5.5 Constructing an Initial Algebra over a Fixed Point

Both propositions 5.3.1 and 5.4.2 rely on the hypothesis that there exists an initial algebra for the functor $X \mapsto P + \sum_Q \varepsilon^* X$ over fixed points $\psi : T_{S \triangleright Q} A \to A$.

Note however, that the *existence* of the initial algebras over fixed points is not yet established. In particular, the functor $\psi^{-1*} G$ is not a container functor, so we cannot simply construct its initial algebra as a W-type. We can construct this initial algebra, but the construction requires further work.

Proposition 5.5.1 *For each fixed point $\psi : T_{S \triangleright Q} A \cong A$ the functor*

$$(A \vdash X) \stackrel{G}{\longmapsto} \Big(S, A^Q \vdash P + \sum\nolimits_Q \varepsilon^* X\Big)$$

has an initial algebra over ψ.

Proof. Write $S, A^Q \vdash \varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$ for the initial algebra over ψ that we wish to construct. As already noted, we cannot directly appeal to W-types to construct this fixed point, so the first step is to create a fixed point equation that we *can* solve. Begin by "erasing" the type dependency of B and construct (writing $\sum_Q Y \cong Q \times Y$, etc)

$$\begin{aligned}\widehat{B} &\equiv \mu Y.\, \sum\nolimits_S \sum\nolimits_{A^Q} (P + Q \times Y) \cong \mu Y.\, \Big(\sum\nolimits_S (A^Q \times P) + \big(\sum\nolimits_S (A^Q \times Q)\big) \times Y\Big) \\ &\cong \mathrm{List}\Big(\sum\nolimits_S (A^Q \times Q)\Big) \times \sum\nolimits_S (A^Q \times P) \ ;\end{aligned}$$

there is no problem in constructing arbitrary lists in $\mathbb{C}$ so $\widehat{B}$ clearly exists.

The task now is to select the "well-formed" elements of $\widehat{B}$. A list in $\widehat{B}$ can be thought of as a putative path through a tree in $\mu Y.\, T_{S \triangleright P,Q}(X, Y)$; we want $B(a)$ to be the set of all valid paths to X-substitutable locations in the tree.

An element of $\widehat{B}$ can be conveniently written as a list followed by a tuple thus

$$([(s_0, f_0, q_0), \ldots, (s_{n-1}, f_{n-1}, q_{n-1})], (s_n, f_n, p))$$

for $s_i : S$, $f_i : A^{Q(s_i)}$, $q_i : Q(s_i)$ and $p : P(s_n)$. The condition that this is a well formed element of $B(\psi(s_0, f_0))$ can be expressed as the n equations

$$f_i(q_i) = \psi(s_{i+1}, f_{i+1}) \quad \text{for } i < n$$

which can be captured as an equaliser diagram

$$\begin{tikzcd} \sum_A B \arrow[r, tail, "e"] \arrow[dr, "\pi_B"'] & \widehat{B} \arrow[r, shift left, "\alpha"] \arrow[r, shift right, "\beta"'] \arrow[d, "\varpi"] & \mathrm{List} A \\ & A & \end{tikzcd}$$

where α, β and ϖ are defined inductively on $\widehat{B}$ as follows (and $\pi_B \equiv \varpi \cdot e$):

$$\begin{aligned} \alpha(\mathsf{nil}, p') &= \mathsf{nil} & \alpha(\mathsf{cons}((s,f,q),l),p') &= \mathsf{cons}(fq, \alpha(l,p')) \\ \varpi(\mathsf{nil},(s,f,p)) &= \psi(s,f) & \varpi(\mathsf{cons}((s,f,q),l),p') &= \psi(s,f) \\ \beta(\mathsf{nil},p') &= \mathsf{nil} & \beta(\mathsf{cons}(b,l),p') &= \mathsf{cons}(\varpi(l,p'),\beta(l,p')) \ . \end{aligned}$$

The property that $b : \widehat{B}$ is an element of B can be written $b : B(\varpi b)$ and can be expressed inductively as follows:

$$\top \implies (\mathsf{nil},(s,f,p)) : B(\psi(s,f)) \tag{5.2}$$

$$fq = \varpi(l,p') \wedge (l,p') : B(fq) \implies (\mathsf{cons}((s,f,q),l),p') : B(\psi(s,f)) \ . \tag{5.3}$$

The converse to (5.3) also holds, since $(\mathsf{cons}((s,f,q),l),p') : B(\psi(s,f)) \iff \mathsf{cons}(fq,\alpha(l,p')) = \mathsf{cons}(\varpi(l,p'),\beta(l,p')) \iff fq = \varpi(l,p') \wedge (l,p') : B(fq)$.

The isomorphism $\widehat{\varphi} : \sum_S \sum_{A^Q}(P + Q \times \widehat{B}) \cong \widehat{B}$ can now be used to construct the initial algebra for B. Writing an element of $\sum_S \sum_{A^Q}(P + Q \times \widehat{B})$ as $(s,f,\kappa p)$ or $(s,f,\kappa'(q,b))$, the function $\widehat{\varphi}$ can be computed thus:

$$\begin{array}{ccc} \sum_S \sum_{A^Q}(P + Q \times \widehat{B}) & \overset{\widehat{\varphi}}{\cong} & \begin{array}{c}\mathrm{List}\big(\sum_S(A^Q \times Q)\big) \\ \times \sum_S(A^Q \times P)\end{array} = \widehat{B} \\ (s,f,\kappa p) & \longleftrightarrow & (\mathsf{nil},(s,f,p)) \\ (s,f,\kappa'(q,(l,p'))) & \longleftrightarrow & (\mathsf{cons}((s,f,q),l),p') \ . \end{array}$$

To show that $\widehat{\varphi}$ restricts to a morphism $\varphi : P + \sum_Q \varepsilon^* B \to \psi^* B$ we need to show for each $s : S$ and $f : A^Q$ that $x : (P(s) + \sum q : Q(s). B(fq))$ implies $\widehat{\varphi}(s,f,x) : B(\psi(s,f))$.

When $x = \kappa p$ we immediately have $\widehat{\varphi}(s,f,\kappa p) = (\mathsf{nil},(s,f,p)) : B(\psi(s,f))$ by (5.2) above. Now let $(s,f,\kappa'(q,(l,p')))$ be given with $(l,p') : B(fq)$ (which means, in particular, that $\varpi(l,p') = fq$) and consider the equation $\widehat{\varphi}(s,f,\kappa'(q,(l,p'))) = (\mathsf{cons}((s,f,q),l),p')$, then by (5.3) this is also in $B(\psi(s,f))$. Thus $\widehat{\varphi}$ restricts to

$$s : S, f : A^{Q(s)} \vdash \varphi_{s,f} : P(s) + \sum q : Q(s).\ B(fq) \longrightarrow B(\psi(s,f)) \ .$$

We have in effect constructed φ making the diagram below commute:

$$\begin{tikzcd}
\sum_S \sum_{A^Q} \left(P + \sum_Q \varepsilon^* B\right) \arrow[rrr, "\varphi"] \arrow[dr, "\pi"] \arrow[dd, hook] & & & \sum_A B \arrow[dl, "\pi_B"'] \arrow[dd, hook, "e"] \\
& \sum_S A^Q \arrow[r, "\psi"] & A & \\
\sum_S \sum_{A^Q} \left(P + Q \times \widehat{B}\right) \arrow[ur, "\pi"] \arrow[rrr, "\widehat{\varphi}"'] & & & \widehat{B} \arrow[ul, "\varpi"']
\end{tikzcd} .$$

Finally to show that φ is an *initial* morphism let $A \vdash X$ be given together with $S, A^Q \vdash h : P + \sum_Q \varepsilon^* X \to \psi^* X$. The condition that a map $A \vdash \overline{h} : B \to X$ is an algebra morphism can be written as the pair of equations

$$s:S,\ f:A^{Q(s)},\ p:P(s)\ \vdash\ \overline{h}_{\psi(s,f)} \varphi_{s,f} \kappa p = h_{s,f} \kappa p \tag{5.4}$$

$$s:S,\ f:A^{Q(s)},\ q:Q(s),\ b:B(fq)\ \vdash\ \overline{h}_{\psi(s,f)} \varphi_{s,f} \kappa'(q,b) = h_{s,f} \kappa'(q, \overline{h}_{fq} b)\ . \tag{5.5}$$

We will construct $\overline{h} : B \to X$ by induction over $\widehat{B}$, but some preparation is required. Write $A \vdash \widehat{B}_\varpi$ for $\widehat{B}$ regarded as a type over A with display map ϖ, then $\widehat{B} \cong \sum_A \widehat{B}_\varpi$. Similarly observe that B as the equaliser $B \cong \mathrm{Eq}(\alpha, \beta)$ can now be written using the isomorphism $A \vdash B \cong \sum_{\widehat{B}_\varpi} \mathrm{Eq}(\alpha, \beta)$. We can now transpose $\overline{h}$ into a form suitable for induction over $\widehat{B}$ thus:

$$\begin{array}{c}
A\ \vdash\ B \cong \sum_{\widehat{B}_\varpi} \mathrm{Eq}(\alpha,\beta) \xrightarrow{\ \overline{h}\ } X \\
\hline\hline
\widehat{B} \cong \sum_A \widehat{B}_\varpi\ \vdash\ \mathrm{Eq}(\alpha,\beta) \longrightarrow \varpi^* X \\
\hline\hline
\widehat{B}\ \vdash\ \tilde{h} : (\varpi^* X)^{\mathrm{Eq}(\alpha,\beta)}
\end{array} .$$

We can relate $\overline{h}$ and $\tilde{h}$ by the equation (the parameter w can be silently ignored: only its presence is important)

$$a:A,\ b:\widehat{B}_\varpi(a),\ w:\mathrm{Eq}(\alpha b, \beta b)\ \vdash\ \tilde{h}(b) = \overline{h}_a(b) : X(a)\ . \tag{5.6}$$

As observed after corollary 5.2.4 we can construct $\tilde{h}$ and verify the equations it satisfies by constructing two terms h_0 and h_1 as follows. We can use the membership rules (5.2, 5.3) to reason about elements of $X^{\mathrm{Eq}(\alpha,\beta)}$, ie $b : B(\varpi b)$ iff $\alpha b = \beta b$ iff $b^* \mathrm{Eq}(\alpha, \beta)$ is inhabited. Now write $\mathrm{Wf}(b) \equiv \mathrm{Eq}(\alpha b, \beta b)$ (abbreviating the type that says that "b is a well formed element of B"), and then $\mathrm{Wf}(\mathsf{nil}, (s, f, p)) \cong 1$ and so we can define h_0:

$$s:S,\ f:A^{Q(s)},\ p:P(s)\ \vdash\ h_0 p \equiv h_{s,f}\kappa p : X(\psi(s,f)) \ .$$

Now consider the construction of h_1 in context:

$$s:S,\ f:A^{Q(s)},\ q:Q(s),\ l:\mathrm{List}\big(\textstyle\sum_S(A^Q\times Q)\big),\ p':\textstyle\sum_S(A^Q\times P),$$
$$x:(l,p')^*\big((\varpi^*X)^{\mathrm{Eq}(\alpha,\beta)}\big)\ \vdash\ h_1:(\mathsf{cons}((s,f,q),l),p')^*\big((\varpi^*X)^{\mathrm{Eq}(\alpha,\beta)}\big) \ .$$

In context s,f,q,l,p' define $b\equiv(\mathsf{cons}((s,f,q),l),p')$; this can now be written as

$$w_1:\mathrm{Wf}(l,p'),\ x:X(\varpi(l,p')),\ w_2:\mathrm{Wf}(b)\ \vdash\ h_1:X(\varpi b) \ .$$

Now $\varpi b=\psi(s,f)$ and the existence of $w_2:\mathrm{Wf}(b)$ implies $\varpi(l,p')=fq$ and hence $x:X(fq)$ and so we can define $h_1\equiv h_{s,f}\kappa'(q,x)$. Now $\tilde{h}\equiv\mathrm{lrec}_{h_0,h_1}$ can be constructed by induction and finally define $\overline{h}$ to be the transpose of $\tilde{h}$.

It remains to verify that the equations (5.1) for $\tilde{h}$ transpose using (5.6) to the equations (5.4, 5.5) for $\overline{h}$:

$$\tilde{h}(\mathsf{nil},(s,f,p))=\overline{h}_{\psi(s,f)}(\mathsf{nil},(s,f,p))=\overline{h}_{\psi(s,f)}\varphi_{s,f}\kappa p$$
$$h_0(s,f,p)=h_{s,f}\kappa p$$
$$\tilde{h}(\mathsf{cons}((s,f,q),l),p')=\overline{h}_{\psi(s,f)}\varphi_{s,f}\kappa'(q,(l,p'))$$
$$h_1((s,f,q),l,p',\tilde{h}(l,p'))=h_{s,f}\kappa'(q,\overline{h}((l,p'))) \ . \qquad \square$$

We conclude our development with the following summary result as a corollary.

Corollary 5.5.2 *If $\mathbb{C}$ has W-types then containers are closed under the construction of μ-types. Similarly, if $\mathbb{C}$ has W and M-types then containers are closed under ν-types as well.*

Note that that since μF and νF are fixed points, they satisfy the isomorphisms

$$\mu F\cong F[\mu F] \quad \text{and} \quad \nu F\cong F[\nu F] \ .$$

5.6 Strictly Positive Types

We now have enough machinery in place to observe that all strictly positive types can be described as containers.

Definition 5.6.1 *A* strictly positive type in n variables *(Abel and Altenkirch, 2000) is a type expression (with type variables $X_1, \ldots, X_n$) built up inductively according to the following rules:*

- *if K is a constant type (with no type variables) then K is a strictly positive type;*
- *each type variable X_i is a strictly positive type;*
- *if F, G are strictly positive types then so are $F+G$ and $F \times G$;*
- *if K is a constant type and F a strictly positive type then $K \Rightarrow F$ is a strictly positive type;*
- *if F is a strictly positive type in $n+1$ variables then $\mu X.F$ and $\nu X.F$ are strictly positive types in n variables (for X any type variable).*

Note that the type expression for a strictly positive type F can be interpreted as a functor $F : \mathbb{C}^n \to \mathbb{C}$, and indeed we can see that each strictly positive type corresponds to a container in $\mathscr{G}_n$.

Let strictly positive types F, G be represented by containers $(A \triangleright B)$ and $(C \triangleright D)$ respectively, then the table below shows the correspondence between strictly positive types and containers.

$$K \mapsto (K \triangleright 0) \qquad X_i \mapsto (1 \triangleright (\delta_{i,j})_{j \in I})$$
$$F+G \mapsto (A+C \triangleright B \overset{\circ}{+} D) \qquad F \times G \mapsto (a : A,\ c : C \triangleright B(a) \times D(c))$$
$$K \Rightarrow F \mapsto \left(f : A^K \triangleright \textstyle\sum k : K.\ B(fk)\right)$$

As we have seen in this chapter the construction of fixed points can be described in a uniform way. Let F be represented by $(S \triangleright P, Q) \in \mathscr{G}_{I+1}$, then for each fixed point $\psi : T_{S \triangleright Q} A \cong A$ of $T_{S \triangleright Q}$ we have constructed in proposition 5.5.1 an initial algebra over ψ, written here as $A \vdash B_A$, of the form

$$s : S,\ f : A^{Q(s)} \vdash \varphi : P(s) + \textstyle\sum q : Q(s).\ B_A(fs) \longrightarrow B_A(\psi(s, f)) \ ;$$

we can now define

$$\mu Y.\, F \mapsto (\mathrm{W}_S Q \,\triangleright\, B_{\mathrm{W}_S Q}) \qquad\qquad \nu Y.\, F \mapsto (\mathrm{M}_S Q \,\triangleright\, B_{\mathrm{M}_S Q}) \enspace .$$

It is intriguing to observe that μ and ν only differ in the type of shapes and that in both cases the positions are constructed in the same way. The reason for this is that each position in a container type, even a coalgebraic type such as the type of streams $X^{\mathbb{N}} \cong \mu Y.X \times Y$, is accessible in a finite number of steps.

A detailed working of this example is instructive. Define $F(X,Y) \equiv X \times Y$ which is given by the container $(S \triangleright P, Q) \equiv (1 \triangleright 1, 1)$. Following the construction of $\nu Y.F(X,Y)$ in this chapter, clearly $A = \nu Y.Y \cong 1$, and so the task is to construct B as a global object satisfying the equation $1 + B \cong B$, with initial solution $B \cong \mathbb{N}$. This gives us the expected result $\nu Y.X \times Y \cong X^{\mathbb{N}}$.

The equation $1 + B \cong B$ also has a final solution, $\mathbb{N}_\infty = \nu X.1 + X$ with an extra "stationary" value $\omega \in \mathbb{N}_\infty$, so it is natural to ask why $X^{\mathbb{N}_\infty}$ is not the solution. A value $f \in X^{\mathbb{N}}$ represents a stream, with each $n \in \mathbb{N}$ indicating how many iterations of $F(X,-)$ are require to reach the position where the value $f(n) \in X$ is found. The problem with $X^{\mathbb{N}_\infty}$ is that the extra value $\omega \in \mathbb{N}_\infty$ represents the unreachable end of the list: thus a value $f(\omega) \in X$ assigns a superfluous value.

Conversely it seems that there are containers which do not correspond to strictly positive types. A probable counterexample is the type of nests, defined as the least solution to the equation

$$N(X) \cong 1 + X \times N(X \times X) \enspace .$$

The datatype N *is* a container since it can be written as $N(X) \cong \sum n : \mathbb{N}.X^{2^n - 1}$, but it should be possible to show that it is not strictly positive following the argument used in Moggi et al. (1999) to show that the type of square matrices is not regular.

Chapter 6

Derivatives of Containers

In this chapter we present an application of containers to the construction of the "derivative" of a data type. This is a useful notion which can be captured by a nice universal property, but only when attention is restricted to the category of containers or a related class of functors.

6.1 Introduction

In his classic functional pearl Huet (1997) shows how to represent a tree with one of its subtrees "in focus" by a pair of the subtree and the one-hole context (or "zipper") in which it sits. The unpublished article McBride (2001) gives a "generic program" for computing the type of one-hole contexts for any regular inductive datatype: remarkably, the key step is to *differentiate* the functor which generates the datatype by the rules we learned from Leibniz (1684). It was an observation in search of an explanation. In this chapter, derived largely from Abbott et al. (2003b), we find such an explanation from a categorical perspective.

The treatment of data types as containers provides the key to the mystery. We can now specify differentiation by a universal property in the category of containers, more precisely $\partial F \cong X \multimap F$ where $H \multimap -$ is the right adjoint of $- \times H$ in the category $\widehat{\mathscr{G}}$ of cartesian morphisms between containers. We thus uncover the linear notion of *tangent* required for the construction of the zipper and the treatment in McBride (2001).

The extension of a container, $\sum_A X^B$ can be regarded as a generalised power series generating a functor in X. In this chapter we will see that the "derivative" of this container can be defined obeying an isomorphism of the form

$$(\partial T_{A \triangleright B})X \cong \sum a : A.\ \sum b : B(a).\ X^{B(a)'(b)}$$

where for any suitable object B and $b : B$ we define $B'(b)$ to be "B with b deleted", or more precisely,

$$b : B \vdash B'(b) \equiv \sum b' : B.\ \neg \mathrm{Eq}(b, b') \ .$$

In fact, it turns out that this construction yields the derivative only when equality on B is "decidable".

Naïvely we would expect $\sum_B B' \cong B \times (B \setminus 1)$ where $B \setminus 1$ is B with a chosen element removed; certainly this does hold in **Set**. However, in the context of a general category $\mathbb{C}$ it is necessary to keep the dependency of B' on B; see example 6.3.8.

6.2 Introducing the Zipper

In Huet (1997) the problem of navigating a "cursor" through an inductively defined data structure is investigated. One programming oriented application is the implementation of an efficient structure oriented editor: in this case it is desirable to efficiently implement the operations of moving the point of focus through the data structure and replacing the part of the structure in focus.

For example, given a binary tree $T \equiv \mu Y.\, 1 + Y \times Y$ (with constructors nil and tree), the constructions of Huet (1997) and McBride (2001) define a "path" into the interior of the tree to be the data type $P \equiv \mathrm{List}(2 \times T) \times T$ where $2 \cong \{L, R\}$ selects which branch at each node is on the path and which branch is bypassed. Here I will refer to P as a *cursor* and the $\mathrm{List}(2 \times T)$ part of the cursor as the *path* to the cursor.

A cursor $([(d_1, t_1), \ldots, (d_n, t_n)], t')$ can be regarded as putting the sub-tree t' "in focus" in a context given by the path of directions and bypassed subtrees. In effect, this can be thought of as a "hole" in the data structure (the path) together with a value, t', to be plugged into the hole.

Navigating a cursor is now easy. Moving the focus up towards the root of the tree can be defined with the help of an auxiliary function which reconstructs the immediately enclosing context from one element of the path and the subtree in context:

$$\mathsf{join}(t', (L, t_R)) = \mathsf{tree}(t', t_R) \qquad \mathsf{join}(t', (R, t_L)) = \mathsf{tree}(t_L, t') \ .$$

Moving up is only meaningful on a non empty path, and can be defined by a single application of join as $\mathsf{up}(\mathsf{cons}(s,p),t') = (p,\mathsf{join}(s,t'))$.

The original tree can be reconstructed from the cursor by iterating join (using the foldl operation from standard Haskell, Peyton Jones, 2003):

$$\mathsf{build}(l,t') = \mathsf{foldl}(\mathsf{join},t',l) \quad .$$

In general we can illustrate a cursor in a general inductively defined data structure as shown in figure 6.1.

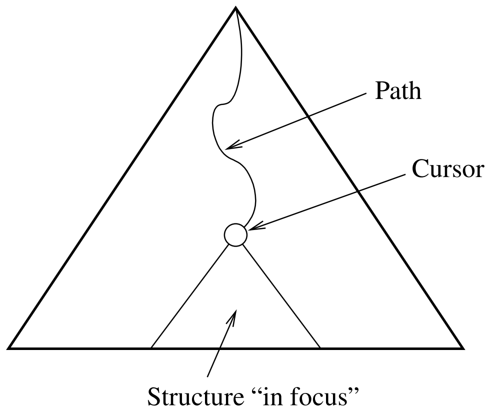


Figure 6.1: A Path in an Inductive Data Structure

In Huet (1997) we are given an explicit construction of the cursor for two main data structures: the binary tree described above, and a tree with lists of nodes at each node, $RX \cong \mu Y.X + \mathrm{List}Y$ (sometimes called a “Rose tree”). McBride (2001) presents a syntactic generalisation of this construction which is reinterpreted in this thesis as a semantic construction involving the derivative of a datatype.

Explaining the Cursor

In general the construction of a path into a data structure $TX \equiv \mu Y.F(X,Y)$, as described above, should be thought of as identifying the location of one Y: into this location we can substitute a further instance of TX. Thus a complete cursor will in general consist of an instance of TX together with a path providing instructions for building the rest of the data structure.

In general a cursor will consist either of the entire structure in focus, in which case the path is empty, or else it will lie below at least one instance of $F(X,Y)$. In this case to specify the location of the cursor we need to specify two things: first, the particular location of Y which contains the path to the cursor; secondly, values for TX to fill in the remaining Y locations.

This value is precisely captured by the construction $(\partial_2 F)(X,TX)$, the partial derivative of F with respect to its second argument (the Y position) with TX substituted in after taking the derivative.

Following this argument we can see that the appropriate data type for a cursor in a data structure $TX \equiv \mu Y.F(X,Y)$ is $\mathrm{List}((\partial_2 F)(X,TX)) \times TX$. We will make this argument more formally in the rest of this chapter.

6.3 Decidable Objects

The building blocks of derivatives will be "decidable objects", where equality of positions is decidable. This is required to distinguish the "hole" of the derivative from the rest of the positions in the data type.

Definition 6.3.1 *Say that an object $A \vdash B$ is* decidable *iff for each $a : A$ the equality relation on $B(a)$ is decidable, ie*

$$a : A,\ b, b' : B(a) \ \vdash\ \mathrm{Eq}(b,b') + \neg\mathrm{Eq}(b,b') \cong 1 \quad .$$

Say that a container $(A \rhd (B_i)_{i \in I}) \in \mathscr{G}_I$ is decidable at i *if $A \vdash B_i$ is decidable, and say that it is* decidable *iff each $A \vdash B_i$ is decidable.*

Note that the notion of decidability is purely local, in that we require that each $B(a)$ be decidable, and not that the total object $\sum_A B$ be decidable. As we will note later, if A is also decidable then indeed $\sum_A B$ is decidable, but not necessarily otherwise.

The notion of decidability corresponds to the ability to separate out each object of B from its companions, as expressed by the following proposition.

Proposition 6.3.2 *An object $A \vdash B$ is decidable iff there exists an object $\sum_A B \vdash B'$, called the* complement of B, *with an isomorphism $A, B \vdash \theta : 1 + B' \cong \pi_B^* B$ in $\mathbb{C}/\sum_A B$ such that $\theta \cdot \kappa = \pi'_B : 1 \to 1 + B'$.*

Proof. ($\Longrightarrow$) Let B be decidable, construct $B' \equiv \sum_B \neg\mathrm{Eq}$ and calculate

$$\begin{aligned} A, b : B \ \vdash\ B &\cong \textstyle\sum b' : B.\ 1 \cong \sum b' : B.\ (\mathrm{Eq}(b,b') + \neg\mathrm{Eq}(b,b')) \\ &\cong \left(\textstyle\sum b' : B.\ \mathrm{Eq}(b,b')\right) + \left(\textstyle\sum b' : B.\ \neg\mathrm{Eq}(b,b')\right) \cong 1 + B' \end{aligned}$$

and it's clear that this isomorphism satisfies $\theta \cdot \kappa = \pi'_B$ since π'_B represents b.

($\Longleftarrow$) Given $B \cong 1 + B'(b)$ satisfying $\theta \cdot \pi'_B = \kappa$ we can construct $B, B \vdash C$ such that $B \vdash B' \cong \sum_B C$ and conclude that for $b, b' : B$ there is (since $\sum_B \mathrm{Eq} \cong 1$) an isomorphism $\mathrm{Eq}(b, b') + C(b, b') \cong 1$. This is enough to show that B is decidable. □

Note that when an object $A \vdash B$ is decidable we can also write the isomorphism as

$$A \ \vdash \ B \times B \cong \sum\nolimits_B \pi_B^* B \cong \sum\nolimits_B (1 + B') \cong B + \sum\nolimits_B B' \ .$$

It is tempting here to imagine that B' is essentially independent of B and can therefore be written as $\pi_B^*(B \setminus 1)$, yielding the isomorphism $B \times B \cong B \times (1 + (B \setminus 1))$ and even perhaps simply $B \cong 1 + (B \setminus 1)$. Although this clearly holds in **Set**, we see below in example 6.3.8 that this is too much to hope in general.

In an extensive category we have a cancellation rule $1 + A \cong 1 + B \implies A \cong B$; the following lemma then immediately follows.

Lemma 6.3.3 *If $A \vdash B$ is decidable and $\pi_B^* B \cong 1 + D$ then $B' \cong D$.*

Some important properties of decidable objects now follow.

Lemma 6.3.4 *If $A \vdash B$ is decidable then so is its complement $\sum_A B \vdash B'$.*

Proof. Fixing $b : B$ then $b_1, b_2 : B'(b)$ is equivalent to $b_1, b_2 : B$ satisfying $b_1 \neq b$ and $b_2 \neq b$, and clearly decidability in B can then be used to compute decidability in B'. □

Lemma 6.3.5 *If $A \vdash B$ is decidable then for each $u : C \to A$ the pullback $u^* B$ is decidable with $(u^* B)' \cong u_B^* B'$.*

Proof. Observing that $b : u^* B$ can be written as $c : C \vdash b : B(uc)$ it is clear that decidability of B is inherited by $u^* B$. To verify the equation for $(u^* B)'$, calculate $\pi_{u^* B}^* u^* B \cong u_D^* \pi_B^* B \cong u_D^*(1 + B') \cong 1 + u_D^* B'$. □

Lemma 6.3.6 *Complements are closed under products and coproducts, and satisfy the following equations.*

$$A + B \ \vdash \ (A + B)' \cong (A' + \pi_A^* B) \mathbin{\varphi} (\pi_B^* A + B')$$
$$A \times B \ \vdash \ (A \times B)' \cong \pi^* A' + \pi'^* B' + \pi^* A' \times \pi'^* B'$$
$$0 \ \vdash \ 0' \cong 0 \qquad 1 \ \vdash \ 1' \cong 0 \ .$$

Proof. The results for $0'$ and $1'$ are immediate.

To show that $A+B$ is decidable, note first that since $\mathbb{C}$ is extensive we can analyse a binding $x : A+B$ into cases: $a : A \vdash x = \kappa a$ or $b : B \vdash y = \kappa' b$, and we know that $\kappa a \neq \kappa' b$ universally holds. Thus given $x, y : A+B$ decidability of equality reduces to decidability of equality separately in A and B.

To verify the complement of $A+B$ calculate:

$$\begin{aligned}
\pi^*_{A+B}(A+B) &\cong \pi^*_{A+B}A + \pi^*_{A+B}B \cong \pi^*_A(A+B) \mathbin{\mathring{+}} \pi^*_B(A+B) \\
&\cong (\pi^*_A A + \pi^*_A B) \mathbin{\mathring{+}} (\pi^*_B A + \pi^*_B B) \\
&\cong (1 + A' + \pi^*_A B) \mathbin{\mathring{+}} (\pi^*_B A + 1 + B') \\
&\cong 1 + ((A' + \pi^*_A B) \mathbin{\mathring{+}} (\pi^*_B A + B')) \quad .
\end{aligned}$$

$A \times B$ is immediately decidable. The calculation of its complement is not needed in this thesis and is omitted. $\square$

The corresponding results for $\sum$ and $\mathbin{\mathring{+}}$ require a little more attention to the base.

Lemma 6.3.7 *If $A \vdash B$ and $C \vdash D$ are both decidable then $A + C \vdash B \mathbin{\mathring{+}} D$ is decidable with complement*

$$\left(\textstyle\sum_{A+C}(B \mathbin{\mathring{+}} D) \;\vdash\; (B \mathbin{\mathring{+}} D)'\right) \cong \left(\textstyle\sum_A B + \sum_C D \;\vdash\; B' \mathbin{\mathring{+}} D'\right) \quad .$$

If A is also decidable then so is $\sum_A B$ with complement

$$\textstyle\sum_A B \;\vdash\; \left(\sum_A B\right)' \cong B' + \sum_{A'} B \quad . \qquad \square$$

Can Complements be Simplified?

We'll conclude this section with an example illustrating the fact that we cannot in general simplify the complement $B \vdash B'$ of a global object B into the form $B' \cong B^*(B \setminus 1)$ for some global object $B \setminus 1$.

Let M be an arbitrary monoid and take $\mathbb{C} \equiv \mathbf{Set}^M$, the category of M-actions; we'll write an object of this category as X with action $m \cdot x$ for $m \in M$, $x \in X$. This category, being a topos, clearly satisfies all the conditions required for the interpretation of the container framework.

If an M-action $X \in \mathbf{Set}^M$ has a complement, it is clear that X' must satisfy the isomorphism $x : X \vdash X'(x) \cong \{x' \mid x' \neq x\}$, and the action of m on $x' \in X'(x)$ must be given by the action on X. Thus for X to have a complement it is necessary that the action have the property

$$x \neq x' \implies m \cdot x \neq m \cdot x'$$

for each $m \in M$, $x, x' \in X$, in other words each $m \cdot -$ is a monomorphism.

Thus we see that unless M is a group (in which case all actions are isomorphisms) the category $\mathbf{Set}^M$ provides a plentiful supply of non-complementable objects.

The question still arises as to whether a complementable object $B^* B \cong 1 + B'$ can be written in a simplified form $B' \cong B^*(B \setminus 1)$. When this *can* be done we get the isomorphism

$$B \times B \cong B + \sum_B B' \cong B + \sum_B B^*(B \setminus 1) \cong B + B \times (B \setminus 1) \ .$$

The following example shows that this cannot be done in general

Example 6.3.8 *Let $M \equiv (\mathbb{Z}, +)$ be the group of integers under addition. An M-action can be specified as a cycle, for example take*

$$X \equiv (a, b) \quad (c, d, e) \ .$$

This means that $X = a, b, c, d, e$ with $2n \cdot a \equiv a$, $(2n+1) \cdot a \equiv b$, $3n \cdot c \equiv c$, $(3n+1) \cdot c \equiv d$, etcetera. This object is clearly complementable (since M is a group) and we can write the product $X \times X$ as the following set of 7 cycles:

$$\begin{array}{ll} X & \{ \quad ((a,a),(b,b)) \quad ((c,c),(d,d),(e,e)) \\ \sum_X X' & \left\{ \begin{array}{l} ((a,b),(b,a)) \quad ((c,d),(d,e),(e,c)) \quad ((c,e),(d,c),(e,d)) \\ ((a,c),(b,d),(a,e),(b,c),(a,d),(b,e)) \\ ((c,a),(d,b),(e,a),(c,b),(d,a),(e,b)) \ . \end{array} \right. \end{array}$$

Note however that $\sum_X X'$ cannot be written in the form $X \times K$ for any object K.

6.4 Derivatives of Containers

In this section we will discuss the treatment of derivatives as linear exponentials. Recall the definition of the category $\widehat{\mathscr{G}}$ of containers and cartesian container morphisms in 4.6.1.

In the context of this chapter we can regard cartesian morphisms between containers as a notion of "linear" morphism. From this perspective we make the following definition of a "linear exponential". This idea of the derivative as a linear exponential appears in Huet (2003).

Definition 6.4.1 *If for containers F and H there exists a universal arrow in $\widehat{\mathscr{G}}$ from the functor $- \times H$ to F say that the* linear exponential of F by H *exists. Write the linear exponential as $H \multimap F$.*

Note that $\times$ here refers to the cartesian product in $\mathscr{G}$, it is just a tensor product in $\widehat{\mathscr{G}}$. In this situation we have a bijection of morphisms in $\widehat{\mathscr{G}}$

$$\frac{G \times H \longrightarrow F}{\overline{\overline{G \longrightarrow H \multimap F}}} \quad .$$

In this chapter we investigate the special case when $H = \pi_i$, ie $HX = X_i$ for $X \in \mathbb{C}^I$; this yields an appropriate notion of derivative.

Now, $\multimap$ gives us a convenient way to introduce the type of *one-hole contexts*:

Definition 6.4.2 *Say that a container $F \in \mathscr{G}_I$ is* differentiable at $i \in I$ *iff the linear exponential $\pi_i \multimap F$ exists. Call this the* derivative at i *of F, written $\partial_i F \equiv \pi_i \multimap F$.*

In the special case $I = 1$ and $\pi_i = \mathrm{Id}$ we drop the reference to i and simply say that F is differentiable *with derivative $\partial F \equiv \mathrm{Id} \multimap F$.*

Write $\mathscr{D}_I$ for the full subcategory of $\widehat{\mathscr{G}_I}$ of containers differentiable at all $i \in I$.

In particular, the counit $s_{F,i} : (\partial_i F) \times \pi_i \to F$ can be regarded as an operation which takes a derivative (a data type with an i-indexed hole) together with a value to plug into that hole and returns a value in the original data type F. The bijection of morphisms can now usefully be written thus:

$$\frac{G \times \pi_i \longrightarrow F}{\overline{\overline{G \longrightarrow \partial_i F}}} \quad .$$

The mapping $F \mapsto \partial_i F$ extends (through universality) to a functor $\partial_i : \mathscr{D}_I \to \widehat{\mathscr{G}_I}$, and we'll see in proposition 6.5.1 that in fact this factors as $\mathscr{D}_I \to \mathscr{D}_I \hookrightarrow \widehat{\mathscr{G}_I}$.

It is possible to deduce the interchange of coproducts and derivatives using this universal property, but it is technically much easier to first prove the following key theorem. Here we write $(A \triangleright_{j \neq i} B_j, C)$ (for fixed $i \in I$) as a shorthand for the container in $\mathscr{G}_I$ with shape B_j for $j \neq i$ and with shape C at i.

Theorem 6.4.3 *If an I-indexed container is decidable at $i \in I$ then it is differentiable at i with derivative*

$$\partial_i (A \ \triangleright \ B) = \left(\textstyle\sum_A B_i \ \triangleright_{j \neq i} \ \pi^*_{B_i} B_j ,\ B'_i \right) \quad .$$

Proof. First some preliminary observations. Define $F \equiv (A \triangleright B)$, $G \equiv (C \triangleright D)$ and recall that $\pi_i = (1 \triangleright_{j \in I} \delta_{i,j})$. Note that $G \times \pi_i \cong (C \triangleright_{j \in I} \delta_{i,j} + D_j)$ and a cartesian morphism $G \times \pi_i \to F$ corresponds to the following set of maps (taking advantage of the symmetry of f to write a cartesian container morphism covariantly)

$$u : C \longrightarrow A \qquad f_j : D_j \cong u^* B_j \text{ for } j \neq i \qquad f_i : 1 + D_i \cong u^* B_i \ .$$

For the purposes of this proof it will be convenient to establish a convention which separates the constant part from the variable part in the notation, so write this as

$$(C \triangleright_{j \neq i} D_j,\ 1 + D_i) \xrightarrow{(u, f_j, f_i)_{j \neq i}} (A \triangleright B) \ .$$

Let $F \equiv (A \triangleright B)$ be decidable at i and write $\theta : 1 + B'_i \cong \pi^*_{B_i} B_i$. Define

$$dF \equiv \left(\textstyle\sum_A B_i \triangleright_{j \neq i} \pi^*_{B_i} B_j,\ B'_i\right)$$

and define $s_F : dF \times \pi_i \to F$ to be the cartesian map $s_F \equiv (\pi_{B_i}, \mathrm{id}_{\pi^*_{B_i} B_j}, \theta)_{j \neq i}$. We will show that this is a universal arrow and therefore that $dF \cong \partial_i F$.

As noted in the preamble to this proof a cartesian morphism $G \times \pi_i \to F$ can be written as $(u : C \to A,\ f_j : D_j \cong u^* B_j,\ f_i : 1 + D_i \cong u^* B_i)_{j \neq i}$. We will construct a unique factorisation $G \to dF$ through s_F; write this as

$$\left(v : C \longrightarrow \textstyle\sum_A B,\ g_j : D_j \cong v^* \pi^*_{B_i} B_j,\ g_i : D_i \cong v^* B'_i\right)_{j \neq i}$$

and the equation to be satisfied is $s_F \cdot ((v, g) \times \pi_i) = (u, f)$. Composing in $\mathscr{G}_I$ this can be written out in detail as the following three equations:

$$\pi_{B_i} \cdot v = u \qquad g_j = v^* \mathrm{id}_{\pi^*_{B_i} B_j} \cdot g_j = f_j \quad \text{for } j \neq i \qquad v^* \theta \cdot (1 + g_i) = f_i \ .$$

Clearly $(g_j)_{j \neq i}$ is fully determined, but it remains to determine v and g_i. Define the term $b \equiv f_i \cdot \kappa : 1 \to u^* B_i$ and calculate $b = f_i \cdot \kappa = v^* \theta \cdot (1 + g_i) \cdot \kappa = v^* \theta \cdot \kappa = v^* \pi'_{B_i}$. The two equations on v, namely $\pi_{B_i} \cdot v = u$ and $v^* \pi'_{B_i} = b$, fully determine $v = (u, b)$.

Finally to determine g_i appeal to lemma 1.4.6 to write $v^* \theta^{-1} \cdot f_i = 1 + g_i$ for a unique g_i, since $(v^* \theta^{-1} \cdot f_i) \cdot \kappa = (1 + g_i) \cdot \kappa = \kappa$. This shows that (u, f) fully determines a unique (v, g) with factorisation $(u, f) = s_F \cdot (v, g)$ and thus that $dF \cong \partial_i F$, showing that F is differentiable at i. □

This theorem only tells us about the derivatives of *decidable* containers. As we will see in example 6.4.4 below, this is not the whole story.

However, the development of derivatives in this chapter is based on the representation established by this theorem. We will therefore for the rest of this chapter concentrate on the special case of decidable containers: from now on let $\mathscr{D}_I$ be the full subcategory of $\widehat{\mathscr{G}_I}$ generated by the decidable containers.

Derivatives and Decidability

Theorem 6.4.3 tells us how to construct the derivative of a decidable container. As we have seen in the preamble to example 6.3.8 there is a plentiful supply of undecidable containers in a general category $\mathbb{C}$, and indeed any non boolean topos will provide such a supply.

The question therefore arises as to whether such containers have derivatives. Example 6.4.4 shows that they can do, which indicates that the treatment of derivatives in this chapter is incomplete.

To construct the example let $M \equiv (\mathbb{N}, +)$ be the monoid of natural numbers under addition and note that each object of the category $\mathbf{Set}^M$ of M-actions can be described as a set equipped with an endofunctor. In particular, let $\mathbf{2} \in \mathbf{Set}^M$ be the set $2 = \{a, b\}$ together with the function $2 \to 2$ taking each element of 2 to a. This is not a decidable object, but we have the following result.

Example 6.4.4 *The container $(1 \triangleright \mathbf{2})$ with extension functor $X \mapsto X^{\mathbf{2}}$, for $\mathbf{2} \in \mathbf{Set}^M$ described above, has derivative $\partial(X^{\mathbf{2}}) \cong 0$.*

Proof. Let $G \equiv (A \triangleright B)$ be a container (constructed with $\mathbb{C} \equiv \mathbf{Set}^M$), then a cartesian container morphism $G \times \mathrm{Id} \to (1 \triangleright \mathbf{2})$ amounts to an isomorphism $A^*\mathbf{2} \cong B + 1$ in the slice category $\mathbf{Set}^M/A$. This can be described by a commuting diagram in $\mathbf{Set}$

$$\begin{tikzcd}
A^*\mathbf{2} = A+A \arrow[r, phantom, "\underset{\theta}{\cong}"] \arrow[rrr, bend left, "\beta"] \arrow[dr, "\varepsilon_A"'] & {} & A+B \arrow[dl, "{[\mathrm{id}_A;\pi_B]}"] \arrow[rrr, bend left, "\mathrm{id}_A + g"] & A+A \arrow[r, phantom, "\cong"] \arrow[dr] & {} & A+B \arrow[dl] \\
& A \arrow[rrr, "f"'] & & & A &
\end{tikzcd}$$

where the action of A is f, the action of B is g, the induced action on $A^*\mathbf{2}$ is $\beta \equiv \kappa \cdot f \cdot \varepsilon_A$ and $\varepsilon_A \equiv [\mathrm{id}_A; \mathrm{id}_A]$; we'll write $\alpha \equiv \kappa \cdot \varepsilon_1$ for the action on $2 = 1 + 1$.

Now let $a \in A$ be given and pull back the above diagram to yield an isomorphism $\theta_a : 1 + 1 \cong 1 + B_a$ satisfying the equation $(1 + g_a) \cdot \theta_a = \theta_a \cdot \alpha$. This is of course impossible, so we conclude that A must be the empty set. Thus given a cartesian container morphism $G \times \mathrm{Id} \to (1 \triangleright \mathbf{2})$ then $G = 0$ and there is indeed a bijection with maps $G \to 0$, showing that $0 \cong \partial(1 \triangleright \mathbf{2})$. □

This example informs us that the notion of decidable container used to capture derivatives in this thesis is incomplete. It seems possible that the construction of the derivative can be thought of as selecting a decidable sub-component of the container being differentiated.

This is an irksome gap in the treatment here: the remainder of this chapter uses the representation in defined in this theorem to define constructions on derivatives. This means that for the rest of the chapter we will be proving theorems about decidable containers only.

The Importance of Containers

Note that theorem 6.4.3 only holds if we restrict our attention to the category of containers. If we instead were to seek a cartesian universal arrow $\partial_i F \times \pi_i \to F$ in the category of functors $[\mathbb{C}^I, \mathbb{C}]$, the theorem above would not hold, and indeed 1 would not be the derivative of the identity functor. We will present a counterexample to show this.

First a couple of properties of the functor $\neg\neg : \mathbb{C} \to \mathbb{C}$ defined by $\neg X \equiv 0^X$ and so $\neg\neg X = 0^{(0^X)}$.

Proposition 6.4.5 *For all $X \in \mathbb{C}$ the isomorphism $X \times \neg\neg X \cong X$ holds.*

Proof. Write $n : X \to \neg\neg X$ for the transpose of the evaluation map $X \times \neg X \to 0$; we clearly have maps $X \xrightarrow{(\mathrm{id}_X, n)} X \times \neg\neg X \xrightarrow{\pi} X$ with $\pi \cdot (\mathrm{id}_X, n) = \mathrm{id}_X$, and it is clear that $\pi \cdot (\mathrm{id}_X, n) \cdot \pi = \pi$, so it remains to show $\pi' \cdot (\mathrm{id}_X, n) \cdot \pi = n \cdot \pi$ is equal to π'.

However, observe that *all* morphisms into $\neg X$ (and hence $\neg\neg X$) are equal (in other words, $\neg X$ is a subobject of 1) since parallel maps into $\neg X$ transpose into parallel maps into 0. Since $\mathbb{C}$ has distributive coproducts, any morphism into 0 is an isomorphism, and all maps out of 0 are necessarily equal. □

We will now show that the functor $\neg\neg$ cannot be a container.

Proposition 6.4.6 *If $\mathbb{C}$ has disjoint coproducts then the functor $\neg\neg : \mathbb{C} \to \mathbb{C}$ is a container only if $\mathbb{C}$ is degenerate (ie, equivalent to the single object category).*

Proof. Let $\neg\neg X \cong T_{A \triangleright B} X$ for some container $(A \triangleright B)$. From $\neg\neg 1 \cong 1$ conclude $A \cong 1$ and so $\neg\neg X \cong X^B$. Note also that $\neg\neg(1+1) \cong 1$, which means that there is a unique morphism $f : B \to 1+1$, and by disjointness of coproducts this can be decomposed into $f = f_0 + f_1 : B \cong B_0 + B_1 \to 1+1$ for $B_0 = \kappa^* B$ and $B_1 = \kappa'^* B$.

Since f is unique the equation $[\kappa';\kappa] \cdot (f_0 + f_1) = f_0 + f_1$ holds, and now composing with $\kappa : B_0 \to B_0 + B_1$ yields the equation $\kappa \cdot f_0 = (f_0 + f_1) \cdot \kappa = [\kappa';\kappa] \cdot (f_0 + f_1) \cdot \kappa = [\kappa';\kappa] \cdot \kappa \cdot f_0 = \kappa' \cdot f_0$, forming two sides of a commutative square

$$\begin{tikzcd}
B_0 \arrow[r, "f_0"] \arrow[d, "f_0"'] & 1 \arrow[d, "\kappa'"] \\
1 \arrow[r, "\kappa"'] & 1+1
\end{tikzcd} .$$

Disjointness of coproducts gives us a map $B_0 \to 0$, which implies $B_0 \cong 0$ and similarly $B_1 \cong 0$ and so $B \cong 0$. However, $0^B \cong 0^0 \cong 1 \not\cong 0 \cong \neg\neg 0$ unless $\mathbb{C}$ is degenerate. □

Writing X for the container $(1 \triangleright 1)$, observe that $\partial X \cong 1$ and the universal arrow s_X is just the isomorphism $\partial X \times X \cong 1 \times X \cong X$. We can now observe that the universal property of ∂ does not extend in general from the category of containers to the category of functors.

Example 6.4.7 *The function* $s_X : 1 \times \mathrm{Id} \to \mathrm{Id}$ *capturing the derivative* $\partial X \cong 1$ *is not a universal arrow in the category of functors* $[\mathbb{C},\mathbb{C}]$ *and cartesian natural transformations. In particular, the isomorphism* $(\neg\neg) \times \mathrm{Id} \cong \mathrm{Id}$ *does not have a factorisation* $(\neg\neg) \to \partial\,\mathrm{Id} \cong 1$.

The argument is very simple: the isomorphism $\neg\neg X \times X \cong X$ *would, through universality, establish a cartesian natural transformation* $\neg\neg X \to \partial X \cong 1$*. However, by proposition 4.6.3 this would imply that* $\neg\neg$ *is a container.*

6.5 Properties of Derivatives

We can now use the concrete representation of a derivative provided by theorem 6.4.3 to show that derivatives behave just like the traditional derivatives of calculus.

The following proposition tells us that derivatives of decidable containers can be iterated, and that taking a derivative can be regarded as a functor $\mathscr{D}_I \to \mathscr{D}_I$.

Proposition 6.5.1 *If a container is decidable then so is its derivative.*

Proof. This follows from lemma 6.3.4. □

The elementary type operations interact with differentiation as one might expect. For simplicity the proofs are presented here for derivatives in one parameter of the form

$$\partial(A \triangleright B) \cong \left(\textstyle\sum_A B \triangleright B'\right)$$

but the generalisation to ∂_i is routine.

Proposition 6.5.2 *Derivatives of decidable containers satisfy the following*

$$\partial(F+G) \cong \partial F + \partial G \qquad \partial(F \times G) \cong \partial F \times G + F \times \partial G$$
$$\partial K \cong 0 \qquad \partial_i \pi_j \cong \delta_{i,j} \ .$$

The equations for $F+G$, $F \times G$ and K are unchanged if ∂ is replaced by ∂_i.

Proof. Each of these isomorphisms follows pretty directly from the results already proved for decidable objects. Let $F \equiv (A \triangleright B)$, $G \equiv (C \triangleright D)$, then:

$$\begin{aligned}
\partial(F+G) &\cong \partial((A \triangleright B) + (C \triangleright D)) \cong \partial(A + C \triangleright B \overset{\circ}{+} D) \\
&\cong \left(\textstyle\sum_{A+C}(B \overset{\circ}{+} D) \triangleright (B \overset{\circ}{+} D)'\right) \cong \left(\textstyle\sum_A B + \textstyle\sum_C D \triangleright B' \overset{\circ}{+} D'\right) \\
&\cong \left(\textstyle\sum_A B \triangleright B'\right) + \left(\textstyle\sum_C D \triangleright D'\right) \cong \partial F + \partial G \ .
\end{aligned}$$

The following derivation of $\partial(F \times G)$ omits both variables and explicit weakening, but these can be inferred unambiguously:

$$\begin{aligned}
\partial(F \times G) &\cong \partial((A \triangleright B) \times (C \triangleright D)) \cong \partial(A \times C \triangleright B + D) \\
&\cong \left(\textstyle\sum_{A \times C}(B+D) \triangleright (B+D)'\right) \\
&\cong \left(\textstyle\sum_{A \times C}(B+D) \triangleright (B'+D) \overset{\circ}{+} (B+D')\right) \\
&\cong \left(\textstyle\sum_{A \times C} B \triangleright B'+D\right) + \left(\textstyle\sum_{A \times C} D \triangleright B + D'\right) \\
&\cong \left(\textstyle\sum_A B \triangleright B'\right) \times (C \triangleright D) + (A \triangleright B) \times \left(\textstyle\sum_C D \triangleright D'\right) \\
&\cong \partial F \times G + F \times \partial G \ .
\end{aligned}$$

A constant type $K \equiv (K \triangleright 0)$ has derivative $\partial K = (\sum_K 0 \triangleright 0') \cong (0 \triangleright 0) \cong 0$. Similarly $\partial_i \pi_j = (\delta_{i,j} \triangleright \delta'_{i,j}) \cong (\delta_{i,j} \triangleright 0) \cong \delta_{i,j}$. □

The product and coproduct rules above can be generalised to infinite products and coproducts thus:

$$\begin{aligned}
&\partial\left(\textstyle\sum_{k \in K} F_k\right) \cong \textstyle\sum_{k \in K} \partial F_k \\
&\partial\left(\textstyle\prod_{k \in K} F_k\right) \cong \textstyle\sum_{k \in K}\left(\partial F_k \times \textstyle\prod_{k' \neq k} F_{k'}\right) \ .
\end{aligned}$$

An analogous form of the chain rule for derivatives also holds. Here we'll prove the chain rule for the special case of composition of the form $F[G]$ where F takes two parameters and G takes one parameter. The generalisation to full composition yields an equation of the form

$$\partial_i(F \circ G) \cong \sum_{j\in J}(((\partial_j F)\circ G)\times \partial_i G_j)$$

but this will not be proved here, as the proof below is quite complex enough.

Proposition 6.5.3 *For decidable $F \in \mathscr{G}_2$ and $G \in \mathscr{G}_1$ the chain rule holds:*

$$\partial(F[G]) \cong \partial_1 F[G] + \partial_2 F[G] \times \partial G \ .$$

Proof. First note that in general

$$\begin{aligned}
\partial(A \vartriangleright B+C) &\cong \left(\sum_A (B+C) \vartriangleright (B+C)'\right)\\
&\cong \left(\sum_A (B+C) \vartriangleright (B'+\pi_B^* C) \mathbin{\dot{+}} (\pi_C^* B + C')\right)\\
&\cong \left(\sum_A B \vartriangleright B'+\pi_B^* C\right) + \left(\sum_A C \vartriangleright \pi_C^* B + C'\right) \ .
\end{aligned}$$

Now let $F \equiv (A \vartriangleright B, E)$ and $G \equiv (C \vartriangleright D)$ and then (omitting explicit variables and weakening where possible)

$$\begin{aligned}
&\partial(F[G]) = \partial\left(\sum_A f : C^E \vartriangleright B + \sum e : E.\ D(fe)\right) \cong (1)+(2)\\
&\text{where} \quad (1) \equiv \left(\sum_A \sum f : C^E.\ B \vartriangleright B' + \sum e : E.\ D(fe)\right)\\
&\qquad\qquad (2) \equiv \left(\sum_A \sum f : C^E.\ \sum e : E.\ D(fe) \vartriangleright B + \left(\sum e' : E.\ D(fe')\right)'\right) \ .
\end{aligned}$$

The first part, $(1) \cong \partial_1 F[G]$ follows pretty immediately:

$$\begin{aligned}
(1) &\cong \left(\sum_A \sum_B f : C^E \vartriangleright B' + \sum e : E.\ D(fe)\right)\\
&\cong \left(\sum_A B \vartriangleright B', E\right)[(C \vartriangleright D)] \cong \partial_1 F[G] \ .
\end{aligned}$$

To reduce (2) to $\partial_2 F[G] \times \partial G$ it is necessary to be a little more explicit about the variables and context. First note in context A, $f : C^E$, $e : E$, $d : D(fe)$ that

$$\left(\sum e' : E.\ D(fe')\right)' \cong D(fe)'(d) + \sum e' : E'.\ D(fe') \ .$$

Next observe that we can rearrange the context to bring $e : E$ before $f : C^E$ and can then write $C^E \cong C^{E'+1} \cong C^{E'} \times C$ giving us new bindings $f' : C^{E'}$ and $c : C$. The function value fe must then be replaced by c, and so finally in context A, $e : E$, $f : C^{E'}$, $c : C$, $d : D(c)$ we can write

$$D(c)'(d) + \sum e' : E'.\ D(fe')$$

which is isomorphic (modulo the change of base just described) to $(\sum e' : E.D(fe'))'$. This finally allows us to write

$$\begin{aligned}(2) &\cong \left(\sum_A \sum_E \sum f : C^{E'}.\ \sum_C D \triangleright B + D' + \sum e' : E'.\ D(fe')\right) \\ &\cong \left(\sum_A E \triangleright B, E'\right)[(C \triangleright D)] \times \left(\sum_C D \triangleright D'\right) \cong \partial_2 F[G] \times \partial G\ . \qquad \square\end{aligned}$$

Derivatives of Fixed Points

An immediate consequence of the chain rule above is that if fixed points $\partial(\mu F)$ and $\partial(\nu F)$ exist then since $\mu F \cong F[\mu F]$ and $\nu F \cong F[\nu F]$ exist they must satisfy the equations

$$\begin{aligned}&\partial(\mu F) \cong \partial_1 F[\mu F] + \partial_2 F[\mu F] \times \partial(\mu F) \\ &\partial(\nu F) \cong \partial_1 F[\nu F] + \partial_2 F[\nu F] \times \partial(\nu F)\ .\end{aligned}$$

The development in this thesis of derivatives is incomplete (note for example the remarks after theorem 6.4.3), and it should be possible to develop the theory here to prove the following conjecture.

Conjecture 6.5.4 *If a container $F \in \mathscr{G}_2$ is differentiable with fixed point $A \cong F[A]$ in $\mathscr{G}_1$ then A is differentiable with derivative*

$$\partial A \cong \mu(\partial_1 F[A] + \partial_2 F[A] \times -) \cong \mathrm{List}(\partial_2 F[A]) \times \partial_1 F[A]\ .$$

This is justified by the observation that if it exists ∂A is a fixed point, and as we have already seen in the construction of the fixed points of containers, paths into fixed points are always given by *initial* algebras, and we have already seen that a derivative is basically a path into a container.

We can show that this conjecture holds for the special case when F is decidable and the fixed point $A = \mu F$ is constructed as a filtered colimit. This special case of the construction of μF is basically an application of theorem 5.2.7 using the cartesian colimits in section 4.6 (and is the method used in Abbott et al., 2003a).

First we need the following technical lemma that allows us to lift filtered colimits to decidable containers.

Lemma 6.5.5 *The complement of a cartesian filtered colimit of decidable containers is the filtered colimit of their complements, hence $\partial(\bigvee_i F_i) \cong \bigvee_i \partial F_i$.*

Proof. Let each $F_i \equiv (A_i \triangleright B_i)$ and first observe that $\bigvee F \cong (\bigvee A \triangleright \bigvee B)$. Then for each $i \to j$ the following diagram can be constructed in $\mathbb{C}$:

$$\begin{tikzcd}
B'_i \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner"] & B'_j \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner"] & \dots \arrow[r] & \bigvee B' \arrow[d] \\
B_i \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner"] & B_j \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner"] & \dots \arrow[r] & \bigvee B \arrow[d] \\
A_i \arrow[r] & A_j \arrow[r] & \dots \arrow[r] & \bigvee A
\end{tikzcd}$$

Since the bottom row consists of pullback squares then so does the top row (since the pullback of a complement is a complement), and thus $\bigvee \partial F$ is the top right hand arrow.

The object $\bigvee B'$ is the required complement of $\bigvee B$. $\square$

Finally we can make the following statement about derivatives of initial algebras.

Proposition 6.5.6 *If F is a decidable container in two parameters and μF can be written as a filtered colimit in $[\mathbb{C}, \mathbb{C}]$ then μF is differentiable with derivative*

$$\partial \mu F \cong \mu(\partial_1 F[\mu F] + \partial_2 F[\mu F] \times -) \ .$$

Proof. Define $G[-] \equiv \partial_1 F[\mu F] + \partial_2 F[\mu F] \times -$ and use the chain rule to observe

$$\partial \mu F \cong \partial(F[\mu F]) \cong \partial_1 F[\mu F] + \partial_2 F[\mu F] \times \partial \mu F = G[\partial \mu F] \ ;$$

this gives us a morphism $\theta : \mu G \to \partial \mu F$.

Conversely note that $\mu F \cong \bigvee_n (F^n[0])$ and so $\partial \mu F \cong \partial \bigvee_n (F^n[0]) \cong \bigvee_n \partial(F^n[0])$. We can therefore construct morphisms $\alpha_n : \partial(F^n[0]) \to G^n[0]$ inductively by setting

$$\alpha_{n+1} \equiv \partial_1 F[\kappa_n] + \partial_2 F[\kappa_n] \times \alpha_n$$

where $\kappa_n : F^n[0] \to \mu F$ is the colimiting cone of F. From the morphisms α we obtain a morphism $\phi : \partial \mu F \to \mu G$. It remains only to show that θ and ϕ are inverses. $\square$

Note that this result gives us a decomposition of the derivative of an inductive type $GX \equiv \mu Y. F(X, Y)$ into a path to a structural substitution point ("a path to Y") together with an ordinary derivative in X thus:

$$(\partial G)X \cong \mu Y.\ (\partial_1 F(X, GX) + \partial_2 F(X, GX) \times Y) \cong P_F X \times \partial_1 F(X, GX)$$

where the path $P_F X$ is a list of derivatives in the second argument

$$P_F X \equiv \mathrm{List}(\partial_2 F(X, GX)) \ ,$$

or more concisely: $\partial G \cong P_F \times (\partial_1 F)[G]$ and $P_F = \mathrm{List}((\partial_2 F)[G])$.

Writing $s_{F,1} : \partial_1 F(X,Y) \times X \to F(X,Y)$ and $s_{F,2} : \partial_2 F(X,Y) \times Y \to F(X,Y)$, or more concisely, $s_{F,i} : \partial_i F \times \pi_i \to F$, $i = 1,2$, for the first and second derivative reconstruction maps we can decompose the reconstruction map s_G for the derivative of G into the composite $s_G = b_G \cdot (\mathrm{id}_{P_F} \times s_{F,1})$

$$\partial G \times \pi_1 \cong P_F \times (\partial_1 F)[G] \times \pi_1 \xrightarrow{\mathrm{id}_{P_F} \times s_{F,1}} P_F \times G \xrightarrow{b_G} G$$

where the structural reconstruction map b_G is given by iterating the second derivative reconstruction map $s_{F,2}G : (\partial_2 F)[G] \times G \to G$ at G along the list P_F.

This presentation of the derivative of an inductive type mirrors the construction of the positions of an inductive container as a set of paths, and ties the development in this chapter back to the introduction in section 6.2.

Chapter 7

Conclusions and Future Work

The work in this thesis lays the ground for further research into generic programming as well as opening up other interesting avenues of investigation into the application of dependent types.

7.1 Conclusions

The theory of containers as data types presented in this thesis provides an abstract categorical semantics for generic programming. In particular, we have seen how the semantic properties of containers are strictly tied to specific properties of the ambient or modelling category.

This thesis has concentrated on showing how containers can provide a functorial semantic model for a wide class of polymorphic datatypes. We have seen how the "strictly positive" types are captured as containers, and we have also seen that this class of types is not exhaustive: it seems likely that "nested" data types generated using higher-order induction can also be modelled using containers.

The data types modelled by containers are particularly well behaved, and can be regarded as "data independent" in a very strong sense: all polymorphic functions between container types are defined purely in terms of their operations on "shapes" and "positions".

Note indeed that container types cannot support such operations as equality comparison on data, since a generic equality operation cannot satisfy the naturality equations in any category of interest (in particular, naturality of equality implies that every morphism is monic). It should be possible to capture container types with equality by making use of "dependent" containers: this is future work of particular interest, as the generalisation to the dependent case appears to be a direct generalisation of the current work. We can then look to capture data types similar to the type classes of Haskell using dependent containers.

One contribution of this thesis is a precise characterisation of how the semantics of data types corresponds to properties in the ambient category or *model*. For the purposes of this thesis we have restricted our attention to locally cartesian closed categories, but it is possible to extend this interpretation to virtually *any* category (at the cost of losing specific closure properties depending on the model) by restricting the fibres of the interpretation to those morphisms which have enough pullbacks and restricting the class of "positions" used to construct containers to the exponentiable types. This generalisation can be dealt with in future work by developing the "abstract framework" described in the next section. For example we have seen how "shapely types" arise by restricting positions to the "discretely finite" objects of the model.

We have seen that products of containers depend on products and coproducts in the ambient category, and similarly that coproducts of containers rely on disjoint coproducts in the model. As equalisers of containers derive from equalisers and coequalisers in the model, we can see how container equality types depend on quotients in the underlying model. Finally we have seen how the construction of fixed points derives directly from W-types and M-types in the model.

Thus the theory of containers can be adapted to the chosen semantic model: for example, we see that shapely types are not closed under the construction of final coalgebras, because the class of discretely finite objects does not support the construction of the appropriate algebra.

The application of containers to derivatives is one instance of a generic operation on datatypes, and one which seems to be handled well by the container formulation. It would be interesting to see if other generic operations on datatypes can be captured in a similar way: this would constitute a kind of higher-order calculus of datatypes. The work on derivatives also ties the development of containers to prior work by Joyal (1986) and Bergeron et al. (1997) in an unexpected way, as well as to more concrete applications to type theory such as Huet (1997, 2003), McBride (2001) and Gibbons (2000).

The derivatives of containers also cast light in another interesting direction: there is a linkage between the type of *paths* into a inductively (or coinductively) constructed type and the construction of fixed points of containers. The construction of the positions of a fixed point can be regarded as a process of constructing the type of all possible paths into the datatype: the difference is that the derivative allows one path to be isolated from the rest as a shape, hence the role of decidability in the construction of derivatives.

The research in this thesis seems to have opened up a number of interesting opportunities for both further theoretical work and applications.

Related Work

There is much related work in the area of semantics of data types, particularly the subject of "types with shape". In section 4.7 we see how the shapely types of Jay (1995) can be regarded as a particular class of containers obtained by restricting the positions to the universe of discretely finite types.

Another important approach to container types is in Hoogendijk and de Moor (2000), where container types are defined in terms of categories of relations. Some work will be required to link the work of Hoogendijk and de Moor (2000) to the work in this thesis; we require very little logical structure on the ambient category, whereas regularity plays a key role in the correspondence between categories and the allegory framework (Freyd and Scedrov, 1990; Johnstone, 2002) used in their work.

The more abstract work of Hasegawa (2002) and Joyal (1986) deals with "normal" and "analytic" functors on **Set**. In these works the restriction to finite exponentials rules out the construction of coinductive types with infinite sets of positions (just as for shapely types), and the restriction to **Set** rules out weaker model categories.

7.2 Future Work

The work in this thesis creates a number of opportunities for future work, some of which are discussed below. The most interesting are the following:

- Extending the the class of types beyond "strictly positive" types.
- Extending containers to capture dependent types. This is a straightforward generalisation with some very powerful consequences.
- Developing inductive families.
- Developing quotients of containers, extending the treatment to *analytic* functors.
- Extending the treatment of derivatives.
- Abstracting the framework in which the theory of containers is developed.

Higher-Order Fixed Points

We have already observed that the syntactically generated class of "strictly positive" types does not necessarily exhaust the class of types which can be captured using the formalism of containers. For example, "nested types" which are constructed using higher-order fixed points, eg Abel et al. (2003), should be describable as containers.

Recall that the higher-order recursion equation $NX \cong 1 + X \times N(X \times X)$ has a solution of the form $NX \cong \sum n : \mathbb{N}. X^{2n-1}$, which we believe not to be strictly positive, essentially because the syntactic character of strictly positive types restricts the positions which can be constructed. To show that *all* equations of this form have solutions in containers it will be necessary to develop the theory of fixed points of containers a little further.

The transformation $N \mapsto \lambda X. 1 + X \times N(X \times X)$ can be regarded as a functor $F : \mathscr{G}_1 \to \mathscr{G}_1$, and so we can look to construct N as the least fixed point in $\mathscr{G}_1$ of F. This construction should generalised readily to an algebra of higher-order induction on container types.

Dependent Containers

Throughout this thesis we have taken containers to define container functors of the form $\mathbb{C}^I \to \mathbb{C}^J$. If, however, we take the indexing sets I and J to be objects of $\mathbb{C}$

instead, we can define a larger class of container functors of the form $\mathbb{C}/I \to \mathbb{C}/J$, where a container $(A \triangleright B)$ is now given by a pair $(A \in \mathbb{C}/J,\ B \in \mathbb{C}/(I \times \sum_J A))$; the rest of the development of the theory of containers carries through largely unchanged.

It now turns out that a great deal of the interesting type theoretic structure of $\mathbb{C}$ can be described using this framework of dependent containers. For instance, given a map $\gamma\colon \Delta \to \Gamma$ in $\mathbb{C}$, all of the functors γ^*, $\sum_\gamma$ and $\prod_\gamma$ turn out to be container functors.

We can now capture polymorphic dependent types as container types. Just as a (strictly positive) type theoretic expression $\star \to \star$ is captured as a container functor $\mathbb{C} \to \mathbb{C}$, so a type expression $\star \to (K \to \star)$ can be a *dependent* container functor of the form $\mathbb{C} \to \mathbb{C}/K$. More generally we move from capturing types of the form $\star^n \to \star$ (modelled by container functors $\mathbb{C}^n \to \mathbb{C}$) to types $(A \to \star) \to (B \to \star)$ (for constant types A and B).

The use of dependent containers will allow the functor G of proposition 5.5.1 to be captured as a fixed point of a container functor, and indeed we would expect a more general form of this theorem. This would seem to provide an interesting alternative approach to the construction of fixed points of families extending the treatment of Dybjer (1991, 1996).

Quotients of Containers

The category of containers lacks good coequalisers: although $\mathscr{G}$ has coequalisers of parallel pairs, virtually none of them are preserved by T. We can increase the class of parallel pairs with coequalisers by introducing the notion of a "quotient container".

A quotient container is a natural generalisation of an analytic functor (Hasegawa, 2002; Joyal, 1986): a quotient container $(A \triangleright B \;/\; G)$ is a container $(A \triangleright B)$ together with G a (fibre-wise) subgroup of automorphisms on B. The extension of a quotient container is then a functor $T_{A \triangleright B/G} \colon \mathbb{C}^I \to \mathbb{C}$ thus;

$$T_{A \triangleright B/G} X \equiv \sum\nolimits_A \prod\nolimits_I \left(X^B / \sim_G\right) \quad ,$$

where $f \sim_G f'$ as maps $B \to X$ iff there exists $g \in G$ such that $f = f' \cdot g$.

It is possible to define a notion of morphism between quotient containers constructing a category $\mathscr{Q}_I$ equipped with a full and faithful functor $T : \mathscr{Q}_I \to [\mathbb{C}^I, \mathbb{C}]$ such that a representation result similar to proposition 4.3.1 holds (Abbott, 2003).

The theory of container quotients can be developed in the abstract framework via a generalisation of the partial product construction in section 4.8 in a way which naturally generalises the development in this thesis.

Further Work on Derivatives

Once quotients of containers have been developed, the theory of derivatives of containers becomes richer. In particular, it is now possible to capture the "bag" functor which can be defined as

$$\mathrm{Bag}\, X \equiv \sum n : \mathbb{N}.\ X^n / n!$$

where $n!$ is the permutation group on the set of n elements. This functor plays a role analogous to the exponential in classical calculus, in that there is an isomorphism $\partial\,\mathrm{Bag} \cong \mathrm{Bag}$.

There are several areas for further investigation into derivatives of containers. Firstly, the development in this thesis makes little use of the universal property and relies on the description of a derivative as the complement of a decidable container. As pointed out in example 6.4.4 this approach is incomplete.

Also, there is much prior work in this area. Two references in particular are Joyal (1986) and the book Bergeron et al. (1997). In particular, the role of analytic functors needs to be looked at more closely.

The Abstract Framework

Although the first half of this thesis is dedicated to the description of the internal language and its interpretation in fibrations, we actually make little use of the power of this framework. In particular, all the results in this thesis are written for application in a locally cartesian closed category $\mathbb{C}$.

There is much scope for rather more application of the fibrational framework, in particular there are many categories of interest which fail to be cartesian closed and yet where Pi types (or local exponentials) can be defined by restricting the class of *display maps* used in the type theory.

For example, to interpret containers in categories of domains or related frameworks for the interpretation of "partial" functional programming we would need to take account of the fact that these categories cannot be locally cartesian closed. Domain theoretic models of dependent type theory (Jacobs, 1999; Taylor, 1986) provide a promising area for extending the work described here.

A detailed description of this extension of this more abstract framework has been omitted from this thesis, as otherwise the framework would have dominated the development. The key notion seems to that of a *fibration with comprehension* (Jacobs, 1991, 1999), which can be seen (modulo some subtleties) to generalise the notion of a category with families.

More generally, the theory of containers can be developed with respect to a fairly arbitrary comprehension fibrations p and q, where p specifies the allowable positions and q the allowable shapes. So long as p has q-indexed coproducts we can define a fibration of containers by a construction on fibrations:

$$\begin{tikzcd}
\mathscr{G}_I \arrow[r] \arrow[d, "\Sigma^*(p^I)^{\mathrm{op}}"] \arrow[dd, bend right=60, "g_I"'] \arrow[dr, phantom, "\lrcorner", very near start] & (\mathbb{E}^{(I)})^{(\mathrm{op})} \arrow[d, "(p^I)^{\mathrm{op}}"] \\
\mathbb{D} \arrow[r, "\Sigma"'] \arrow[d, "q"] & \mathbb{C} \\
\mathbb{C} &
\end{tikzcd} .$$

In this diagram Σ is part of the *comprehension* structure on q taking $A \in \mathbb{D}_\Gamma$ to the domain of its display map $\Sigma A = \Gamma.A \xrightarrow{\pi_A} \Gamma = pA$, and $p^{\mathrm{op}} : \mathbb{E}^{(\mathrm{op})} \to \mathbb{C}$ is the *opposite fibration* of p obtained by defining $(\mathbb{E}^{(\mathrm{op})})_\Gamma \equiv (\mathbb{E}_\Gamma)^{\mathrm{op}}$. The fibration g_I can be written as $g_I = \mathbf{Fam}_q((p^I)^{\mathrm{op}})$ and inherits a considerable amount of fibred and quantification structure by this construction.

Further structure relating p and q, derivable from an abstraction of the partial product in section 4.8, allows us to construct a fibred functor $t : g_I \times q^I \to q$, or equivalently $T : g_I \to (q^I \Rightarrow q)$. This plays exactly the same role as the more familiar functor $T : \mathscr{G}_I \to [\mathbb{C}^I, \mathbb{C}]$, since the fibre over 1 of $q^I \Rightarrow q$ is $\mathbf{Fib}_{\mathbb{C}}(q^I, q)$.

This approach involves the detailed development of rather more of the fibrational framework than is presented here, including a detailed development of the 2-categorical properties of fibrations, but can provide more insight into the general construction. In particular, it appears that some of the theory of container quotients can be developed in this way using a modification of the partial product construction.

Bibliography

M. Abbott. Quotients of containers. Presentation to PSSL 79, 2003.

M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *Proceedings of Foundations of Software Science and Computation Structures*, volume 2620 of *Lecture Notes in Computer Science*, 2003a.

M. Abbott, T. Altenkirch, N. Ghani, and C. McBride. Derivatives of containers. In *6th International Conference on Typed Lambda Calculi and Applications*, volume 2701 of *Lecture Notes in Computer Science*, 2003b.

A. Abel and T. Altenkirch. A predicative strong normalisation proof for a λ-calculus with interleaving inductive types. In *Types for Proof and Programs, TYPES '99*, volume 1956 of *Lecture Notes in Computer Science*, 2000.

A. Abel, R. Matthes, and T. Uustalu. Generalized iteration and coiteration for higher-order nested datatypes. In *Foundations of Software Science and Computation Structures*, volume 2620 of *Lecture Notes in Computer Science*, 2003.

J. Adámek and V. Koubek. Least fixed point of a functor. *Journal of Computer and System Sciences*, 19:163–178, 1979.

J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*. Number 189 in London Mathematical Society Lecture Note Series. Cambridge University Press, 1994.

E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. Functorial polymorphism, preliminary report. In G. Huet, editor, *Logical Foundations of Functional Programming*, chapter 14, pages 315–327. Addison-Wesley, 1990.

H. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pages 117–309. Oxford University Press, 1992.

J. Bénabou. Fibrations petites et localement petites. *C. R. Acad. Sc. Paris*, 281:A831–A834, 1975.

J. Bénabou. Fibred categories and the foundation of naive category theory. *Journal of Symbolic Logic*, 50(1):10–37, 1985.

F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial Species and Tree-Like Structures*, volume 67 of *Encyclopedia of Mathematics*. Cambridge University Press, 1997.

F. Borceux. *Handbook of Categorical Algebra 2*, volume 51 of *Encyclopedia of Mathematics*. Cambridge University Press, 1994.

A. Carboni and P. Johnstone. Connected limits, familial representability and Artin glueing. *Math. Struct. in Comp. Science*, 5:441–459, 1995.

R. L. Crole. *Categories for Types*. Cambridge University Press, 1993.

N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.

P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

P. Dybjer. Internal type theory. In *TYPES*, pages 120–134, 1995.

P. Dybjer. Representing inductively defined sets by wellorderings in Martin-Löf's type theory. *Theoretical Computer Science*, 170(1–2):245–276, 1996.

R. Dyckhoff and W. Tholen. Exponentiable morphisms, partial products and pullback complements. *J. Pure Appl. Algebra*, 49(1-2):103–116, 1987.

P. J. Freyd and A. Scedrov. *Categories, Allegories*, volume 39 of *North-Holland Mathematical Library*. North-Holland, 1990.

J. Gibbons. Generic downwards accumulations. *Science of Computer Programming*, 37:37–65, 2000.

R. Hasegawa. Two applications of analytic functors. *Theoretical Computer Science*, 272(1-2):112–175, 2002.

M. Hofmann. On the interpretation of type theory in locally cartesian closed categories. In *CSL*, pages 427–441, 1994.

M. Hofmann. *Extensional Constructs in Intensional Type Theory*. Springer, 1997a.

M. Hofmann. Syntax and semantics of dependent types. In A. M. Pitts and P. Dybjer, editors, *Semantics and Logics of Computation*, volume 14, pages 79–130. Cambridge University Press, 1997b.

P. Hoogendijk and O. de Moor. Container types categorically. *Journal of Functional Programming*, 10(2):191–225, 2000.

G. Huet. The zipper. *Journal of Functional Programming*, 7(5):549–554, 1997.

G. Huet. Linear contexts and the sharing functor: Techniques for symbolic computation. In *Workshop on Thirty Five Years of Automath*, 2003.

J. M. E. Hyland. The effective topos. In A. S. Troelstra and D. van Dalen, editors, *The L.E.J.Brouwer Centenary Symposium*, volume 110 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, 1982.

B. Jacobs. *Categorical Type Theory*. PhD thesis, Katholieke Universiteit Nijmegen, 1991.

B. Jacobs. On cubism. *Journal of Functional Programming*, pages 379–391, 1996.

B. Jacobs. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. Elsevier, 1999.

C. B. Jay. A semantics for shape. *Science of Computer Programming*, 25:251–283, 1995.

C. B. Jay and J. R. B. Cockett. Shapely types and shape polymorphism. In *ESOP '94: 5th European Symposium on Programming*, Lecture Notes in Computer Science, pages 302–316. Springer, 1994.

P. T. Johnstone. *Topos Theory*. Academic Press, 1977.

P. T. Johnstone. Partial products, bagdomains and hyperlocal toposes. In M. P. Fourman, P. T. Johnstone, and A. M. Pitts, editors, *Applications of Categories in Computer Science*, London Mathematical Society Lecture Note Series, pages 315–339. Cambridge University Press, 1991.

P. T. Johnstone. *Sketches of an Elephant: A Topos Theory Compendium: Volume 1*. Number 43 in Oxford Logic Guides. Oxford University Press, 2002.

A. Joyal. Foncteurs analytiques et espèces de structures. In *Combinatoire Énumérative*, number 1234 in Lecture Notes in Mathematics, pages 126–159. Springer, 1986.

J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic*. Number 7 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1986.

G. Leibniz. Nova methodus pro maximis et minimis, itemque tangentibus, qua nec irrationals quantitates moratur. *Acta eruditorum*, 1684.

S. MacLane. *Categories for the Working Mathematician*. Number 5 in Graduate Texts in Mathematics. Springer, 1971.

P. Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Proceedings of the Logic Colloquium*, pages 73–118. North-Holland, 1974.

C. McBride. The derivative of a regular type is its type of one-hole contexts. URL `http://www.dur.ac.uk/c.t.mcbride/`. Available electronically, 2001.

I. Moerdijk and E. Palmgren. Wellfounded trees in categories. *Annals of Pure and Applied Logic*, 104:189–218, 2000.

E. Moggi, G. Bellè, and C. B. Jay. Monads, shapely functors and traversals. *Electronic Notes in Theoretical Computer Science*, 29, 1999.

B. Nordström, K. Petersson, and J. M. Smith. *Programming in Martin-Löf's Type Theory*. Number 7 in International Series of Monographs on Computer Science. Oxford University Press, 1990.

R. Paré and D. Schumacher. Abstract families and the adjoint functor theorems. In P. T. Johnstone and R. Paré, editors, *Indexed Categories and Their Applications*, number 661 in Lecture Notes in Mathematics. Springer-Verlag, 1978.

B. A. Pasynkov. Partial topological products. *Transactions of the Moscow Mathematical Society*, 13:153–272, 1965.

S. Peyton Jones. Haskell 98 language and libraries: The revised report. *Journal of Functional Programming*, 13(1), 2003.

A. Poigné. Basic category theory. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1 of *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

R. A. G. Seely. Locally cartesian closed categories and type theory. *Math. Proc. Camb. Phil. Soc.*, 95:33–48, 1984.

T. Streicher. *Semantics of Type Theory*. Progress in Theoretical Computer Science. Birkhäuser Verlag, 1991.

P. Taylor. *Recursive Domains Indexed Category Theory and Polymorphism*. PhD thesis, Cambridge University, 1986.

P. Taylor. *Practical Foundations of Mathematics*. Cambridge University Press, 1999.

D. Turner. Elementary strong functional programming. In R. Plasmeijer and P. Hartel, editors, *First International Symposium on Functional Programming Languages in Education*, number 1022 in Lecture Notes in Computer Science, pages 1–13. Springer, 1996.

P. Wadler. Theorems for free! In *Functional Programming Languages and Computer Architecture*, pages 347–359, 1990.