THE UNIVERSITY OF CALGARY

Categorical Combinators for Charity

by

Peter M. Vesely

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

CALGARY, ALBERTA
NOVEMBER, 1996

Abstract

A 2-categorical notion of *parametric combinator* is introduced. These combinators are operations for constructing arrows in a category and satisfy a 2-categorical abstraction of parametricity for strictly covariant functors. Inductive and co-inductive datatypes are compatible with these combinators and their universal properties are closely related to parametricity. The Charity programming language is based on *strong datatypes* which are a specialization of 2-categorical datatypes. *Stong combinators* (which are an appropriate specialization of 2-categorical combinators) provide a model of polymorphic functions in Charity.

Contents

Chapter 1

Introduction

In this thesis we investigate categorical combinators and parametricity. By a combinator we are informally thinking of some operation which takes some functions as arguments and produces a new function. As a first example consider the usual set theoretic function composition:

$$\frac{f : A \to B \qquad g : B \to C}{f ; g : A \to C}$$

where $f;g$ denotes the composite "f followed by g". If functions are viewed as arrows in a category then combinators can be viewed as operations for "combining" arrows to build new arrows. With these combinators, as with composition, the types of the functions being combined are relevant. For example, a pair of functions can be composed only if the domain of one is the same as the codomain of the other.

The kinds of combinators we are mainly interested in are those available in the Charity programming language. Charity is based on a categorical formulation of inductive and co-inductive datatypes. These datatypes provide primitive combinators from which Charity functions are constructed. Of particular interest is how combinators are used to construct *polymorphic* functions and the fact that all such functions satisfy parametricity.

Parametric polymorphism is a term that was introduced by Strachey describing those kinds of polymorphic functions having a uniformly given algorithm in all types [bfss90]. Much of the literature on parametricity has to do with the second-order polymorphic lambda calculus, $\lambda 2$, and with models of $\lambda 2$ in which polymorphic functions satisfy meaningful formalizations of Strachey's original idea. Since Charity is based on a categorical semantics, one of our goals here is to formulate a categorical notion of parametricity appropriate to Charity.

Initial comparison of $\lambda 2$ with Charity suggests that the parametric semantics of Charity should be somewhat simpler. Parametricity for $\lambda 2$ is complicated by the bivariance of the exponential type constructor and by the type variable binding associated with second-order polymorphism. In Charity all the type constructors are strictly covariant and the polymorphism is first-order.

As parametricity ought to be a property of all polymorphic Charity functions, and since all such functions are constructed from combinators, we expect parametricity to be a fundamental property of our combinators. The general form of our combinators is:

$$\frac{f : S_0(A) \to T_0(A)}{c\{f\} : S_1(A) \to T_1(A)}$$

where S_0, T_0, S_1, T_1 are (covariant) type constructors. Such a combinator can be regarded as a kind of algebraic operator. The natural formulation of parametricity for such an operator is then preservation of algebraic homomorphisms: for all $h : A \to B$, $f : S_0(A) \to T_0(A)$, $f' : S_0(B) \to T_0(B)$, it should be the case that

$$\begin{tikzcd} S_0(A) \arrow[r, "f"] \arrow[d, "S_0(h)"'] & T_0(A) \arrow[d, "T_0(h)"] \\ S_0(B) \arrow[r, "f'"'] & T_0(B) \end{tikzcd} \quad \Rightarrow \quad \begin{tikzcd} S_1(A) \arrow[r, "c\{f\}"] \arrow[d, "S_1(h)"'] & T_1(A) \arrow[d, "T_1(h)"] \\ S_1(B) \arrow[r, "c\{f'\}"'] & T_1(B) \end{tikzcd}$$

It is known that parametricity is related to the universal properties of inductive and co-inductive datatypes. In models of $\lambda 2$, for example, parametricity turns out to be necessary and sufficient for the initiality of weakly initial algebras and for the finality of weakly final co-algebras [has94]. For Charity there are similar statements relating parametricity to the uniqueness of folds and unfolds.

The descriptions of datatypes and combinators that we work with are at the 2-categorical level of generality. This follows previous work on the semantics of Charity ([d1]) where the general datatype constructions are given in terms of a 2-categorical abstraction of algebra categories, namely *inserters*. The general constructions are then moved into a specific 2-category making the datatypes *strong* (as they actually are in Charity). The result is a description of datatypes which is orthogonal to strength. We wish to have a similarly general notion of combinator. As a result we may examine the relationship between datatypes, combinators, and parametricity both at the 2-categorical level and in the presence of strength (and actually, in any 2-category where datatypes may be of interest).

In Chapter 2 we describe the basic syntax and semantics of the Charity language and consider the semantic issues of polymorphic Charity programs. This establishes a few intuitions against which our subsequent formalization of combinators can be evaluated.

In Chapter 3 we give a 2-categorical formulation of combinators based on inserters. These combinators satisfy a 2-categorical version of parametricity. An abstract theory of polymorphic combinators is also described, along with a fibrational semantics which arises in any 2-category with finite limits and inserters.

In Chapter 4 inductive and co-inductive datatypes are described at the 2-categorical level of generality. In particular, these datatypes are shown to be compatible with the combinators defined in Chapter 3. A discussion relating the universal properties with parametricity is included.

In Chapter 5 the 2-categorical concepts are instantiated to the case where the datatypes and combinators are strong. This leads to a definition of *strong combinator* which provides an appropriate formalization of combinators and parametricity in Charity.

The reader is assumed to have a basic knowledge of category theory (e.g. [bor94]) and fibrations (e.g. [jac94]). At a less technical level familiarity with functional programming (e.g. [bw88]), type theory (e.g. [tho91]), polymorphic lambda calculus (e.g. [rob92]), combinators in the lambda calculus (e.g. [hls72], [hin85]), parametricity (e.g. [rey83],[acc93],[bfss90]), and of course Charity (e.g. [cf92],[ft96]) are all useful.

Chapter 2

Charity

Charity is a programming language based on categorical datatypes. Informally, a datatype consists of a type constructor and some combinators for building functions representing data structures and programs. A category provides a formal framework in which types (corresponding to objects) and functions (corresponding to arrows) can be expressed. Datatypes can then be formalized as constructions within a category. The Charity language lifts these formal constructions into a programming-style syntax whose semantics is dictated by the underlying category theory.

The idea of modeling a programming language after categorical datatypes was first investigated by Hagino [hag87]. He provided a categorical formalization of datatypes in general and described a programming language based on this formalization. Charity has followed a similar line in its development (see [d1] and [d2], or [spe93]). The difference is that Charity is strictly first-order and the datatypes are *strong* and of two kinds: inductive and co-inductive.

Charity programs and data can be written as categorical combinator expressions. While it is possible to program at this level, practically it is rather awkward. For this reason a more convenient language called the *term logic* was introduced in [d2]. Semantically it is equivalent to the underlying category theory but syntactically it is easier to use. Recent implementations of a Charity interpreter (see [ft96]) are based on the term logic and have taken the categorical concepts to a usable software tool. Among the features provided by this interpreter is the ability to define polymorphic functions. Such functions can be parametrized by both types and functions. For example, in the presence of suitable datatypes one can define a polymorphic sorting function,

$$sort\{p : A \times A \to Bool\} : List(A) \to List(A),$$

where p is a function parameter and A is a type parameter.

Given the categorical semantics of non-polymorphic Charity functions (i.e. the original term logic) we wish to consider how polymorphic Charity functions can be interpreted categorically. The meaning of function parameters in a setting which is not higher order is of particular interest.

In this chapter we begin with an overview of the syntax and semantics of the original term logic. Our presentation differs from that in [d2] primarily in our use of type theory to express the grammar and type structure of the language. The semantics is then described in terms of the *simple fibration* (see [jac94]) which is a categorical structure appropriate to the type theory. Next strong datatypes are described as they appear in the type theory and how they are modeled in the simple fibration. Finally we consider polymorphic functions in Charity and their parametric properties. The emerging view will be to regard such functions as categorical combinators satisfying an appropriate form of parametricity.

2.1 The Term Logic

Charity programs and data can be expressed in terms of variable-free categorical combinators. While programming at this level is fine in theory, in practice it can be rather awkward as a plethora of projections is often needed to distribute context throughout a combinator expression. The term logic of [d2] simplifies the problem of context distribution by allowing the use of declared term variables in combinator expressions. In this section we give a type theoretic formulation of the term logic. The syntactic entities include types, terms, functions, and equalities.

Types

Assume $\mathcal{T}$ is a set of *types* which is closed to finite products:

- $1 \in \mathcal{T}$, and
- $S, T \in \mathcal{T}$ implies $(S \times T) \in \mathcal{T}$.

Additional structure can be imposed on $\mathcal{T}$ with the declaration of an inductive or co-inductive datatype (see 2.3).

Terms and Functions

Assume that for each ordered pair of types $S, T \in \mathcal{T}$, there is a set $\mathcal{F}[\mathcal{S}, \mathcal{T}]$ of *function symbols* with *domain* S and *codomain* T. The function symbols include *projections* $p_0 \in \mathcal{F}[\mathcal{S} \times \mathcal{T}, \mathcal{S}]$ and $p_1 \in \mathcal{F}[\mathcal{S} \times \mathcal{T}, \mathcal{T}]$, and *identity functions* $id \in \mathcal{F}[\mathcal{S}, \mathcal{S}]$ for each $S, T \in \mathcal{T}$.

The syntax of terms and functions is given by a collection of rules for deriving *judgements* of the form

$$\Gamma \vdash \varphi$$

where Γ is a *context* and φ is a term or function that is well-formed with respect to Γ. The general form of a context is

$$\Gamma = v_1 : S_1, \ldots, v_n : S_n$$

$(n \geq 0)$ where $v_1, \ldots, v_n$ are *variable bases* (see [d2]) containing individual term variables drawn from some countably infinite supply of symbols, and $S_1, \ldots, S_n$ are types. The rules for context formation are as follows:

- $\dfrac{}{\emptyset \text{ context}}$ *empty*

- $\dfrac{\Gamma \text{ context} \quad S \in \mathcal{T} \quad x \notin \Gamma}{\Gamma\,,\, x : S \text{ context}}$ *var intro*

- $\dfrac{\Gamma \text{ context}}{\Gamma\,,\, () : 1 \text{ context}}$ *unit vbase*

- $\dfrac{\Gamma\,,\, v : S\,,\, w : T \text{ context}}{\Gamma\,,\, (v, w) : (S \times T) \text{ context}}$ *pair vbase*

The two forms of judgement derivable in the type theory are

$$\Gamma \vdash t : T \quad \text{and} \quad \Gamma \vdash f : S \to T$$

which can be read as "t is a term of type T" and "f is a function with domain S and codomain T". In each case all the free term variables occurring in t or f must be listed in Γ.

The rules for deriving *term judgements* are as follows:

- $$\frac{\Gamma \text{ context} \qquad x : S \in \Gamma}{\Gamma \vdash x : S}\ \mathit{var}$$

- $$\frac{\Gamma \text{ context}}{\Gamma \vdash () : 1}\ \mathit{unit}$$

- $$\frac{\Gamma \vdash s : S \qquad \Gamma \vdash t : T}{\Gamma \vdash (s, t) : (S \times T)}\ \mathit{pair}$$

- $$\frac{\Gamma \vdash s : S \qquad \Gamma \vdash f : S \to T}{\Gamma \vdash f(s) : T}\ \mathit{apply}$$

In the *var* rule $x : S$ is an individual term variable which occurs in some variable base in Γ. In the *apply* rule redundant parentheses are generally omitted. For example, we write $f(x, y)$ instead of $f((x, y))$.

The rules for deriving *function judgements* are as follows:

- $$\frac{\Gamma \text{ context} \qquad f \in \mathcal{F}[\mathcal{S}, \mathcal{T}]}{\Gamma \vdash f : S \to T}\ \mathit{fn\ sym}$$

- $$\frac{\Gamma, v : S \vdash t : T}{\Gamma \vdash \{v \mapsto t\} : S \to T}\ \mathit{bind}$$

- $$\frac{\Gamma \text{ context} \qquad S, T \in \mathcal{T}}{\Gamma \vdash p_0 : (S \times T) \to S}\ p_0$$

- $$\frac{\Gamma \text{ context} \qquad S, T \in \mathcal{T}}{\Gamma \vdash p_1 : (S \times T) \to T}\ p_1$$

- $$\frac{\Gamma \text{ context} \qquad S \in \mathcal{T}}{\Gamma \vdash \mathit{id} : S \to S}\ \mathit{id}$$

Of interest are functions of the form $\{v \mapsto t\}$ constructed using the *bind* rule. These functions are called *abstracted terms* and resemble lambda calculus expressions of the form $\lambda v.t$ in the sense that all the term variables occurring in v are *bound* within $\{v \mapsto t\}$. Any variables in t which are not in v (and are not bound deeper in t) are called *free* in $\{v \mapsto t\}$. The definitions of free and bound variables can be extended to arbitrary terms and functions.

The role of variable bases in abstracted terms is to provide a simple form of pattern matching of arguments to a function. Like beta reduction in the lambda calculus we have $\{v \mapsto t\}(s) = t[s/v]$ in the term logic where $t[s/v]$ is

t with s substituted for (i.e. pattern-matched against) v. For example, a term of the form $\{(x, y) \mapsto f(x, g(y))\}(t)$ can be evaluated to $f(p_0(t), g(p_1(t)))$. Pattern matching is also used in the following general substitution rule:

- $\dfrac{v_1 : S_1, \ldots, v_n : S_n \vdash \varphi \qquad \Gamma \vdash s_1 : S_1, \ldots, s_n : S_n}{\Gamma \vdash \varphi[s_1/v_1, \ldots, s_n/v_n]}\ subst$

where $\varphi[s_1/v_1, \ldots, s_n/v_n]$ means simultaneously substitute terms $s_1, \ldots, s_n$ for variable bases $v_1, \ldots, v_n$ in φ while avoiding the capture of free variables. For a more formal definition of pattern-matched substitution see [d2].

Equalities

Given a pair of term judgements $\Gamma \vdash s : S$ and $\Gamma \vdash s' : S$ we write

$$\Gamma \vdash s = s' : S$$

when s and s' are equal terms with respect to Γ. The rules are as follows:

- $\dfrac{\Gamma \vdash s : S}{\Gamma \vdash s = s : S}\ \mathit{refl}$

- $\dfrac{\Gamma \vdash s = s' : S}{\Gamma \vdash s' = s : S}\ \mathit{symm}$

- $\dfrac{\Gamma \vdash s = s' : S \qquad \Gamma \vdash s' = s'' : S}{\Gamma \vdash s = s'' : S}\ \mathit{trans}$

- $\dfrac{\Gamma \vdash t : 1}{\Gamma \vdash t = () : 1}\ \mathit{unit}$

- $\dfrac{\Gamma \vdash s : S \qquad \Gamma \vdash t : T}{\Gamma \vdash p_0(s, t) = s : S}\ \mathit{fst\ proj}$

- $\dfrac{\Gamma \vdash s : S \qquad \Gamma \vdash t : T}{\Gamma \vdash p_1(s, t) = t : T}\ \mathit{snd\ proj}$

- $\dfrac{\Gamma \vdash r : (S \times T)}{\Gamma \vdash (p_0(r), p_1(r)) = r : (S \times T)}\ \mathit{surj\ pair}$

- $\dfrac{\Gamma \vdash s : S}{\Gamma \vdash id(s) = s : S}\ \mathit{identity}$

- $\dfrac{\Gamma,\, v : S \vdash t : T \qquad \Gamma \vdash s : S}{\Gamma \vdash \{v \mapsto t\}(s) = t[s/v] : T}$ *applied abs*

- $\dfrac{\Gamma,\, v : S \vdash t = t' : T \qquad \Gamma \vdash s = s' : S}{\Gamma \vdash \{v \mapsto t\}(s) = \{v \mapsto t'\}(s') : T}$ *congr*

Given a pair of function judgements $\Gamma \vdash f : S \to T$ and $\Gamma \vdash f' : S \to T$ write

$$\Gamma \vdash f = f' : S \to T$$

when the term equality

$$\Gamma \vdash f(s) = f'(s) : T$$

holds for every term judgement $\Gamma \vdash s : S$. As a consequence, abstracted terms which differ only in their bound variables are considered equal. This property is useful when performing substitution as bound variables may be renamed to avoid the capture of free variables. For example, the substitution $\{x \mapsto f(x,y)\}[g(x)/y]$ can be correctly evaluated to $\{x' \mapsto f(x', g(x))\}$ if x' does not occur freely in g.

2.2 Semantics

In [d2] the semantics of the term logic is expressed by a pair of inverse, equality-preserving translations between closed abstractions $\{v \mapsto t\}$ and categorical combinator expressions. This establishes a correspondence between the term logic and a category with finite products: objects in the category correspond to types and arrows correspond to closed abstractions. In this section we abstract away from closed abstractions to obtain a categorical interpretation of arbitrary term and function judgements.

Suppose $\mathbf{X}$ is a category with finite products. For objects this means:

- $1 \in \mathbf{X}$ (i.e. the final object), and

- $X, Y \in \mathbf{X}$ implies $(X \times Y) \in \mathbf{X}$ (i.e. binary products).

Arrows in $\mathbf{X}$ must satisfy the following formation rules:

- $\dfrac{X \in \mathbf{X}}{id : X \to X}$

- $\dfrac{f : X \to Y \qquad g : Y \to Z}{f; g : X \to Z}$

- $\dfrac{X \in \mathbf{X}}{! : X \to 1}$

- $\dfrac{X, Y \in \mathbf{X}}{p_0 : (X \times Y) \to X}$

- $\dfrac{X, Y \in \mathbf{X}}{p_1 : (X \times Y) \to Y}$

- $\dfrac{f : X \to Y \qquad g : X \to Z}{\langle f, g\rangle : X \to (Y \times Z)}$

Arrows in $\mathbf{X}$ must also satisfy the following identities:

- $id; f = f = f; id$
- $(f; g); h = f; (g; h)$
- $f; ! =!$
- $\langle f, g\rangle\, ; p_0 = f$
- $\langle f, g\rangle\, ; p_1 = g$
- $\langle p_0, p_1\rangle = id.$

The interpretation of the type theory begins with an assignment of types $S \in \mathcal{T}$ to objects $S \in \mathbf{X}$ (i.e. objects in $\mathbf{X}$ correspond to types in $\mathcal{T}$) and an assignment of function symbols $f \in \mathcal{F}[\mathcal{S}, \mathcal{T}]$ to arrows $f : S \to T$. Generalizing the [d2] translation, term and function judgements $\Gamma \vdash \varphi$ are interpreted as arrows $[\![\Gamma \vdash \varphi]\!]$ in $\mathbf{X}$. If $\Gamma = v_1 : S_1, \ldots, v_n : S_n$ and if V is the type expression

$$(\cdots((S_1 \times S_2) \times S_3) \cdots \times S_n)$$

(V is 1 when $n = 0$) then the types of the arrows produced by the translation are:

$$\begin{array}{l}[\![\Gamma \vdash t : T]\!] : V \to T \\ [\![\Gamma \vdash f : S \to T]\!] : V \times S \to T\end{array}$$

The translation is given by the following equations which can be read as left-to-right rewrite rules:

- $[\![\Gamma \vdash () : 1]\!] = !$,
- $[\![\Gamma \vdash (s,t) : (S \times T)]\!] = \langle [\![\Gamma \vdash s : S]\!] , [\![\Gamma \vdash t : T]\!] \rangle$,
- $[\![\Gamma \vdash f(s) : T]\!] = \langle id, [\![\Gamma \vdash s : S]\!] \rangle ; [\![\Gamma \vdash f : S \to T]\!]$,
- if x is a term variable then
  - $[\![x : T \vdash x : T]\!] = id$,
  - $[\![\Gamma \, , \, v : S \vdash x : T]\!] = p_0; [\![\Gamma \vdash x : T]\!]$ if $x \in \Gamma$,
  - $[\![\Gamma \, , \, v : S \vdash x : T]\!] = p_1; [\![v : S \vdash x : T]\!]$ if $x \notin \Gamma$,
  - $[\![(v,v') : (S \times S') \vdash x : T]\!] = [\![v : S \, , \, v' : S' \vdash x : T]\!]$,
- if $f \in \mathcal{F}[\mathcal{S}, \mathcal{T}]$ then $[\![\Gamma \vdash f : S \to T]\!] = p_1; f$,
- $[\![\Gamma \vdash \{v \mapsto t\} : S \to T]\!] = [\![\Gamma \, , \, v : S \vdash t : T]\!]$ assuming no variables in v also occur in Γ (any such variables must first be renamed in $\{v \mapsto t\}$).

Given that $\mathbf{X}$ has finite products we can form the *simple fibration over* $\mathbf{X}$ (see also [jac94]) given by the category $s(\mathbf{X})$ in which

- objects are pairs (V,X) where $V,X \in \mathbf{X}$,
- arrows $(u,f) : (U,X) \to (V,Y)$ are pairs of arrows $u : U \to V$ and $f : U \times X \to Y$,
- composition is given by $(u,f);(v,g) = (u;v, \langle p_0;u,f \rangle ; g)$, and
- identities are (id, p_1).

The category $s(\mathbf{X})$ is fibred over $\mathbf{X}$ by the functor $\delta : s(\mathbf{X}) \to \mathbf{X}$ given by $(u,f) \mapsto u$. The cartesian arrows in this (split) fibration are $(u,p_1) : (U,X) \to (V,X)$ for each $(V,X) \in s(\mathbf{X})$ and $u : U \to V$ in $\mathbf{X}$. These cartesian arrows induce for each $u : U \to V$ in $\mathbf{X}$ a *reindexing* functor $u^* : \delta^{-1}(V) \to \delta^{-1}(U)$ given by

$$\begin{array}{ccc} (V,X) & & (U,X) \\ {\scriptstyle (id,f)}\downarrow & \mapsto & \downarrow{\scriptstyle (id,(u \times id);f)} \\ (V,Y) & & (U,Y) \end{array}$$

The simple fibration provides a categorical structure in which certain aspects of the type theory can be interpreted nicely. If term and function judgements $v : V \vdash \varphi$ are viewed as vertical arrows in $\delta^{-1}(V)$ then substitution can be expressed in terms of reindexing functors. For example, each substitution of the form

$$\frac{v : V \vdash \varphi \qquad \Gamma \vdash s : V}{\Gamma \vdash \varphi[s/v]}$$

satisfies

$$[\![\Gamma \vdash \varphi[s/v]]\!] = [\![\Gamma \vdash s : V]\!]^* \left([\![v : V \vdash \varphi]\!]\right).$$

The simple fibration can also be used to describe strong datatypes and strong combinators.

2.3 Strong Datatypes

On its own the term logic is rather unexpressive as a programming language. This is remedied by assuming the presence of various inductive and co-inductive datatypes which deliver additional types, functions, and equalities to the theory. In Charity these datatypes are assumed to be *strong*. In this section we show how strong datatypes are expressed in the type theory and in terms of the simple fibration.

The general form of an inductive datatype declaration in Charity is

$$\begin{array}{rcl} \text{data } L(A) \to C & = & c_1 : E_1(A, C) \to C \\ & | & \cdots \\ & | & c_n : E_n(A, C) \to C \end{array}$$

where A, C are $k+1$ distinct type variables (think of A as $A_1, \ldots, A_k$) for some $k \geq 0$ and each $E_i(A, C)$ is a type which may contain occurrences of A and C. Such a declaration introduces the following:

- a new *type constructor* L of arity k,
- new function symbols $c_i : E_i(A, L(A)) \to L(A)$ for each $A \in \mathcal{T}^k$ and $1 \leq i \leq n$ called *term constructors*, and
- *fold*, *case*, and *map* combinators.

Type theoretically the term constructors and combinators are given by the following rules:

- $A \in \mathcal{T}^k$ implies $L(A) \in \mathcal{T}$
- $\dfrac{\Gamma \text{ context} \qquad A \in \mathcal{T}^k}{\Gamma \vdash c_i : E_i(A, L(A)) \to L(A)}\ c_i\ (1 \leq i \leq n)$
- $$\frac{\Gamma \vdash f_1 : E_1(S,T) \to T \quad \cdots \quad \Gamma \vdash f_n : E_n(S,T) \to T}{\Gamma \vdash \left\{\!\left| \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right|\!\right\} : L(S) \to T}\ \mathit{fold}$$
- $$\frac{\Gamma \vdash f_1 : E_1(S,L(S)) \to T \quad \cdots \quad \Gamma \vdash f_n : E_n(S,L(S)) \to T}{\Gamma \vdash \left\{ \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right\} : L(S) \to T}\ \mathit{case}$$
- $\dfrac{\Gamma \vdash f_1 : S_1 \to T_1 \quad \cdots \quad \Gamma \vdash f_k : S_k \to T_k}{\Gamma \vdash L\{f_1 \cdots f_k\} : L(S_1 \cdots S_k) \to L(T_1 \cdots T_k)}\ \mathit{map}$

The equalities associated with an inductive datatype are as follows:

- *fold unrolling*
$$\forall_{i=1}^{n}. \left\{\!\left| \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right|\!\right\} (c_i(x_i)) = f_i(E_i\{id, \left\{\!\left| \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right|\!\right\}\}(x_i))$$
- *case unrolling*
$$\forall_{i=1}^{n}. \left\{ \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right\} (c_i(x_i)) = f_i(x_i)$$
- *map unrolling*
$$\forall_{i=1}^{n}. L\{f\}(c_i(x_i)) = c_i(E_i\{f, L\{f\}\}(x_i))$$
- *fold uniqueness*: if $h : L(A) \to C$ satisfies
$$h(c_i(x_i)) = f_i(E_i\{id, h\}(x_i))$$
for each $1 \leq i \leq n$ then
$$h = \left\{\!\left| \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right|\!\right\}$$

The above *unrolling* rules can be read as left-to-right rewrite rules for evaluating the functions when they are applied to L data structures. In light of the *fold uniqueness* rule it is possible to express *case* and *map* functions in terms of *fold*. The *case* and *map* combinators are provided explicitly in the term logic for convenience as well as efficiency in evaluation (see [d2]).

In order to extend the interpretation of the type theory with a datatype, L, requires additional categorical structure in $\mathbf{X}$. Say a category $\mathbf{C}$ *has datatype* L when $E_1, \ldots, E_n$ are functors $\mathbf{C}^{k+1} \to \mathbf{C}$, $L : \mathbf{C}^k \to \mathbf{C}$ is a functor, $c_1, \ldots, c_n$ are natural transformations of the form

$$c_i : E_i(A, L(A)) \to L(A)$$

(natural in A) for each $1 \leq i \leq n$, and for each n-tuple of arrows

$$f_1 : E_1(A, C) \to C, \ldots, f_n : E_n(A, C) \to C$$

there is a unique arrow $\mathit{fold}^L\{f_1 \cdots f_n\} : L(A) \to C$ such that

$$\begin{tikzcd}
E_i(A, L(A)) \arrow[r, "c_i"] \arrow[d, "E_i(id, \mathit{fold}^L\{f_1 \cdots f_n\})"'] & L(A) \arrow[d, "\mathit{fold}^L\{f_1 \cdots f_n\}"] \\
E_i(A, C) \arrow[r, "f_i"'] & C
\end{tikzcd}$$

commutes for each $1 \leq i \leq n$.

Asserting that L is a *strong* datatype involves reformulating the above definition for *strong* functors $E_1, \ldots, E_n$ and L (see [d1]). Using the known correspondence between strong functors and morphisms of fibrations an equivalent formulation can be expressed in terms of the simple fibration (see 2.6 in [jac94]). Given strong functors $E_1, \ldots, E_n : \mathbf{X}^{k+1} \to \mathbf{X}$ say $\mathbf{X}$ *has a strong datatype* L when $L : \mathbf{X}^k \to \mathbf{X}$ is a strong functor and for each $V \in \mathbf{X}$ the fibre category $\delta^{-1}(V)$ has datatype L and reindexing functors preserve these datatypes on the nose.

The type theory interpretation is extended as follows:

- $[\![\Gamma \vdash c_i]\!] = p_1; c_i$ for each $1 \leq i \leq n$
- $\left[\!\!\left[\Gamma \vdash \left\{ \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right\}\right]\!\!\right] = \mathit{fold}^L\{[\![\Gamma \vdash f_1]\!] \cdots [\![\Gamma \vdash f_n]\!]\}$

- $\left[\!\!\left[\Gamma \vdash \left\{ \begin{array}{c} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right\} \right]\!\!\right] = case^L\{[\![\Gamma \vdash f_1]\!] \cdots [\![\Gamma \vdash f_n]\!]\}$
- $[\![\Gamma \vdash L\{f_1, \ldots, f_k\}]\!] = map^L\{[\![\Gamma \vdash f_1]\!] \cdots [\![\Gamma \vdash f_n]\!]\}$

where $case^L$ and map^L denote the appropriate derived combinators based on fold uniqueness (see [d1]).

Co-inductive datatypes are the categorical dual of inductive datatypes. The general form of such a declaration in Charity is

$$\begin{array}{rcl} \text{data } C \to R(A) & = & d_1 : C \to E_1(A, C) \\ & | & \cdots \\ & | & d_n : C \to E_n(A, C) \end{array}$$

where R is a new type constructor and $d_1, \ldots, d_n$ are new function symbols called *term destructors*. Also delivered with such a datatype is a *unfold* combinator (dual to *fold*) as well as *record* and *map* combinators (analogous to *case* and *map*). The type theory rules are as follows:

- $A \in \mathcal{T}^k$ implies $R(A) \in \mathcal{T}$
- $\dfrac{\Gamma \text{ context} \qquad A \in \mathcal{T}^k}{\Gamma \vdash d_i : R(A) \to E_i(A, R(A))}\ d_i\ (1 \leq i \leq n)$
- $\dfrac{\Gamma \vdash g_1 : T \to E_1(S, T) \quad \cdots \quad \Gamma \vdash g_n : T \to E_n(S, T)}{\Gamma \vdash \left(\left| \begin{array}{c} d_1 : g_1 \\ \cdots \\ d_n : g_n \end{array} \right|\right) : T \to R(S)}\ unfold$
- $\dfrac{\Gamma \vdash t_1 : E_1(S, R(S)) \quad \cdots \quad \Gamma \vdash t_n : E_n(S, R(S))}{\Gamma \vdash (d_1 : t_1, \ldots, d_n : t_n) : R(S)}\ record$
- $\dfrac{\Gamma \vdash f_1 : S_1 \to T_1 \quad \cdots \quad \Gamma \vdash f_k : S_k \to T_k}{\Gamma \vdash R\{f_1 \cdots f_k\} : R(S_1 \cdots S_k) \to R(T_1 \cdots T_k)}\ map$

The equality rules are:

- *unfold unrolling*

$$\forall_{i=1}^{n}.d_i \left(\left| \begin{array}{c} d_1 : g_1 \\ \cdots \\ d_n : g_n \end{array} \right|\right)(x) = E_i\{id, \left(\left| \begin{array}{c} d_1 : g_1 \\ \cdots \\ d_n : g_n \end{array} \right|\right)\}(g_i(x))$$

- *record destruction*

$$\forall_{i=1}^{n}.d_i(d_1 : t_1, \ldots, d_n : t_n) = t_i$$

- *map unrolling*

$$\forall_{i=1}^{n}.d_i(R\{f\}(x)) = E_i\{f, R\{f\}\}(d_i(x))$$

- *unfold uniqueness*: if $k : C \to R(A)$ satisfies

$$d_i(k(x)) = E_i\{id, k\}(g_i(x))$$

  for each $1 \leq i \leq n$ then

$$k = \left(\left| \begin{array}{c} d_1 : g_1 \\ \cdots \\ d_n : g_n \end{array} \right|\right)$$

As for inductive datatypes the uniqueness rule permits expression of records and maps as unfolds. As for the *case* and *map* of an inductive datatype these combinators are provided for convenience and efficiency.

The categorical formulation for a strong co-inductive datatype parallels that for a strong inductive datatype. This time the universal diagram looks like this:

$$\begin{tikzcd} C \arrow[r, "g_i"] \arrow[d, "unfold^R\{g_1 \cdots g_n\}"'] & E_i(A, C) \arrow[d, "E_i(id, unfold^R\{g_1 \cdots g_n\})"] \\ R(A) \arrow[r, "d_i"'] & E_i(A, R(A)) \end{tikzcd}$$

and the presence of a strong such datatype in $\mathbf{X}$ means each fibre has R as a datatype and these datatypes are preserved on the nose by reindexing functors. The type theory interpretation is extended as follows:

- $[\![\Gamma \vdash d_i]\!] = p_1; d_i$ for each $1 \leq i \leq n$
- $\left[\!\!\left[\Gamma \vdash \left(\left| \begin{array}{c} d_1 : g_1 \\ \cdots \\ d_n : g_n \end{array} \right|\right)\right]\!\!\right] = unfold^R\{[\![\Gamma \vdash g_1]\!] \cdots [\![\Gamma \vdash g_n]\!]\}$

- $[\![\Gamma \vdash (d_1 : t_1, \ldots, d_n : t_n)]\!] = \mathit{record}^R\{[\![\Gamma \vdash t_1]\!] \cdots [\![\Gamma \vdash t_n]\!]\}$
- $[\![\Gamma \vdash R\{f_1, \ldots, f_k\}]\!] = \mathit{map}^R\{[\![\Gamma \vdash f_1]\!] \cdots [\![\Gamma \vdash f_n]\!]\}$

where record^R and map^R denote the appropriate combinators derived from the combinator unfold^R.

2.4 Strong Combinators

The declaration of datatypes in Charity introduces combinators which can be used to construct interesting functions and terms. Since these datatypes are strong, the combinators are fibred. For example, the fold combinator whose application rule is

$$\frac{\Gamma \vdash f_1 : E_1(A, C) \to C \quad \cdots \quad \Gamma \vdash f_n : E_n(A, C) \to C}{\Gamma \vdash \left\{ \begin{array}{|c|} c_1 : f_1 \\ \cdots \\ c_n : f_n \end{array} \right\} : L(A) \to C}$$

can be expressed categorically as a fibred operation for constructing arrows: for each $V \in \mathbf{X}$ and $f_1 : E_1(A, C) \to C, \ldots, f_n : E_n(A, C) \to C$ in $\delta^{-1}(V)$ there is an arrow $\mathit{fold}^L\{f_1 \cdots f_n\} : L(A) \to C$ in $\delta^{-1}(V)$ such that for each $u : U \to V$ in $\mathbf{X}$,

$$u^*(\mathit{fold}^L\{f_1 \cdots f_n\}) = \mathit{fold}^L\{u^*(f_1) \cdots u^*(f_n)\}.$$

The fold combinator can be seen as an instance of the following general form of combinator

$$\frac{\Gamma \vdash f_1 : S_1(A) \to T_1(A) \quad \cdots \quad \Gamma \vdash f_n : S_n(A) \to T_n(A)}{\Gamma \vdash c\{f_1 \cdots f_n\} : S_0(A) \to T_0(A)}$$

where for each $0 \leq i \leq n$, S_i and T_i are strong type constructing functors of common arity $|A|$. The fact that such a combinator is preserved by reindexing can be expressed in terms of substitution in the type theory: given judgements

$$v : V \vdash f_1 : S_1(A) \to T_1(A) \ \cdots \ v : V \vdash f_n : S_n(A) \to T_n(A)$$

and

$$\Gamma \vdash s : V$$

then the following syntactic equivalence holds:

$$\Gamma \vdash c\{f_1 \cdots f_n\}[s/v] \quad \equiv \quad \Gamma \vdash c\{f_1[s/v] \cdots f_n[s/v]\}$$

Since such fibred combinators depend on strong functors, we shall call them *strong combinators* from now on.

Assuming the presence of datatypes in the type theory, it is possible to derive many interesting examples of strong combinators. For example, consider the *filter* function on finite lists:

$$\frac{\Gamma \vdash p : A \to \mathit{Bool}}{\Gamma \vdash \mathit{filter}\{p\} : \mathit{List}(A) \to \mathit{List}(A)}$$

where $\mathit{filter}\{p\}$ is defined as

$$\mathit{filter}\{p\} \equiv \left\{\!\!\left| \begin{array}{ll} \mathit{nil}: & \mathit{nil} \\ \mathit{cons}: & (a,l) \mapsto \left\{ \begin{array}{ll} \mathit{false}: & () \mapsto l \\ \mathit{true}: & () \mapsto \mathit{cons}(a,l) \end{array} \right\} p(a) \end{array} \right|\!\!\right\}.$$

A function such as *filter* is called *polymorphic* because it is parametrized by a choice of type for A and a choice of function for p. In this sense combinators in general can be viewed as polymorphic functions.

Another example of a strong combinator is the polymorphic *append* function,

$$\mathit{append} \equiv \{(x,y) \mapsto \left\{\!\!\left| \begin{array}{ll} \mathit{nil}: & () \mapsto y \\ \mathit{cons}: & \mathit{cons} \end{array} \right|\!\!\right\} (x)\}$$

which can be expressed type theoretically as

$$\frac{\Gamma \text{ context} \qquad A \in \mathcal{T}}{\Gamma \vdash \mathit{append} : (\mathit{List}(A) \times \mathit{List}(A)) \to \mathit{List}(A)}$$

Like *filter*, this function is polymorphic in the sense that it depends on a type parameter A. Unlike *filter*, however, *append* does not depend on a function parameter.

Intuitively, polymorphic functions like *filter* and *append* can be viewed as families of functions exhibiting some sense of type independent behavior. Since Charity has no means for specifying type dependent behavior, we expect a similar statement about any polymorphic Charity function. By viewing combinators as algebraic operators, a natural way to model this behavioral invariance is by preservation of algebraic homomorphisms. Following the literature, we call this invariance *parametricity* and state it as follows for strong combinators. Suppose c is a strong combinator given by the rule

$$\frac{\Gamma \vdash f_1 : S_1(A) \to T_1(A) \quad \cdots \quad \Gamma \vdash f_n : S_n(A) \to T_n(A)}{\Gamma \vdash c\{f_1 \cdots f_n\} : S_0(A) \to T_0(A)}$$

and suppose $\Gamma \vdash c\{f_1 \cdots f_n\} : S_0(Y) \to T_0(Y)$, $\Gamma \vdash c\{g_1 \cdots g_n\} : S_0(Z) \to T_0(Z)$, and $\Gamma \vdash h : Y \to Z$ are judgements. If, for all $1 \leq i \leq n$,

$$\Gamma \vdash f_i; T_i\{h\} = S_i\{h\}; g_i$$

then

$$\Gamma \vdash c\{f_1 \cdots f_n\}; T_0\{h\} = S_0\{h\}; c\{g_1 \cdots g_n\}$$

where these equations are between arrows in the fibre category "$\delta^{-1}(\Gamma)$". Diagrammatically we have the following (in "$\delta^{-1}(\Gamma)$"):

$$\begin{tikzcd} S_i(A) \arrow[r, "f_i"] \arrow[d, "S_i\{h\}"'] \arrow[dr, phantom, "\forall_{i=1}^{n}"] & T_i(A) \arrow[d, "T_i\{h\}"] \\ S_i(B) \arrow[r, "g_i"'] & T_i(B) \end{tikzcd}$$

$$\Downarrow$$

$$\begin{tikzcd} S_0(A) \arrow[r, "c\{f_1 \cdots f_n\}"] \arrow[d, "S_0\{h\}"'] & T_0(A) \arrow[d, "T_0\{h\}"] \\ S_0(B) \arrow[r, "c\{g_1 \cdots g_n\}"'] & T_0(B) \end{tikzcd}$$

The main point of this section has been to motivate the view of polymorphic functions in Charity as strong combinators. Strong combinators can be regarded as fibred operations for constructing arrows. Furthermore strong combinators satisfy a fibred notion of parametricity. In the next chapter we define a more abstract notion of combinator satisfying an abstract notion of parametricity. Later (in Chapter 5) we verify that strong combinators are an instance of this abstraction.

Chapter 3

2-categorical Combinators

In the previous chapter, combinators have been presented as operations for constructing arrows in a category. Such operations can be viewed more formally as functors between categories of algebras. The object map of such a functor represents a construction of arrows and the arrow map represents parametricity as the preservation of algebraic homomorphisms.

Combinators can be expressed 2-categorically using *inserters* which are an abstraction of categories of algebras. In the first part of this chapter (3.2 and 3.3) we describe inserters and use them to define combinators. In the process we formulate a 2-categorical view of parametricity. By working at this level of generality we isolate the essential properties of combinators in any 2-category. In Chapter 5 the 2-categorical notions are instantiated to a particular 2-category to obtain *strong combinators* as they appear in Charity.

In the second part of this chapter (3.4 and 3.5), we develop a 2-categorical theory of combinators in which "polymorphic" combinator expressions (depending on type parameters and function parameters) are interpreted as 2-categorical combinators. Such a theory could be specialized to a theory of strong combinators in order to potentially obtain a polymorphic Charity type theory.

3.1 Preliminaries

The purpose of this section is to establish our 2-categorical notational conventions. For the definition of a 2-category see [bor94] or [ks74].

Suppose $\mathbb{C}$ is a 2-category and $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \in \mathbb{C}$ are 0-cells. Composition of

1-cells $F : \mathbf{X} \to \mathbf{Y}$ and $G : \mathbf{Y} \to \mathbf{Z}$ is written either as $F;; G : \mathbf{X} \to \mathbf{Z}$ or $GF : \mathbf{X} \to \mathbf{Z}$. Identity 1-cells are written $Id : \mathbf{X} \to \mathbf{X}$. We write $\mathbb{C}[\mathbf{X}, \mathbf{Y}]$ for the hom-category in which

- objects are 1-cells $F : \mathbf{X} \to \mathbf{Y}$,
- arrows are 2-cells $\alpha : F \Rightarrow F'$,
- composition of 2-cells $\alpha : F \Rightarrow F'$ and $\alpha' : F' \Rightarrow F''$ is written $\alpha; \alpha'$ and is called vertical composition, and
- identity 2-cells are written interchangeably as F or $id : F \Rightarrow F$.

Horizontal composition of 2-cells $\alpha : F \Rightarrow F'$ and $\beta : G \Rightarrow G'$ is written $\alpha;; \beta : (F;; G) \Rightarrow (F';; G')$ or $\beta\alpha : GF \Rightarrow G'F'$.

The standard example of a 2-category is **Cat** in which 0-cells are (small, locally small) categories, 1-cells are functors, and 2-cells are natural transformations. For now we shall mostly work in an arbitrary 2-category $\mathbb{C}$ with finite limits. For concreteness we shall occasionally exemplify 2-categorical concepts in **Cat**.

3.2 Inserters

An inserter is an example of a 2-categorical indexed limit [kel89], or a weighted limit. We begin with the general definition and then describe inserters in **Cat**, where they are shown to be an abstraction of categories of algebras. In the next section we use inserters to describe 2-categorical parametric combinators.

Definition 3.2.1 *Given 1-cells $F, G : \mathbf{Y} \to \mathbf{Z}$ an* **inserter** *of F and G is a triple $(\mathbf{I}, U, \alpha)$ where*

- $\mathbf{I}$ *is a 0-cell,*
- $U : \mathbf{I} \to \mathbf{Y}$ *is a 1-cell, and*
- $\alpha : U;; F \Rightarrow U;; G$ *is a 2-cell*

satisfying the following universal properties:

(1D) Given any $(\mathbf{J}, V, \beta)$ *where* $\mathbf{J}$ *is a 0-cell,* $V : \mathbf{J} \to \mathbf{Y}$ *is a 1-cell, and* $\beta : V;; F \Rightarrow V;; G$ *is a 2-cell, there is a unique 1-cell* $W : \mathbf{J} \to \mathbf{I}$ *such that* $W;; U = V$ *and* $W;; \alpha = \beta$*. The picture is:*

$$
\begin{tikzcd}
\mathbf{J} \arrow[r, dashed, "W"] \arrow[dr, "V"'] & \mathbf{I} \arrow[d, "U"] \\
& \mathbf{Y} \arrow[d, bend right, "F"'] \arrow[d, bend left, "G"] \arrow[d, phantom, "\Rightarrow"] \\
& \mathbf{Z}
\end{tikzcd}
$$

(2D) Given 1-cells $W_1, W_2 : \mathbf{J} \to \mathbf{I}$ *and a 2-cell* $\gamma : W_1;; U \Rightarrow W_2;; U$ *satisfying* $(\gamma;; F); (W_2;; \alpha) = (W_1;; \alpha); (\gamma;; G)$*, there is a unique 2-cell* $\delta : W_1 \Rightarrow W_2$ *such that* $\delta;; U = \gamma$*.*

An inserter is a generalization of an equalizer in that equalization on the nose is replaced with a 2-cell as in the 1-dimensional property, (1D). In addition, being a weighted limit means there is a 2-dimensional property, (2D), formalizing the idea that cones related by a 2-cell have comparison maps related by a 2-cell. An immediate consequence of the (2D) property of an inserter $(\mathbf{I}, U, \alpha)$ is that U is always faithful: given 2-cells $\delta, \delta' : W_1 \Rightarrow W_2$ such that $\delta;; U = \delta';; U$ it follows that $\delta = \delta'$.

It is easy to verify that inserters are unique up to isomorphism. In particular, if both $(\mathbf{I}, U, \alpha)$ and $(\mathbf{J}, V, \beta)$ are inserters of some F, G then $\mathbf{I} \cong \mathbf{J}$ (i.e. $\mathbf{I}$ is isomorphic to $\mathbf{J}$). For the most part, when working in a particular 2-category we shall work with a particular choice of inserters. We shall usually write $F//G$ for (the 0-cell of) *the* inserter of F, G.

An example of a 2-category that has inserters is **Cat**. Suppose $\mathbf{Y}$ and $\mathbf{Z}$ are 0-cells in **Cat**. Given functors $F, G : \mathbf{Y} \to \mathbf{Z}$, let us define the category $F//G$ as follows:

- objects are pairs (Y, f) where $Y \in \mathbf{Y}$ and $f : F(Y) \to G(Y)$ in $\mathbf{Z}$,

- arrows from (Y, f) to (Y', f') are arrows $h : Y \to Y'$ in $\mathbf{Y}$ such that

$$\begin{tikzcd} F(Y) \arrow[r, "f"] \arrow[d, "F(h)"'] & G(Y) \arrow[d, "G(h)"] \\ F(Y') \arrow[r, "f'"'] & G(Y') \end{tikzcd}$$

  commutes,

- composition and identities are taken from $\mathbf{Y}$.

Let $U : F//G \to \mathbf{Y}$ be the functor given by

$$[(Y, f) \xrightarrow{h} (Y', f')] \ \mapsto \ [Y \xrightarrow{h} Y'].$$

Notice that U is faithful. Let $\alpha : U;; F \Rightarrow U;; G$ be the natural transformation given by

$$\alpha_{(Y,f)} = f : F(Y) \to G(Y)$$

for each $(Y, f) \in F//G$. Naturality of α is immediate since every arrow in $F//G$ corresponds to a naturality square.

Theorem 3.2.2 $(F//G, U, \alpha)$ *as defined above is an inserter of* F, G *in* $\mathbf{Cat}$.

Proof. In order to show the (1D) property, suppose $V : \mathbf{J} \to \mathbf{Y}$ is a functor and $\beta : V;; F \Rightarrow V;; G$ is a natural transformation. The unique functor $V' : \mathbf{J} \to F//G$ such that $V';; U = V$ and $V';; \alpha = \beta$ is given by

$$[J \xrightarrow{k} J'] \ \mapsto \ [(V(J), \beta_J) \xrightarrow{V(k)} (V(J'), \beta_{J'})].$$

To verify uniqueness suppose $V'' : \mathbf{J} \to F//G$ is another functor such that $V'';; U = V$ and $V'';; \alpha = \beta$. The first equation implies V'' is of the form

$$[J \xrightarrow{k} J'] \ \mapsto \ [(V(J), \alpha_{V''(J)}) \xrightarrow{V(k)} (V(J'), \alpha_{V''(J')})]$$

and the second equation implies $\alpha_{V''(J)} = \beta_J$ so $V'' = V'$.

In order to show the (2D) property, suppose $W_1, W_2 : \mathbf{J} \to F//G$ are functors and $\gamma : W_1;; U \Rightarrow W_2;; U$ is a natural transformation such that

$$F(\gamma_J); \alpha_{W_2(J)} = \alpha_{W_1(J)}; G(\gamma_J)$$

for each $J \in \mathbf{J}$. The unique natural transformation $\delta : W_1 \Rightarrow W_2$ such that $\delta;; U = \gamma$ is given by

$$\gamma_J : (U(W_1(J)), \alpha_{W_1(J)}) \to (U(W_2(J)), \alpha_{W_2(J)})$$

for each $J \in \mathbf{J}$. Naturality of γ (in $\mathbf{Y}$) implies naturality of δ (in $F//G$). For uniqueness, note that if $\delta' : W_1 \Rightarrow W_2$ is another natural transformation satisfying $\delta';; U = \gamma$ then $\delta' = \delta$ since U is faithful. $\square$

The above definition of $F//G$ motivates the use of inserters as a 2-categorical abstraction of categories of algebras. For example, if $F : \mathbf{Y} \to \mathbf{Y}$ is a functor then the category of F-algebras is given by $F//Id$ and the category of F-coalgebras is given by $Id//F$. In general we shall call arrows of the form $f : F(Y) \to G(Y)$ *F, G-algebras*. We choose not to use Hagino's "dialgebra" terminology [hag87] because our functors cannot have mixed variances (i.e. they are strictly covariant).

An important example of an inserter is the *arrow category* $\mathbf{Z}^{\to}$ of an arbitrary category $\mathbf{Z}$. In general, $\mathbf{Z}^{\to}$ is isomorphic to $\Pi_0//\Pi_1$ where $\Pi_0, \Pi_1 : \mathbf{Z}^2 \to \mathbf{Z}$ are the canonical projections. In **Cat**, the objects of $\Pi_0//\Pi_1$ correspond to arrows in $\mathbf{Z}$ and the arrows correspond to commuting squares.

By using inserters in this way we can obtain a 2-categorical generalization of arrow categories in any 2-category $\mathbb{C}$ with finite limits and inserters. More generally, suppose $(F//G, U, \alpha)$ is an inserter of $F, G : \mathbf{Y} \to \mathbf{Z}$. By the (1D) property of $F//G$, for each 2-cell $f : Y;; F \to Y;; G$ there is a unique 1-cell $\ulcorner f \urcorner$ such that $\ulcorner f \urcorner;; U = Y$ and $\ulcorner f \urcorner;; \alpha = f$. The point is 2-cells into $\mathbf{Z}$ correspond to 1-cells into $\Pi_0//\Pi_1$.

3.3 Combinators

We now introduce our 2-categorical notion of combinator, which can be viewed as an operation for constructing 2-cells from other 2-cells. In a 2-category with inserters, combinators can be interpreted as 1-cells between inserters. Under this interpretation, combinators satisfy a 2-categorical version of parametricity.

Definition 3.3.1 *Suppose* $F, G : \mathbf{Y} \to \mathbf{Z}$ *and* $F', G' : \mathbf{Y} \to \mathbf{Z}'$ *are 1-cells.*

1. *A* **combinator**
$$c\{F \to G\} : F' \to G'$$
is a construction of 2-cells: for every 1-cell Y *and 2-cell* $f : Y;; F \Rightarrow Y;; G$ *there is a specified 2-cell* $c\{f\} : Y;; F' \Rightarrow Y;; G'$.

2. *A combinator* $c\{F \to G\} : F' \to G'$ *is called a* **functorial combinator** *if the following property is satisfied:*

(func) *for every pair of composable 1-cells* X *and* Y, *and for every 2-cell* $f : Y;; F \Rightarrow Y;; G$ *the equation*
$$X;; c\{f\} = c\{X;; f\}$$
holds.

3. *A functorial combinator* $c\{F \to G\} : F' \to G'$ *is called a* **parametric combinator** *if the following property is satisfied:*

(para) *for all 2-cells* $f : Y;; F \Rightarrow Y;; G$, $f' : Y';; F \Rightarrow Y';; G$, *and* $h : Y \Rightarrow Y'$, *if the equation*
$$f; (h;; G) = (h;; F); f'$$
holds, then the equation
$$c\{f\}; (h;; G') = (h;; F'); c\{f'\}$$
also holds.

Suppose $(F//G, U, \alpha)$ and $(F'//G', U', \alpha')$ are inserters. In a 2-category with inserters, there is a bijective correspondence between functorial combinators $c\{F \to G\} : F' \to G'$ and 1-cells $C : F//G \to F'//G'$ for which $C;; U' = U$. This correspondence is expressed by a pair of inverse constructions: from combinators to 1-cells, and from 1-cells to combinators.

- (*combinators to 1-cells*) Suppose $c\{F \to G\} : F' \to G'$ is a combinator. Using the 2-cell $c\{\alpha\}$ with the (1D) property of $F'//G'$ gives a unique 1-cell $\ulcorner c\{\alpha\} \urcorner : F//G \to F'//G'$ such that $\ulcorner c\{\alpha\} \urcorner;; U' = U$ and $\ulcorner c\{\alpha\} \urcorner;; \alpha' = c\{\alpha\}$.

- (*1-cells to combinators*) Suppose $C : F//G \to F'//G'$ is a 1-cell such that $C;; U' = U$. Let $cmb(C)\{F \to G\} : F' \to G'$ be the combinator defined by $cmb(C)\{f\} = \ulcorner f \urcorner ;; C ;; \alpha'$ for each appropriate 2-cell f. (In fact, $cmb(C)$ is a parametric combinator. See below.)

Lemma 3.3.2 *If $c\{F \to G\} : F' \to G'$ is a functorial combinator then $c\{f\} = \mathrm{cmb}(\ulcorner c\{\alpha\} \urcorner)\{f\}$ for each 2-cell $f : Y;; F \Rightarrow Y;; G$.*

Proof.

$$
\begin{array}{cl}
 & cmb(\ulcorner c\{\alpha\} \urcorner)\{f\} \\
= & \\
 & \ulcorner f \urcorner ;; \ulcorner c\{\alpha\} \urcorner ;; \alpha' \\
= & \\
 & \ulcorner f \urcorner ;; c\{\alpha\} \\
= & \\
 & c\{\ulcorner f \urcorner ;; \alpha\} \\
= & \\
 & c\{f\}
\end{array}
$$

$\square$

Lemma 3.3.3 *If $C : F//G \to F'//G'$ is a 1-cell such that $C;; U' = U$ then $C = \ulcorner \mathrm{cmb}(C)\{\alpha\} \urcorner$.*

Proof. First notice that $\ulcorner \alpha \urcorner = Id$ follows from the (1D) property of $F//G$ since $\ulcorner \alpha \urcorner ;; U = U$ and $\ulcorner \alpha \urcorner ;; \alpha = \alpha$. Using this we can expand $\ulcorner cmb(C)\{\alpha\} \urcorner$ as follows:

$$
\begin{array}{cl}
 & \ulcorner cmb(C)\{\alpha\} \urcorner \\
= & \\
 & \ulcorner \ulcorner \alpha \urcorner ;; C ;; \alpha' \urcorner \\
= & \\
 & \ulcorner C ;; \alpha' \urcorner
\end{array}
$$

The equation $\ulcorner C;; \alpha' \urcorner = C$ follows from the (1D) property of $F'//G'$ since $\ulcorner C;; \alpha' \urcorner ;; U' = C;; U'$ and $\ulcorner C;; \alpha' \urcorner ;; \alpha' = C;; \alpha'$. $\square$

Our use of inserters in the interpretation of combinators as 1-cells is interesting because these 1-cells adequately model parametricity. For example, suppose $C : F//G \to F'//G'$ is a 1-cell in **Cat** (i.e. a functor) such that

$C;; U' = U$, and suppose c is the combinator $cmb(C)$. The object map of C takes F, G-algebras in $\mathbf{Z}$ to F', G'-algebras in $\mathbf{Z}'$:

$$[Y, F(Y) \xrightarrow{f} G(Y)] \quad \mapsto \quad [Y, F'(Y) \xrightarrow{c\{f\}} G'(Y)]$$

Notice that the parameter Y is common throughout as required by the equation $C;; U' = U$. The arrow map of C maps commuting squares as follows:

$$\begin{array}{ccc}
\begin{array}{ccc} F(Y) & \xrightarrow{f} & G(Y) \\ {\scriptstyle F(h)}\downarrow & & \downarrow{\scriptstyle G(h)} \\ F(Y') & \xrightarrow[f']{} & G(Y') \end{array}
& \mapsto &
\begin{array}{ccc} F'(Y) & \xrightarrow{c\{f\}} & G'(Y) \\ {\scriptstyle F'(h)}\downarrow & & \downarrow{\scriptstyle G'(h)} \\ F'(Y') & \xrightarrow[c\{f'\}]{} & G'(Y') \end{array}
\end{array}$$

This preservation of algebraic homomorphisms illustrates parametricity for combinators in **Cat**. The parametric behavior of combinators in an arbitrary 2-category with inserters is given by the following lemma.

Lemma 3.3.4 (Parametricity) *If $C : F//G \to F'//G'$ is a 1-cell such that $C;; U' = U$ then $cmb(C)$ is a parametric combinator.*

Proof. First we show $cmb(C)$ is functorial. By definition of $cmb(C)$, we have that $cmb(C)\{X;; f\} = \ulcorner X;; f \urcorner;; C;; \alpha'$ and $X;; cmb(C)\{f\} = X;; \ulcorner f \urcorner;; C;; \alpha'$. The equation $X;; \ulcorner f \urcorner = \ulcorner X;; f \urcorner$ follows from the (1D) property of $F//G$ since $X;; \ulcorner f \urcorner;; U = \ulcorner X;; f \urcorner;; U$ and $X;; \ulcorner f \urcorner;; \alpha = \ulcorner X;; f \urcorner;; \alpha$.

Next we show $cmb(C)$ is parametric. Suppose $f; (h;; G) = (h;; F); f'$. This equation implies $(\ulcorner f \urcorner;; \alpha); (h;; G) = (h;; F); (\ulcorner f' \urcorner;; \alpha)$. The (2D) property of $F//G$ implies the existence of a unique $\delta : \ulcorner f \urcorner \Rightarrow \ulcorner f' \urcorner$ such that $\delta;; U = h$. Now using $\delta;; U = \delta;; C;; U'$ and the interchange and identity laws, we have that

$$\begin{array}{rcl}
(h;; F'); (\ulcorner f' \urcorner;; C;; \alpha') & = & (\delta;; C;; U';; F'); (\ulcorner f' \urcorner;; C;; \alpha') \\
& = & ((\delta;; C); (\ulcorner f' \urcorner;; C));; ((U';; F'); \alpha') \\
& = & \delta;; C;; \alpha' \\
& = & ((\ulcorner f \urcorner;; C); (\delta;; C));; (\alpha'; (U';; G')) \\
(\ulcorner f \urcorner;; C;; \alpha'); (h;; G') & = & (\ulcorner f \urcorner;; C;; \alpha'); (\delta;; C;; U';; G')
\end{array}$$

This implies $cmb(C)\{f\}; (h;; G') = (h;; F'); cmb(C)\{f'\}$. $\Box$

Theorem 3.3.5 *In a 2-category with inserters, there is a bijective correspondence between*

- *functorial combinators* $c\{F \to G\} : F' \to G'$*, and*
- *1-cells* $C : F//G \to F'//G'$ *such that* $C;;U' = U$*.*

Furthermore, functorial combinators satisfy parametricity (i.e. they are parametric combinators).

Proof. By Lemma 3.3.2, for each functorial combinator c there is a 1-cell $\ulcorner c\{\alpha\}\urcorner$ such that $\ulcorner c\{\alpha\}\urcorner;;U' = U$ and $cmb(\ulcorner c\{\alpha\}\urcorner) = c$. By Lemma 3.3.3, for each 1-cell C such that $C;;U' = U$ there is a combinator $cmb(C)$ such that $C = \ulcorner cmb(C)\{\alpha\}\urcorner$. By Lemma 3.3.4, $cmb(C)$ is parametric. $\square$

3.4 Type Theoretic Structure

In the term logic of Chapter 2, combinators are expressed by rules of the form

$$\frac{\Gamma \vdash f : S(A) \to T(A)}{\Gamma \vdash c\{f\} : S'(A) \to T'(A)}$$

Such a combinator is called *polymorphic* because it is parametrized by a type variable A and function variable f, where these "variables" are in the meta-language of the theory. We could conceivably regard the meta-language itself as a type theory in which type variables and function variables exist as formal parameters to combinator expressions. For example, the above combinator might be expressed as a judgement of the form

$$A : \text{type} \mid f : S(A) \to T(A) \models c\{f\} : S'(A) \to T'(A).$$

In the remainder of this chapter we pursue an abstract description of such a theory. The emerging view will be to regard polymorphic types as 1-cells and polymorphic combinator expressions as 2-categorical combinators.

In this section we begin by describing a categorical structure appropriate as a semantic framework for a 2-categorical combinator theory. Essentially we have an algebraic theory of combinators layered over an algebraic theory of types. Such a layering can be described in terms of fibrations (see [jac94]).

Assume $\mathbb{C}$ is a 2-category with finite limits and inserters. Define $I(\mathbb{C})$, the category of inserters in $\mathbb{C}$, as follows:

- objects are of the form $[(F,G),(\mathbf{I},U,\alpha)]$ where $(\mathbf{I},U,\alpha)$ is an inserter of the 1-cells F,G, and
- arrows are pairs of 1-cells

  $$(C,H) : [(F,G),(\mathbf{I},U,\alpha)] \to [(F',G'),(\mathbf{I}',U',\alpha')]$$

  such that $C;;U' = U;;H$.

Since inserters are unique only up to isomorphism there may be many isomorphic objects in $I(\mathbb{C})$ for a given pair of 1-cells F,G.

Next define the functor $\rho : I(\mathbb{C}) \to \mathbb{C}$ by $\rho(C,H) = H$. We shall prove that ρ is a fibration. Of particular interest is the fact that the vertical arrows in this fibration are exactly the combinators of $\mathbb{C}$.

Lemma 3.4.1 *Suppose $(\mathbf{I},U,\alpha)$ is an inserter of $F,G : \mathbf{Y} \to \mathbf{Z}$ and suppose $H : \mathbf{X} \to \mathbf{Y}$ is a 1-cell. For every inserter $(\mathbf{J},V,\beta)$ of $FH,GH : \mathbf{X} \to \mathbf{Z}$ there is a unique 1-cell $E : \mathbf{J} \to \mathbf{I}$ such that*

$$\begin{tikzcd} \mathbf{J} \arrow[r, dashed, "E"] \arrow[d, "V"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathbf{I} \arrow[d, "U"] \\ \mathbf{X} \arrow[r, "H"'] & \mathbf{Y} \end{tikzcd}$$

is a pullback and $E;;\alpha = \beta$. Conversely, if $E : \mathbf{J} \to \mathbf{I}$ is a 1-cell such that $E;;U = V;;H$ is a pullback then $(\mathbf{J},V,E;;\alpha)$ is an inserter of FH,GH.

Proof. Using β, the (1D) property of $(\mathbf{I},U,\alpha)$ gives a unique E such that $E;;U = V;;H$ and $E;;\alpha = \beta$. To show that we have a pullback, suppose W and W' satisfy $W;;H = W';;U$. Using $W';;\alpha$, the (1D) property of $(\mathbf{J},V,\beta)$ gives a unique W'' such that $W'';;V = W$ and $W'';;\beta = W';;\alpha$. To show that $W'';;E = W'$, notice that $W'';;E;;U = W';;U$ and $W'';;E;;\alpha = W';;\alpha$ and use the (2D) property of $(\mathbf{I},U,\alpha)$.

Conversely, suppose $E;;U = V;;H$ is a pullback. To show the (1D) property of $(\mathbf{J},V,E;;\alpha)$, suppose W is a 1-cell and $\gamma : FHW \Rightarrow GHW$ is a 2-cell. Using the (1D) property of $(\mathbf{I},U,\alpha)$ gives a unique W' such that $W';;U = W;;H$ and $W';;\alpha = \gamma$. Using the pullback gives a unique W'' such that $W'';;V = W$ and $W'';;E = W'$. It follows that $W'';;E;;\alpha = \gamma$.

To show the (2D) property of $(\mathbf{J}, V, E;; \alpha)$, suppose W_1 and W_2 are 1-cells and $\gamma : W_1;; V \Rightarrow W_2;; V$ is a 2-cell such that $(\gamma;; FH); (W_2;; E;; \alpha) = (W_1;; E;; \alpha); (\gamma;; GH)$. Using $\gamma;; H$, the (2D) property of $(\mathbf{I}, U, \alpha)$ gives a unique 2-cell δ such that $\delta;; U = \gamma;; H$. The (2D) property of the pullback gives a unique 2-cell δ' such that $\delta';; E = \delta$ and $\delta';; V = \gamma$. $\square$

Lemma 3.4.2 *$\rho : I(\mathbb{C}) \to \mathbb{C}$ is a fibration.*

Proof. Suppose $(\mathbf{I}, U, \alpha)$ is an inserter of $F, G : \mathbf{Y} \to \mathbf{Z}$ and suppose $H : \mathbf{X} \to \mathbf{Y}$ is a 1-cell. Under the assumption that $\mathbb{C}$ has inserters, there exists an inserter, say $(\mathbf{J}, V, \beta)$, of FH, GH. The previous lemma asserts the existence of an arrow

$$(E, H) : [(FH, GH), (\mathbf{J}, V, \beta)] \to [(F, G), (\mathbf{I}, U, \alpha)]$$

in $I(\mathbb{C})$ which is a pullback in $\mathbb{C}$. Using this pullback it is easy to show (E, H) is cartesian. $\square$

In our discussion so far we have not made use of any particular choice of inserters; we have only required their existence in $\mathbb{C}$. When working in some specific 2-category, however, we should like to work with a particular choice of inserters (and hence a particular choice of combinators). For example, in **Cat** we like to regard the algebra category $F//G$ as *the* inserter of F, G. Happily the fibrational structure remains intact for any choice of inserters.

Suppose we choose, for every pair of 1-cells $F, G : \mathbf{Y} \to \mathbf{Z}$ in $\mathbb{C}$, a particular inserter $F//G$ designated as *the* inserter of F, G. Let $I(\mathbb{C})^{/\!/}$ be the subcategory of $I(\mathbb{C})$ whose objects consist only of these designated inserters. Similarly, let $\rho^{/\!/}$ be the functor ρ restricted to the domain $I(\mathbb{C})^{/\!/}$.

Theorem 3.4.3 *$\rho^{/\!/} : I(\mathbb{C})^{/\!/} \to \mathbb{C}$ is a cloven fibration (the cleavage is determined by the choice of inserters).*

Proof. For a particular choice of inserters, the cartesian arrows obtained in the proof of Lemma 3.4.2 are uniquely determined, as shown in Lemma 3.4.1. $\square$

A useful consequence of a fibration being cloven is the presence of "reindexing" functors between the fibres. In the cloven fibration $\rho^{/\!/}$ we have, for each 1-cell $H : \mathbf{X} \to \mathbf{Y}$ in $\mathbb{C}$, a reindexing functor

$$H^* : (\rho^{/\!/})^{-1}(\mathbf{Y}) \to (\rho^{/\!/})^{-1}(\mathbf{X})$$

whose behavior on combinators is given by

$$[F//G \xrightarrow{c} F'//G'] \;\mapsto\; [FH//GH \xrightarrow{H^*(c)} F'H//G'H].$$

This behavior corresponds to type substitution in the combinator theory of the next section.

An essential property in the combinator theory semantics is the presence of finite products for constructing algebraic type expressions and algebraic combinator expressions. In the fibration this means there are finite products in the base and fibred finite products in the total category.

Lemma 3.4.4 *$\rho^{/\!/} : I(\mathbb{C})^{/\!/} \to \mathbb{C}$ has fibred finite products: for each 0-cell $\mathbf{Y} \in \mathbb{C}$,*

1. *the final object in $(\rho^{/\!/})^{-1}(\mathbf{Y})$ is $!//!$, and*
2. *the binary product of $F//G$ and $F'//G'$ in $(\rho^{/\!/})^{-1}(\mathbf{Y})$ is $\langle F, F'\rangle \,//\, \langle G, G'\rangle$.*

Furthermore the reindexing functors preserve these finite products.

Proof. For each $F//G \in (\rho^{/\!/})^{-1}(\mathbf{Y})$ the unique $! : F//G \to !//!$ is obtained from the 2-cell $U;;!$ and the (1D) property of $!//!$. Preservation by reindexing is clear.

$\langle F, F'\rangle \,//\, \langle G, G'\rangle$ is a binary product of $F//G$ and $F'//G'$ since

$$\begin{tikzcd}
\langle F, F'\rangle \,//\, \langle G, G'\rangle \arrow[r, "P_1"] \arrow[d, "P_0"'] \arrow[dr, phantom, "\lrcorner", very near start] & F'//G' \arrow[d] \\
F//G \arrow[r] & \mathbf{Y}
\end{tikzcd}$$

is a pullback, where the projections P_0 and P_1 are obtained by the (1D) properties of $F//G$ and $F'//G'$. Preservation by reindexing is again clear. $\square$

3.5 Combinator Theories

In this section a type theory for deriving polymorphic combinator expressions is given. The semantic framework of the previous section is used to interpret polymorphic types and combinator expressions in the 2-category $\mathbb{C}$.

Syntax

Assume TC is a set of symbols called *type constructors*. Each $F \in TC$ comes with a finite non-negative integer called the *arity* of F. Write $F \in TC_k$ when $F \in TC$ is of arity $k \geq 0$. Assume we are given a countably infinite set of symbols called *type variables*. Types are given by judgements of the form

$$\Sigma \| \models T : \text{type}$$

where Σ is a finite sequence of distinct type variables, called a *type context*, and includes all those occurring in T. The rules for deriving types and manipulating type contexts are as follows:

- $\dfrac{}{A \| \models A : \text{type}}$ *type var intro*
- $\dfrac{\Sigma \mid \Theta \models \varphi}{\Sigma, A \mid \Theta \models \varphi}$ *type var weaken* $(A \notin \Gamma)$
- $\dfrac{\Sigma, A, A', \Sigma' \mid \Theta \models \varphi}{\Sigma, A', A, \Sigma' \mid \Theta \models \varphi}$ *type var exch*
- $\dfrac{\Sigma, A \mid \Theta \models \varphi \qquad \Sigma \| \models S : \text{type}}{\Sigma \mid \Theta[S/A] \models \varphi[S/A]}$ *type var subst*
- $\dfrac{}{\| \models F : \text{type}}$ $F \in TC_0$
- $\dfrac{\Sigma \| \models S_1 : \text{type} \quad \cdots \quad \Sigma \| \models S_k : \text{type}}{\Sigma \| \models F(S_1 \cdots S_k) : \text{type}}$ $F \in TC_k$ $(k > 0)$

Assume CS is a set of *combinator specifications* of the form

$$c\{F_1(\bar{A}) \to G_1(\bar{A}) \cdots F_n(\bar{A}) \to G_n(\bar{A}\} : F_{n+1}(\bar{A}) \to G_{n+1}(\bar{A})$$

where $\bar{A}$ is a type variable context and each $F_i(\bar{A})$ and $G_i(\bar{A})$ $(1 \leq i \leq n)$ is a type expression with respect to $\bar{A}$. In the case when $\bar{A}$ is empty and $n = 0$, c is just a function symbol with domain F_1 and codomain G_1. Assume we are given a countably infinite set of symbols called *function variables*. Combinator expressions are given by judgments of the form

$$\Sigma \mid \Theta \models f : S \to T$$

where Σ is a type variable context and Θ is a context of function variables

$$h_1 : X_1 \to Y_1, \ldots, h_m : X_m \to Y_m$$

where $\Sigma | \models X_i$: type, and $\Sigma | \models Y_i$: type, for each $1 \leq i \leq m$. The rules for deriving functions and manipulating function contexts are as follows:

- $\dfrac{\Sigma|\models S : \text{type} \qquad \Sigma|\models T : \text{type}}{\Sigma \mid h : S \to T \models h : S \to T}$ *fn var intro*
- $\dfrac{\Sigma \mid \Theta \models \varphi \quad \Sigma|\models S : \text{type} \quad \Sigma|\models T : \text{type}}{\Sigma \mid \Theta, h : S \to T \models \varphi}$ *fn var weaken* $(h \notin \Theta)$
- $\dfrac{\Sigma \mid \Theta, h, h', \Theta' \models \varphi}{\Sigma \mid \Theta, h', h, \Theta' \models \varphi}$ *fn var exch*
- $\dfrac{\Sigma \mid \Theta, h : S \to T \models \varphi \qquad \Sigma \mid \Theta \models f : S \to T}{\Sigma \mid \Theta \models \varphi[f/h]}$ *fn var subst*
- $\dfrac{\Sigma \mid \Theta \models f_1 : F_1(S) \to G_1(S) \cdots f_n : F_n(S) \to G_n(S)}{\Sigma \mid \Theta \models c\{f_1 \cdots f_n\} : F_{n+1}(S) \to G_{n+1}(S)}$ $c \in CS$
- $\dfrac{\Sigma|\models S : \text{type}}{\Sigma|\models id : S \to S}$ *identity*
- $\dfrac{\Sigma \mid \Theta \models f : S \to T \qquad \Sigma \mid \Theta \models g : T \to U}{\Sigma \mid \Theta \models (f;g) : S \to U}$ *compose*
- $\dfrac{\Sigma \mid \Theta \models f_1 : S_1 \to T_1 \quad \cdots \quad \Sigma \mid \Theta \models f_k : S_k \to T_k}{\Sigma \mid \Theta \models F\{f_1 \cdots f_k\} : F(S_1 \cdots S_k) \to F(T_1 \cdots T_k)}$ $F \in TC_k \ (k > 0)$

The equations of the theory are given by the rules listed below. These include parametricity of all the combinators, functoriality of all the type constructors, and the categorical composition laws.

- *parametricity*: given combinator expressions

$$\begin{array}{l} \Sigma \mid \Theta \models h : S \to T \\ \Sigma \mid \Theta \models c\{f_1 \cdots f_n\} : F_{n+1}(S) \to G_{n+1}(S) \\ \Sigma \mid \Theta \models c\{f'_1 \cdots f'_n\} : F_{n+1}(T) \to G_{n+1}(T) \end{array}$$

then

$$\forall_{i=1}^{n}.f_i; G_i\{h\} = F_i\{h\}; f'_i$$

implies

$$c\{f_1 \cdots f_n\}; G_{n+1}\{h\} = F_{n+1}\{h\}; c\{f'_1 \cdots f'_n\}$$

- *functoriality*: for every $F \in TC$,

$$F\{f;g\} = F\{f\}; F\{g\} \qquad F\{id\} = id$$

- *composition laws*:

$$(f;g);h = f;(g;h) \qquad f;id = f = id;f$$

Semantics

The semantics of a combinator theory begins by fixing a 0-cell $\mathbf{X} \in \mathbb{C}$. Type judgements are interpreted as 1-cells into $\mathbf{X}$ and combinator expressions are interpreted as combinators in $\mathbb{C}$.

Assume each $F \in TC_k$ (for each $k \geq 0$) is assigned a 1-cell $\ulcorner F \urcorner : \mathbf{X}^k \to \mathbf{X}$. Now each type judgement $\Sigma | \models T$: type is interpreted as a 1-cell

$$[\![\Sigma | \models T]\!] : \mathbf{X}^{|\Sigma|} \to \mathbf{X}$$

as follows:

- $[\![A_1 \cdots A_j | \models A_i]\!] = \Pi_i$ for each $1 \leq i \leq j$,
- if $F \in TC_0$ then $[\![\Sigma | \models F]\!] = !;; \ulcorner F \urcorner$,
- if $F \in TC_k$ $(k > 0)$ then

$$[\![\Sigma | \models F(S_1 \cdots S_k)]\!] \quad = \quad \langle [\![\Sigma | \models S_1]\!] \cdots [\![\Sigma | \models S_k]\!] \rangle \,;; \ulcorner F \urcorner$$

Assume each $c \in CS$ is assigned a combinator

$$\ulcorner c \urcorner : \Pi_{i=1}^{n} \ulcorner F_i \urcorner // \ulcorner G_i \urcorner \longrightarrow \ulcorner F_{n+1} \urcorner // \ulcorner G_{n+1} \urcorner$$

Now each combinator expression $\Sigma \mid \Theta \models f : S \to T$ where $\Theta = h_1 : X_1 \to Y_1, \ldots, h_m : X_m \to Y_m$ is interpreted as a combinator

$$[\![\Sigma \mid \Theta \models f]\!] : \Pi_{i=1}^{m} [\![\Sigma | \models X_i]\!] \,//\, [\![\Sigma | \models Y_i]\!] \longrightarrow [\![\Sigma | \models S]\!] \,//\, [\![\Sigma | \models T]\!]$$

in $(\rho^{/\!/})^{-1}(\mathbf{X}^{|\Sigma|})$ as follows:

- $[\![\Sigma \mid \Theta \models h_i]\!] = \Pi_i$ for each $1 \leq i \leq m$,
- if $c : F(\bar{S}) \to G(\bar{S})$ is a combinator of 0 arguments then

$$[\![\Sigma \mid \Theta \models c]\!] \quad = \quad ! ;; \left[\!\left[\Sigma | \!\models \bar{S}\right]\!\right]^* (\ulcorner c \urcorner)$$

- if $c\{f_1 \cdots f_n\} : F(\bar{S}) \to G(\bar{S})$ then

$$\begin{aligned}&[\![\Sigma \mid \Theta \models c\{f_1 \cdots f_n\}]\!] \\ &= \langle [\![\Sigma \mid \Theta \models f_1]\!] \cdots [\![\Sigma \mid \Theta \models f_n]\!] \rangle ;; \left[\!\left[\Sigma | \!\models \bar{S}\right]\!\right]^* (\ulcorner c \urcorner)\end{aligned}$$

The effect of our interpretation is to regard polymorphic combinator expressions as 2-categorical combinators of the following form $(k, n \geq 0)$:

$$\begin{tikzcd}
F//G \arrow[r, "c"] \arrow[d] & F'//G' \arrow[d] \\
\mathbf{X}^k \arrow[r, equal] \arrow[d, bend right] \arrow[d, bend left] \arrow[d, phantom, "\Rightarrow"] & \mathbf{X}^k \arrow[d, bend right] \arrow[d, bend left] \arrow[d, phantom, "\Rightarrow"] \\
\mathbf{X}^n & \mathbf{X}.
\end{tikzcd}$$

Chapter 4

Datatypes

The purpose of this chapter is to give a 2-categorical description of inductive and co-inductive datatypes. Combinators associated with these datatypes as well as the relationship between parametricity and the universal properties are of particular interest. We follow the 2-categorical description given in [d1] where inserters and adjunctions are used to get appropriate abstractions of initial algebras and final co-algebras. Also in [d1], fibrations are used to parametrize the universal conditions but this extra step is unnecessary and is rectified in this chapter.

4.1 Universal Properties and Parametricity

Asserting that a category has certain datatypes means there are objects and arrows which satisfy certain universal properties. In this section we describe these universal properties for inductive and co-inductive datatypes at the 2-categorical level of generality. We also describe how parametricity is related to the universal properties.

Inductive Datatypes

Recall the form of an inductive datatype declaration in Charity:

$$\begin{array}{rcl} \text{data } L(A) \to C & = & c_1 : E_1(A,C) \to C \\ & | & \cdots \\ & | & c_n : E_n(A,C) \to C \end{array}$$

If $E_1, \ldots, E_n$ are 1-cells of the form $E_i : \mathbf{A} \times \mathbf{C} \to \mathbf{C}$ (think of $\mathbf{A}$ as $\mathbf{C}^k$ for some $k \geq 0$) then such a declaration introduces a 1-cell

$$L : \mathbf{A} \to \mathbf{C}$$

and 2-cells

$$c_1 : \langle Id, L \rangle \,;\!;\, E_1 \Rightarrow L, \ldots, c_n : \langle Id, L \rangle \,;\!;\, E_n \Rightarrow L$$

such that the universal property given by the following definition is satisfied.

Definition 4.1.1 $(L, c_1, \ldots, c_n)$ *is an inductive datatype if for each n-tuple of 2-cells*

$$f_1 : \langle A, C \rangle \,;\!;\, E_1 \Rightarrow C, \ldots, f_n : \langle A, C \rangle \,;\!;\, E_n \Rightarrow C$$

there is a unique 2-cell

$$\mathit{fold}^L\{f_1 \cdots f_n\} : A;\!;\, L \Rightarrow C$$

such that

$$\textit{(fold-rec)} \qquad (A;\!;\, c_i); \mathit{fold}^L\{f_1 \cdots f_n\} = (\big\langle A, \mathit{fold}^L\{f_1 \cdots f_n\}\big\rangle \,;\!;\, E_i); f_i$$

holds for each $1 \leq i \leq n$.

For the case when the 1-cells are functors and the 2-cells are natural transformations, the *(fold-rec)* equations are given by commuting squares of the form

$$\begin{tikzcd} E_i(A, L(A)) \arrow[r, "c_i"] \arrow[d, "E_i(id, \mathit{fold}^L\{f_1 \cdots f_n\})"'] & L(A) \arrow[d, dashed, "\mathit{fold}^L\{f_1 \cdots f_n\}"] \\ E_i(A, C) \arrow[r, "f_i"'] & C \end{tikzcd}$$

for each $1 \leq i \leq n$. These squares are called the recursion diagrams of L.

Examples

There are many well-known examples of inductive datatypes. Some of the standard ones along with their recursion diagrams are as follows:

1. *binary coproducts*:

$$\begin{array}{rcl} \text{data } A + B \to C & = & b_0 : A \to C \\ & \mid & b_1 : B \to C \end{array}$$

$$\begin{tikzcd} A \arrow[r, "b_0"] \arrow[dr, "f"'] & A + B \arrow[d, dashed, "{[f,g]}"] & B \arrow[l, "b_1"'] \arrow[dl, "g"] \\ & C & \end{tikzcd}$$

2. *natural numbers*:

$$\begin{array}{rcl} \text{data } \mathit{Nat} \to C & = & \mathit{zero} : 1 \to C \\ & \mid & \mathit{succ} : C \to C \end{array}$$

$$\begin{tikzcd} 1 \arrow[r, "\mathit{zero}"] \arrow[dr, "f"'] & \mathit{Nat} \arrow[d, dashed, "\mathit{fold}^{\mathrm{Nat}}\{f,g\}"] & \mathit{Nat} \arrow[l, "\mathit{succ}"'] \arrow[d, "\mathit{fold}^{\mathrm{Nat}}\{f,g\}"] \\ & C & C \arrow[l, "g"] \end{tikzcd}$$

3. *finite lists*:

$$\begin{array}{rcl} \text{data } \mathit{List}(A) \to C & = & \mathit{nil} : 1 \to C \\ & \mid & \mathit{cons} : (A \times C) \to C \end{array}$$

$$\begin{tikzcd} 1 \arrow[r, "\mathit{nil}"] \arrow[dr, "f"'] & \mathit{List}(A) \arrow[d, dashed, "\mathit{fold}^{\mathrm{List}}\{f,g\}"] & A \times \mathit{List}(A) \arrow[l, "\mathit{cons}"'] \arrow[d, "\mathit{id} \times \mathit{fold}^{\mathrm{List}}\{f,g\}"] \\ & C & A \times C \arrow[l, "g"] \end{tikzcd}$$

The universal property of an inductive datatype asserts both the *existence* and *uniqueness* of 2-cells $\mathit{fold}^L\{f_1 \cdots f_n\}$. In order to isolate the relationship between the universal property and parametricity, we consider a weaker form of the universal property in which the uniqueness requirement is dropped. A similar discussion for datatypes in the lambda calculus can be found in [has94].

Definition 4.1.2 $(L, c_1, \ldots, c_n)$ *is a weak inductive datatype if for each n-tuple of 2-cells*

$$f_1 : \langle A, C\rangle \,;;\, E_1 \Rightarrow C, \ldots, f_n : \langle A, C\rangle \,;;\, E_n \Rightarrow C$$

there is a 2-cell

$$\mathit{fold}^L\{f_1 \cdots f_n\} : A;; L \Rightarrow C$$

such that (fold-rec) holds for each $1 \leq i \leq n$.

Theorem 4.1.3 *If* $(L, c_1, \ldots, c_n)$ *is a weak inductive datatype then the following are equivalent:*

1. *folds uniquely satisfy the (fold-rec) equations,*
2. fold^L *is a parametric combinator and* $\mathit{fold}^L\{c_1 \cdots c_n\} = \mathit{id}_L$.

Proof. $(1 \Rightarrow 2)$. Suppose (1) holds. The equation $\mathit{fold}^L\{c_1 \cdots c_n\} = \mathit{id}_L$ follows immediately. To show fold^L is a functorial combinator we need

$$\mathit{fold}^L\{X;; f_1 \cdots X;; f_n\} = X;; \mathit{fold}^L\{f_1 \cdots f_n\}$$

to hold for each 1-cell X and 2-cells $f_1, \ldots, f_n$. This follows from fold^L uniqueness since

$$\begin{array}{ll} & (X;; A;; c_i); (X;; \mathit{fold}^L\{f_1 \cdots f_n\}) \\ = & \\ & X;; ((A;; c_i); \mathit{fold}^L\{f_1 \cdots f_n\}) \\ = & \\ & X;; ((\big\langle A, \mathit{fold}^L\{f_1 \cdots f_n\}\big\rangle \,;;\, E_i); f_i) \\ = & \\ & (\big\langle X;; A, X;; \mathit{fold}^L\{f_1 \cdots f_n\}\big\rangle \,;;\, E_i); (X;; f_i). \end{array}$$

To show fold^L is a parametric combinator, suppose $f_1, \ldots, f_n$ and $f'_1, \ldots, f'_n$ and a, c are 2-cells such that

$$f_i; c = (\langle a, c\rangle \,;;\, E_i); f'_i$$

holds for each $1 \leq i \leq n$. Let $k = \mathit{fold}^L\{f_1 \cdots f_n\}; c$. The equations *(fold-rec)* imply that

$$(A;; c_i); k = (\langle A, k\rangle ;; E_i); (\langle a, C\rangle ;; E_i); f_i'$$

holds for each $1 \leq i \leq n$. Let $h = (a;; L); \mathit{fold}^L\{f_1' \cdots f_n'\}$. The equations *(fold-rec)* imply that

$$(A;; c_i); h = (\langle A, h\rangle ;; E_i); (\langle a, C\rangle ;; E_i); f_i'$$

holds for each $1 \leq i \leq n$. By uniqueness, it follows that $h = k$.

$(2 \Rightarrow 1)$. Suppose (2) holds, and suppose $h : A;; L \Rightarrow C$ is a 2-cell such that

$$(A;; c_i); h = (\langle A, h\rangle ;; E_i); f_i$$

holds for each $1 \leq i \leq n$. Parametricity of fold^L implies that

$$(A;; \mathit{fold}^L\{c_1 \cdots c_n\}); h = (A;; L); \mathit{fold}^L\{f_1 \cdots f_n\}.$$

Since $\mathit{fold}^L\{c_1 \cdots c_n\} = \mathit{id}_L$, it follows that $h = \mathit{fold}^L\{f_1 \cdots f_n\}$. $\square$

Co-inductive Datatypes

The discussion here essentially parallels that for inductive datatypes above. We therefore omit some of the details which can easily be obtained by dualizing the appropriate inductive concepts.

Recall the form of a co-inductive datatype declaration in Charity:

$$\begin{array}{rcl} \text{data } C \to R(A) & = & d_1 : C \to E_1(A, C) \\ & | & \cdots \\ & | & d_n : C \to E_n(A, C) \end{array}$$

If $E_1, \ldots, E_n$ are 1-cells of the form $E_i : \mathbf{A} \times \mathbf{C} \to \mathbf{C}$ then such a declaration introduces a 1-cell

$$R : \mathbf{A} \to \mathbf{C}$$

and 2-cells

$$d_1 : R \Rightarrow \langle \mathit{Id}, R\rangle ;; E_1, \ldots, d_n : R \Rightarrow \langle \mathit{Id}, R\rangle ;; E_n$$

such that the universal property given by the following definition is satisfied.

Definition 4.1.4 $(R, d_1, \ldots, d_n)$ *is a co-inductive datatype if for each n-tuple of 2-cells*

$$g_1 : C \Rightarrow \langle A, C\rangle \,;;\, E_1, \ldots, g_n : C \Rightarrow \langle A, C\rangle \,;;\, E_n$$

there is a unique 2-cell

$$\mathit{unfold}^R\{g_1 \cdots g_n\} : C \Rightarrow A;; R$$

such that (unfold-rec):

$$\mathit{unfold}^R\{g_1 \cdots g_n\}; (A;; d_i) = g_i; (\left\langle A, \mathit{unfold}^R\{g_1 \cdots g_n\}\right\rangle \,;;\, E_i)$$

holds for each $1 \leq i \leq n$.

For the case when the 1-cells are functors and the 2-cells are natural transformations, the *(unfold-rec)* equations are given by commuting squares of the form

$$\begin{tikzcd} C \arrow[r, "g_i"] \arrow[d, dashed, "\mathit{unfold}^R\{g_1 \cdots g_n\}"'] & E_i(A, C) \arrow[d, "E_i(id, \mathit{unfold}^R\{g_1 \cdots g_n\})"] \\ R(A) \arrow[r, "d_i"'] & E_i(A, R(A)) \end{tikzcd}$$

for each $1 \leq i \leq n$.

Examples

Some examples of co-inductive datatypes are shown here along with their recursion diagrams.

1. *binary products*:

$$\begin{array}{rcl} \text{data } C \to A \times B & = & p_0 : C \to A \\ & | & p_1 : C \to B \end{array}$$

$$\begin{tikzcd} & C \arrow[dl, "f"'] \arrow[d, dashed, "\langle f, g\rangle"] \arrow[dr, "g"] & \\ A & A \times B \arrow[l, "p_0"] \arrow[r, "p_1"'] & B \end{tikzcd}$$

2. *infinite lists*:

$$
\begin{array}{rcl}
\text{data } C \to \mathit{InfList}(A) & = & \mathit{head} : C \to A \\
& | & \mathit{tail} : C \to C
\end{array}
$$

$$
\begin{tikzcd}
& C \arrow[dl, "f"'] \arrow[r, "g"] \arrow[d, dashed, "\mathit{unfold}^{\mathit{InfList}}\{f,g\}"] & C \arrow[d, "\mathit{unfold}^{\mathit{InfList}}\{f,g\}"] \\
A & \mathit{InfList}(A) \arrow[l, "\mathit{head}"] \arrow[r, "\mathit{tail}"'] & \mathit{InfList}(A)
\end{tikzcd}
$$

3. *potentially infinite lists*:

$$
\text{data } C \to \mathit{CoList}(A) \quad = \quad \mathit{split} : C \to 1 + (A \times C)
$$

$$
\begin{tikzcd}
C \arrow[r, "f"] \arrow[d, dashed, "\mathit{unfold}^{\mathit{CoList}}\{f\}"'] & 1 + (A \times C) \arrow[d, "\mathit{id} + (\mathit{id} \times \mathit{unfold}^{\mathit{CoList}}\{f\})"] \\
\mathit{CoList}(A) \arrow[r, "\mathit{split}"'] & 1 + (A \times \mathit{CoList}(A))
\end{tikzcd}
$$

The universal property of a co-inductive datatype is related to parametricity of the unfold^R combinator. As for inductive datatypes, this relationship is isolated by considering a weak form of the universal property. The following is analogous to Theorem 4.1.3.

Theorem 4.1.5 *If $(R, d_1, \ldots, d_n)$ is a weak co-inductive datatype then the following are equivalent:*

1. *unfolds uniquely satisfy the (unfold-rec) equations*
2. unfold^R *is a parametric combinator and* $\mathit{unfold}^R\{d_1 \cdots d_n\} = \mathit{id}_R$.

4.2 2-categorical Initiality and Finality

The universal properties of datatypes can be expressed as adjunctions asserting the existence of initial algebras (for inductive datatypes) and final co-algebras (for co-inductive datatypes). Using inserters we can abstract categories of algebras and categories of co-algebras to 0-cells in a 2-category. Thus, in a 2-category with inserters, datatypes can be expressed in terms of adjunctions.

In this section we describe two kinds of adjunctions, called *reflections* and *coreflections*, which give the appropriate universal properties for the datatypes. In the next section we use these adjunctions with suitable inserters to get the 2-categorical datatypes and their combinators.

Let us first review what an adjunction is, 2-categorically. Given a pair of 1-cells $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$, we say that F is left adjoint to G (or equivalently, G is right adjoint to F) if there is a pair of 2-cells

$$\eta : Id \Rightarrow F;; G \qquad \varepsilon : G;; F \Rightarrow Id$$

such that the following *triangle equalities* hold:

$$\begin{array}{ll} (\triangle 1) & (\eta;; F); (F;; \varepsilon) = F \\ (\triangle 2) & (G;; \eta); (\varepsilon;; G) = G \end{array}$$

It is well-known that adjunctions are equivalent to certain universal properties. In the discussion below, we review these properties for reflections and coreflections.

Initiality as a coreflection

Given a pair of 1-cells $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$, a *strict coreflection* is an adjunction $F \dashv G$ such that $F;; G = Id$ (i.e. η is the identity 2-cell). The universal property inherent in a strict coreflection is given by the following lemma:

Lemma 4.2.1 *If $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$ are 1-cells then the following are equivalent:*

1. *$F \dashv G$ is a strict coreflection,*
2. *$F;; G = Id$ and for each 2-cell $f : A \Rightarrow B;; G$ there is a unique 2-cell $f' : A;; F \Rightarrow B$ such that $f';; G = f$.*

Proof. $(1 \Rightarrow 2)$. Suppose $F \dashv G$ is a strict coreflection. For each $f : A \Rightarrow B;; G$, let $f' = (f;; F); (B;; \varepsilon)$. To show that $f';; G = f$ notice that $\varepsilon;; G = id_G$ holds by $(\triangle 2)$. Now suppose $k : A;; F \Rightarrow B$ satisfies $k;; G = f$. Naturality of

ε implies that $(A;;F;;\varepsilon);k = (k;;G;;F);(B;;\varepsilon)$. Since $F;;\varepsilon = id_F$ (by $(\triangle 1)$) and since $k;;G = f$ (by assumption) it follows that $k = f'$.

$(2 \Rightarrow 1)$. Suppose property (2) holds. Let $\varepsilon = id'_G$. Equation $(\triangle 2)$ holds since $id'_G;G = id_G$. To show equation $(\triangle 1)$, consider the identity 2-cell $F;;G$. By the universal property there is a unique 2-cell $(F;;G)'$ such that $(F;;G)';;G = F;;G$. One possibility is $(F;;G)' = F$. Another is $(F;;G)' = F;;\varepsilon$ since $F;;\varepsilon;;G = F;;id'_G;;G = F;;G$. By uniqueness, $F = F;;\varepsilon$. $\square$

Now we are ready to explain how a strict coreflection implies an initial condition. If A is a 1-cell with codomain $\mathbf{A}$, we will say $A;;F$ is initial with respect to A if for every 1-cell B such that $B;;G = A$, there is a unique 2-cell $u_B : A;;F \Rightarrow B$ such that $u_B;;G = id_A$. In a strict coreflection the unique u_B is given by $B;;\varepsilon$.

Corollary 4.2.2 *If $F \dashv G$ is a strict coreflection then for every 1-cell A with codomain $\mathbf{A}$, $A;;F$ is initial with respect to A.*

Finality as a reflection

We now dualize the previous discussion of coreflections. Given a pair of 1-cells $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$, a *strict reflection* is an adjunction $G \dashv F$ such that $F;;G = Id$ (i.e. ε is the identity 2-cell). The universal property is dual to that stated in Lemma 4.2.1.

Lemma 4.2.3 *If $F : \mathbf{A} \to \mathbf{B}$ and $G : \mathbf{B} \to \mathbf{A}$ are 1-cells then the following are equivalent:*

1. *$G \dashv F$ is a strict reflection,*
2. *for each 2-cell $g : B;;G \Rightarrow A$ there is a unique 2-cell $g' : B \Rightarrow A;;F$ such that $g';;G = g$.*

Just as strict coreflections imply a notion of initiality, reflections imply a dual notion of finality. If A is a 1-cell with codomain $\mathbf{A}$, we will say $A;;F$ is final with respect to A if for every 1-cell B such that $B;;G = A$, there is a unique 2-cell $v_B : B \Rightarrow A;;F$ such that $v_B;;G = id_A$.

Corollary 4.2.4 *If $G \dashv F$ is a strict reflection then, for every 1-cell A with codomain $\mathbf{A}$, $A;;F$ is final with respect to A.*

4.3 Datatypes as Adjunctions

In this section we use the adjunctions of the previous section to describe datatypes 2-categorically. Of particular interest is how the fold and unfold combinators arise at this level of generality. We consider inductive datatypes and then co-inductive datatypes, assuming throughout that $E_1, \ldots, E_n$ are 1-cells of the form $E_i : \mathbf{A} \times \mathbf{C} \to \mathbf{C}$ for each $1 \leq i \leq n$.

Suppose $(\langle E_1 \ldots E_n \rangle /\!/ \Pi_1 ;; \Delta_n, U, \alpha)$ is an inserter. The 0-cell given by $\langle E_1 \ldots E_n \rangle /\!/ \Pi_1 ;; \Delta_n$ is a 2-categorical abstraction of the category of algebras for an inductive datatype. Now asserting the presence of initial algebras (in the sense of Corollary 4.2.2) is expressed as a strict coreflection of the form

$$\begin{tikzcd} \langle E_1 \ldots E_n \rangle /\!/ \Pi_1 ;; \Delta_n \arrow[d, bend left, "U ;; \Pi_0"] \arrow[d, phantom, "\dashv"] \\ \mathbf{A} \arrow[u, bend left, "\ulcorner \bar{c} \urcorner"] \end{tikzcd}$$

(i.e. $\ulcorner \bar{c} \urcorner$ is left adjoint to $U ;; \Pi_0$) with unit the identity 2-cell (i.e. $\ulcorner \bar{c} \urcorner ;; U ;; \Pi_0 = Id$) and co-unit denoted by $\varepsilon : U ;; \Pi_0 ;; \ulcorner \bar{c} \urcorner \Rightarrow Id$. This adjunction gives rise to the following:

- a 1-cell $L : \mathbf{A} \to \mathbf{C}$ (i.e. the new type constructor) given by the composite $\ulcorner \bar{c} \urcorner ;; U ;; \Pi_1$,
- 2-cells $c_i : \langle Id, L \rangle ;; E_i \Rightarrow L$ (for each $1 \leq i \leq n$) (i.e. the new term constructors) collectively given by

  $$\bar{c} = \ulcorner \bar{c} \urcorner ;; \alpha : \langle Id, L \rangle ;; \langle E_1 \ldots E_n \rangle \Longrightarrow L ;; \Delta_n$$

  and individually as $c_i = \bar{c} ;; \Pi_i$, and
- a combinator $\mathit{fold}^L : \langle E_1 \ldots E_n \rangle /\!/ \Pi_1 ;; \Delta_n \to \Pi_0 ;; L /\!/ \Pi_1$ obtained from the universal property of $\Pi_0 ;; L /\!/ \Pi_1$ using the 2-cell $\varepsilon ;; U ;; \Pi_1$ in the

following diagram:

$$\begin{tikzcd}
\langle E_1 \dots E_n\rangle//\Pi_1;;\Delta_n \arrow[r, dashed, "\mathit{fold}^L"] \arrow[dr, "U"'] & \Pi_0;;L//\Pi_1 \arrow[d] \\
& \mathbf{A}\times\mathbf{C} \arrow[d, bend right, "\Pi_0;;L"'] \arrow[d, bend left, "\Pi_1"] \arrow[d, phantom, "\Rightarrow"] \\
& \mathbf{C}
\end{tikzcd}$$

We have just described how an adjunction gives rise to the ingredients of an inductive datatype. More generally, we have the following correspondence:

Theorem 4.3.1 *In a 2-category with inserters and finite limits, the following are equivalent:*

1. $(L, c_1, \dots, c_n)$ *is an inductive datatype,*

2. $\ulcorner \bar{c} \urcorner \dashv U;;\Pi_0$ *is a strict coreflection.*

Proof. $(1 \Rightarrow 2)$. First notice that $\ulcorner \bar{c} \urcorner;;U;;\Pi_0 = \mathit{Id}$ since $\ulcorner \bar{c} \urcorner;;U = \langle \mathit{Id}, L\rangle$. Next we show how to obtain the co-unit ε and then verify that the triangle equalities hold. Consider the 2-cell

$$\left\langle U;;\Pi_0, \mathit{fold}^L\{\alpha\}\right\rangle : U;;\Pi_0;;\ulcorner \bar{c} \urcorner;;U \Rightarrow U.$$

We use the (2D) property of $\langle E_1 \dots E_n\rangle//\Pi_1;;\Delta_n$ to obtain the unique 2-cell $\varepsilon : U;;\Pi_0;;\ulcorner \bar{c} \urcorner \Rightarrow \mathit{Id}$ such that

$$\varepsilon;;U = \left\langle U;;\Pi_0, \mathit{fold}^L\{\alpha\}\right\rangle.$$

The equation required by the (2D) property follows from the *(fold-rec)* equations as follows:

$$\begin{aligned}
& (\left\langle U;;\Pi_0, \mathit{fold}^L\{\alpha\}\right\rangle;;\langle E_1 \dots E_n\rangle);\alpha \\
= \quad & \\
& (U;;\Pi_0;;\bar{c});(\mathit{fold}^L\{\alpha\};;\Delta_n) \\
= \quad & \\
& (U;;\Pi_0;;\bar{c};;\alpha);(\left\langle U;;\Pi_0, \mathit{fold}^L\{\alpha\}\right\rangle;;\Pi_1;;\Delta_n).
\end{aligned}$$

It remains to check the triangle equalities. ($\triangle 2$) is $\varepsilon;; U;; \Pi_0 = U;; \Pi_0$ and follows immediately from the (2D) property above. ($\triangle 1$) is $\ulcorner \bar{c} \urcorner;; \varepsilon = \ulcorner \bar{c} \urcorner$ and follows since $\ulcorner \bar{c} \urcorner;; \varepsilon;; U = \ulcorner \bar{c} \urcorner;; U$ and the fact that U is faithful.

(2 $\Rightarrow$ 1). Suppose $\ulcorner \bar{c} \urcorner \dashv U;; \Pi_0$ is a strict coreflection with co-unit ε. Suppose $f_1, \ldots, f_n$ are 2-cells of the form

$$f_i : \langle A, C \rangle ;; E_i \Rightarrow C$$

for each $1 \leq i \leq n$. Consider the 1-cell $\ulcorner \bar{f} \urcorner$ whose codomain is the 0-cell $\langle E_1 \ldots E_n \rangle // \Pi_1;; \Delta_n$. By the universal property of the coreflection (see Lemma 4.2.1), for each 2-cell $id_A : A \Rightarrow \ulcorner \bar{f} \urcorner;; U;; \Pi_0$ there is a unique 2-cell $\ulcorner \bar{f} \urcorner;; \varepsilon : A;; \ulcorner \bar{c} \urcorner \Rightarrow \ulcorner \bar{f} \urcorner$ such that $\ulcorner \bar{f} \urcorner;; \varepsilon;; U;; \Pi_0 = id_A$. If we write $\mathit{fold}^L\{\bar{f}\}$ for $\ulcorner \bar{f} \urcorner;; \varepsilon;; U;; \Pi_1$ then the familiar *(fold-rec)* equations follow by naturality of α as shown here:

$$\begin{aligned}
& (A;; \bar{c}); \mathit{fold}^L\{\bar{f}\} \\
= \quad & \\
& (\ulcorner \bar{f} \urcorner;; U;; \Pi_0;; \ulcorner \bar{c} \urcorner;; \alpha); (\ulcorner \bar{f} \urcorner;; \varepsilon;; U;; \Pi_1;; \Delta_n) \\
= \quad & \\
& (\ulcorner \bar{f} \urcorner;; \varepsilon;; U;; \langle E_1 \ldots E_n \rangle); (\ulcorner \bar{f} \urcorner;; \alpha) \\
= \quad & \\
& (\langle A, \mathit{fold}^L\{\bar{f}\} \rangle ;; \langle E_1 \ldots E_n \rangle); \bar{f}
\end{aligned}$$

□

The situation for a co-inductive datatype is dual to that of an inductive one. This time the inserter is an abstraction of the co-algebra category for the datatype:

$$(\Pi_1;; \Delta_n // \langle E_1 \ldots E_n \rangle, V, \beta)$$

Now asserting the presence of final co-algebras is expressed as a strict reflection of the form

$$\begin{tikzcd}
\Pi_1;; \Delta_n // \langle E_1 \ldots E_n \rangle \arrow[d, bend left, "V;; \Pi_0"] \\
\mathbf{A} \arrow[u, bend left, "\ulcorner \bar{d} \urcorner"] \arrow[u, phantom, "\vdash"]
\end{tikzcd}$$

(i.e. $\ulcorner \bar{d} \urcorner$ is right adjoint to $V;; \Pi_0$) with co-unit the identity (i.e. $\ulcorner \bar{d} \urcorner;; V;; \Pi_0 = \mathit{Id}$) and unit denoted by $\eta : \mathit{Id} \Rightarrow V;; \Pi_0;; \ulcorner \bar{d} \urcorner$. This adjunction gives rise to the following:

- a 1-cell $R : \mathbf{A} \to \mathbf{C}$ (i.e. the new type constructor) given by $\ulcorner \bar{d} \urcorner ;; V ;; \Pi_1$,

- 2-cells $d_i : R \Rightarrow \langle Id, R\rangle ;; E_i$ (for each $1 \leq i \leq n$) (i.e. the new term destructors) collectively given by

$$\bar{d} = \ulcorner \bar{d} \urcorner ;; \beta : R ;; \Delta_n \Longrightarrow \langle Id, R\rangle ;; \langle E_1 \dots E_n\rangle$$

  and individually as $d_i = \bar{d} ;; \Pi_i$, and

- a combinator $\mathit{unfold}^R : \Pi_1 ;; \Delta_n // \langle E_1 \dots E_n \rangle \to \Pi_1 // \Pi_0 ;; R$ obtained from the universal property of $\Pi_1 // \Pi_0 ;; R$ using the 2-cell $\eta ;; V ;; \Pi_1$ in the following diagram:

$$\begin{tikzcd}
\Pi_1 ;; \Delta_n // \langle E_1 \dots E_n \rangle \arrow[r, dashed, "\mathit{unfold}^R"] \arrow[dr, "V"'] & \Pi_1 // \Pi_0 ;; R \arrow[d] \\
& \mathbf{A} \times \mathbf{C} \arrow[d, bend right, "\Pi_1"'] \arrow[d, bend left, "\Pi_0 ;; R"] \arrow[d, phantom, "\Rightarrow"] \\
& \mathbf{C}
\end{tikzcd}$$

The following result is the dual of Theorem 4.3.1.

Theorem 4.3.2 *In a 2-category with inserters and finite limits, the following are equivalent:*

1. $(R, d_1, \dots, d_n)$ *is a co-inductive datatype,*
2. $V ;; \Pi_0 \dashv \ulcorner \bar{d} \urcorner$ *is a strict reflection.*

Chapter 5

Strong Combinators

Datatypes in Charity are strong. This means the type constructors are strong functors. It also means the combinators have certain properties implied by the strength. One such property is apparent in the Charity type theory: the arguments to a "strong combinator" depend on an arbitrary context parameter. For example, recall the general form of the *fold* combinator:

$$\frac{f_1 : V \times E_1(A, C) \to C \quad \cdots \quad f_n : V \times E_n(A, C) \to C}{\mathit{fold}\{f_1 \cdots f_n\} : V \times L(A) \to C}$$

where V represents an arbitrary context of term variables. Another property, which is less apparent, is what parametricity means for such a combinator.

The purpose of this chapter is to formulate a definition of *strong combinator* appropriate to Charity. We proceed by instantiating our 2-categorical combinators to the case when the type constructing 1-cells correspond to strong functors. By viewing strong functors as morphisms of fibrations we obtain a notion of fibred combinators which captures the idea of context parameters and provides an appropriate notion of parametricity.

Throughout this chapter we assume $\mathbf{X}$ is a category with finite products and write $\mathbf{Fib}(\mathbf{X})$ for the 2-category of fibrations over $\mathbf{X}$.

5.1 Inserters in $\mathbf{Fib}(\mathbf{X})$

Suppose $\mathbf{Y} \xrightarrow{p} \mathbf{X}$ and $\mathbf{Z} \xrightarrow{q} \mathbf{X}$ are fibrations over $\mathbf{X}$. Given a pair of fibration morphisms

$$\begin{tikzcd} \mathbf{Y} \arrow[rr, bend left, "F"] \arrow[dr, "p"'] & & \mathbf{Z} \arrow[dl, "q"] \\ & \mathbf{X} & \end{tikzcd} \qquad \begin{tikzcd} \mathbf{Y} \arrow[rr, bend left, "G"] \arrow[dr, "p"'] & & \mathbf{Z} \arrow[dl, "q"] \\ & \mathbf{X} & \end{tikzcd}$$

(also denoted $F, G : p \to q$) we define the category $F//G$ as follows:

- objects are pairs (Y, f) where $Y \in \mathbf{Y}$ and $f : F(Y) \to G(Y)$ is vertical in $\mathbf{Z}$ (i.e. $q(f) = id_{p(Y)}$),
- arrows $h : (Y, f) \to (Y', f')$ are arrows $h : Y \to Y'$ in $\mathbf{Y}$ such that

$$\begin{tikzcd} Y \arrow[d, "h"'] \\ Y' \end{tikzcd} \qquad \begin{tikzcd} F(Y) \arrow[r, "f"] \arrow[d, "F(h)"'] & G(Y) \arrow[d, "G(h)"] \\ F(Y') \arrow[r, "f'"'] & G(Y') \end{tikzcd}$$

  commutes, and
- composition and identities are taken from $\mathbf{Y}$.

Let $U : F//G \to \mathbf{Y}$ be the functor given by

$$[(Y, f) \xrightarrow{h} (Y', f')] \;\; \mapsto \;\; [Y \xrightarrow{h} Y']$$

and let $\alpha : U;; F \Rightarrow U;; G$ be the natural transformation given by

$$\alpha_{(Y,f)} = f : F(Y) \to G(Y).$$

Our goal is to show $(F//G, U, \alpha)$ is an inserter of F and G in $\mathbf{Fib}(\mathbf{X})$. First we must show that $F//G$ is indeed fibered over $\mathbf{X}$ (i.e. it is a 0-cell in $\mathbf{Fib}(\mathbf{X})$) and that U is a 1-cell and α is a 2-cell. The proof of the inserter properties will follow.

Lemma 5.1.1 *$pU : F//G \to \mathbf{X}$ is a fibration.*

Proof. We show how a cleavage for pU can be obtained from cleavages for p and q. For every $(Y, f) \in F//G$ and $u : X \to p(Y)$ in $\mathbf{X}$ we need a cartesian arrow in $F//G$ over u. Let us write

$$\bar{u}(Y) : u^*(Y) \to Y$$

for the cartesian arrow over u given by the cleavage of p for $Y \in \mathbf{Y}$. Let $\bar{u}(Y, f)$ (in $F//G$) be

$$\bar{u}(Y) : (u^*(Y), \alpha_{(u^*(Y,f))}) \to (Y, f)$$

where $\alpha_{(u^*(Y,f))}$ is given by

$$\begin{tikzcd}
F(u^*(Y)) \arrow[d, dashed, "\alpha_{(u^*(Y,f))}"'] \arrow[rrr, bend left, "F(\bar{u}(Y))"] & \cong & u^*(F(Y)) \arrow[r, "\bar{u}(F(Y))"] \arrow[d, dashed, "u^*(f)"'] & F(Y) \arrow[d, "f"] \\
G(u^*(Y)) \arrow[rrr, bend right, "G(\bar{u}(Y))"'] & \cong & u^*(G(Y)) \arrow[r, "\bar{u}(G(Y))"'] & G(Y)
\end{tikzcd}$$

$F(\bar{u}(Y))$ is cartesian over u since F preserves cartesian arrows, and $\bar{u}(F(Y))$ is cartesian over u by the cleavage of q. Hence there is a unique vertical isomorphism $F(u^*(Y)) \cong u^*(F(Y))$. Similarly $G(u^*(Y)) \cong u^*(G(Y))$. $\alpha_{(u^*(Y,f))}$ is then defined as the vertical composite

$$F(u^*(Y)) \xrightarrow{\sim} u^*(F(Y)) \xrightarrow{u^*(f)} u^*(G(Y)) \xrightarrow{\sim} G(u^*(Y)).$$

It remains to show $\bar{u}(Y) : (u^*(Y), \alpha_{(u^*(Y,f))}) \to (Y, f)$ is cartesian in $F//G$. Suppose $h : (Y', f') \to (Y, f)$ is an arrow in $F//G$ and suppose $p(h) = w; u$ for some w in $\mathbf{X}$. Since $\bar{u}(Y) : u^*(Y) \to Y$ is cartesian in $\mathbf{Y}$ there is a unique $h' : Y' \to u^*(Y)$ such that $h'; \bar{u}(Y) = h$ in $\mathbf{Y}$. For $h' : (Y', f') \to$

$(u^*(Y), \alpha_{(u^*(Y,f))})$ to be an arrow in $F//G$ requires commutativity of

$$\begin{tikzcd} F(Y') \arrow[r, "F(h')"] \arrow[d, "f'"'] & F(u^*(Y)) \arrow[d, "\alpha_{(u^*(Y,f))}"] \\ G(Y') \arrow[r, "G(h')"'] & G(u^*(Y)) \end{tikzcd}$$

but this holds since both $F(h'); \alpha_{(u^*(Y,f))}$ and $f'; G(h')$ are candidates for the unique dashed arrow in

$$\begin{tikzcd} & F(Y') \arrow[dr, dashed] \arrow[drr, bend left, "F(h); f"] & & \\ \mathbf{Z} \arrow[dd, "q"'] & & G(u^*(Y)) \arrow[r, "G(\bar{u}(Y))"'] & G(Y) \\ & p(Y') \arrow[dr, "p(h')"'] \arrow[drr, bend left, "p(h)"] & & \\ \mathbf{X} & & p(u^*(Y)) \arrow[r, "u"'] & p(Y) \end{tikzcd}$$

□

Lemma 5.1.2 $U : F//G \to \mathbf{Y}$ *is a morphism of fibrations in*

$$\begin{tikzcd} F//G \arrow[rr, bend left, "U"] \arrow[dr, "pU"'] & & \mathbf{Y} \arrow[dl, "p"] \\ & \mathbf{X} & \end{tikzcd}$$

Proof. We need to show U preserves cartesian arrows. In particular, if

$$c_u : (Y_u, f_u) \to (Y, f)$$

is cartesian in $F//G$ over $u : X \to p(Y)$ in $\mathbf{X}$ then we must show $c_u : Y_u \to Y$ is cartesian in $\mathbf{Y}$. In the previous proof we showed existence of an arrow $\alpha_{(u^*(Y,f))} : F(u^*(Y)) \to G(u^*(Y))$ in $\mathbf{Z}$ making

$$\bar{u}(Y) : (u^*(Y), \alpha_{(u^*(Y,f))}) \to (Y, f)$$

cartesian in $F//G$ over u. Since c_u and $\bar{u}(Y)$ are both cartesian (in $F//G$) over u, there is a unique vertical isomorphism

$$i : (Y_u, f_u) \xrightarrow{\sim} (u^*(Y), \alpha_{(u^*(Y,f))})$$

such that $i; \bar{u}(Y) = c_u$. Being an isomorphism in $F//G$, i is also an isomorphism in $\mathbf{Y}$ and therefore i is cartesian in $\mathbf{Y}$. It follows that the composite of cartesian arrows $i; \bar{u}(Y)$ is cartesian in $\mathbf{Y}$. $\square$

Lemma 5.1.3 *$\alpha : U;; F \Rightarrow U;; G$ is a 2-cell in* $\mathbf{Fib}(\mathbf{X})$.

Proof. Naturality of α is immediate. Furthermore, for each $(Y, f) \in F//G$, $\alpha_{(Y,f)}$ is vertical since f is vertical. $\square$

Having established $F//G$ as a 0-cell (fibred over $\mathbf{X}$ by $U;; p$), U as a 1-cell, and α as a 2-cell we can proceed with the inserter proof.

Theorem 5.1.4 *$(F//G, U, \alpha)$ as defined above is an inserter of F and G in* $\mathbf{Fib}(\mathbf{X})$.

Proof. The proof is a fibred version of the proof of Theorem 3.2.2. For (1D) suppose $\mathbf{J} \xrightarrow{r} \mathbf{X}$ is a 0-cell, $V : r \to p$ is a 1-cell, and $\beta : V;; F \Rightarrow V;; G$ is a 2-cell. The unique 1-cell V' such that $V';; U = V$ and $V';; \alpha = \beta$ is given by

$$[J \xrightarrow{k} J'] \quad \mapsto \quad [(V(J), \beta_J) \xrightarrow{V(k)} (V(J'), \beta_{J'})]$$

To show V' preserves cartesian arrows, suppose $k_u : J \to J'$ is cartesian in $\mathbf{J}$ over u in $\mathbf{X}$. We need to show $V(k_u) : (V(J), \beta_J) \to (V(J'), \beta_{J'})$ is cartesian in $F//G$. Since V is a 1-cell, $V(k_u) : V(J) \to V(J')$ is cartesian in $\mathbf{Y}$. From the proof of Lemma 5.1.1 there is a unique $\alpha_{(u^*(VJ', \beta_{J'}))}$ making

$$V(k_u) : (V(J), \alpha_{(u^*(V(J'), \beta_{J'}))}) \to (V(J'), \beta_{J'})$$

cartesian in $F//G$. We claim β_J is a candidate for $\alpha_{(u^*(VJ',\beta_{J'}))}$ since it is a candidate for the unique arrow in

$$
\begin{tikzcd}
 & FVJ \arrow[dr, dashed] \arrow[drr, bend left, "FVk_u;\beta_{J'}"] & & \\
\mathbf{Z} \arrow[dd, "q"'] & & GVJ \arrow[r, "GVk_u"'] & GVJ' \\
 & \cdot \arrow[dr, "id"'] \arrow[drr, bend left, "u"] & & \\
\mathbf{X} & & \cdot \arrow[r, "u"'] & \cdot
\end{tikzcd}
$$

For (2D), given 1-cells $W_1, W_2 : r \to U;;p$ and a 2-cell $\gamma : W_1;;U \Rightarrow W_2;;U$ such that $F(\gamma_J);\alpha_{W_2(J)} = \alpha_{W_1(J)};G(\gamma_J)$ for each $J \in \mathbf{J}$, the unique $\delta : W_1 \Rightarrow W_2$ such that $\delta;;U = \gamma$ is defined as

$$\gamma_J : (U(W_1(J)), \alpha_{W_1(J)}) \longrightarrow (U(W_2(J)), \alpha_{W_2(J)})$$

for each $J \in \mathbf{J}$. The naturality of γ and the fact that γ is vertical implies the same in δ. $\Box$

5.2 Inserters of Strong Functors

In this section we review the definition of $\mathbf{X}$-strong functor and how such a functor can be viewed as a 1-cell in $\mathbf{Fib}(\mathbf{X})$ (see also [d1] or [jac94]). Using the results of the previous section, we then examine inserters of such 1-cells. The result is a formulation of algebras and homomorphisms appropriate for strong combinators.

For simplicity, we shall work with a fairly concrete notion of strong functor. It should be straightforward to generalize our discussion to the more abstract strong functors in [d1].

Given that $\mathbf{X}$ is a category with finite products, there is an obvious functor

$$\oslash : \mathbf{X} \times \mathbf{X}^k \to \mathbf{X}^k$$

(for each $k \geq 0$) which distributes the binary product of $\mathbf{X}$ as follows:

$$V \oslash (A_1, \ldots, A_k) = (V \times A_1, \ldots, V \times A_k)$$

Definition 5.2.1 *A functor $F : \mathbf{X}^k \to \mathbf{X}$ $(k \geq 0)$ is called a* strong functor *if it comes with a natural transformation*

$$\theta^F_{V,A} : V \times F(A) \to F(V \oslash A)$$

(natural in V, A) such that the next two diagrams commute:

$$\begin{tikzcd}
V \times F(A) \arrow[r, "\theta^F"] \arrow[dr, "p_1"'] & F(V \oslash A) \arrow[d, "F(p_1)"] \\
& F(A)
\end{tikzcd}$$

$$\begin{tikzcd}
V \times (W \times F(A)) \arrow[r, "id \times \theta^F"] \arrow[d, "\cong"'] & V \times F(W \oslash A) \arrow[r, "\theta^F"] & F(V \oslash (W \oslash A)) \arrow[d, "\cong"] \\
(V \times W) \times F(A) \arrow[rr, "\theta^F"'] & & F((V \times W) \oslash A)
\end{tikzcd}$$

A strong functor $F : \mathbf{X}^k \to \mathbf{X}$ can be viewed as a morphism of split fibrations

$$\begin{tikzcd}
s(\mathbf{X}^k) \arrow[rr, bend left, "\widetilde{F}"] \arrow[dr] & & s(\mathbf{X}) \arrow[dl] \\
& \mathbf{X} &
\end{tikzcd}$$

where $s(\mathbf{X}^k)$ (for any $k \geq 0$) is the category defined as follows:

- objects are pairs (V, A) where $V \in \mathbf{X}$ and $A \in \mathbf{X}^k$,
- arrows $(u, h) : (V, A) \to (W, B)$ are pairs of arrows $u : V \to W$ and $h : V \oslash A \to Y$,
- composition is given by $(u, h); (v, k) = (u; v, \langle p_0; u, h \rangle ; k)$,
- identities are (id, p_1).

The category $s(\mathbf{X}^k)$ is fibred over $\mathbf{X}$ by the functor $(u,h) \mapsto u$. The cartesian arrows in this fibration are $(u,p_1) : (V,A) \to (W,A)$ for each $(W,A) \in s(\mathbf{X}^k)$ and $u : V \to W$ in $\mathbf{X}$. The behavior of $\widetilde{F}$ is given as follows:

$$\begin{array}{ccc} (V,A) & & (V,F(A)) \\ {\scriptstyle (u,h)}\big\downarrow & \mapsto & \big\downarrow{\scriptstyle (u,\theta^F;F(h))} \\ (W,B) & & (W,F(B)) \end{array}$$

Preservation of identities and cartesian arrows is clear as $\theta^F;F(p_1) = p_1$. Preservation of composition is more involved (see [d1]).

Now we are ready to describe inserters of strong functors. Suppose $F,G : \mathbf{X}^k \to \mathbf{X}$ are strong functors. The category $\widetilde{F}//\widetilde{G}$ has objects

$$[(V,A),(id,f) : V \times F(A) \to G(A)]$$

and arrows are commuting squares

$$\begin{array}{ccc} V & V \oslash A & \begin{array}{ccc} V \times F(A) & \xrightarrow{\langle p_0, f\rangle} & V \times G(A) \\ {\scriptstyle \langle p_0;u,\theta^F;F(h)\rangle}\big\downarrow & & \big\downarrow{\scriptstyle \theta^G;G(h)} \\ W \times F(B) & \xrightarrow[f']{} & G(B) \end{array} \\ {\scriptstyle u}\big\downarrow & {\scriptstyle h}\big\downarrow & \\ W & B & \end{array}$$

As a split fibration over $\mathbf{X}$, the cartesian arrows in $\widetilde{F}//\widetilde{G}$ are of the form

$$(u,p_1) : [(V,A),(u \times id);f] \longrightarrow [(W,A),f]$$

for each $[(W,A),f] \in \widetilde{F}//\widetilde{G}$ and $u : V \to W$ in $\mathbf{X}$.

5.3 Strong Combinators Defined

Having described inserters of strong functors (as 1-cells in $\mathbf{Fib}(\mathbf{X})$) we now consider combinators between such inserters. Specifically we consider combinators which preserve the cartesian splitting on the nose. By unraveling the 2-categorical concepts we obtain a formal definition of strong combinator, independent of the fibrational machinery. These strong combinators can be expressed in the type theory of Chapter 2.

Definition 5.3.1 *Suppose* $F, G : \mathbf{Y} \to \mathbf{Z}$ *and* $F', G' : \mathbf{Y} \to \mathbf{Z}'$ *are strong functors. Say* $c : F//G \to F'//G'$ is a strong combinator *when for every* $V \in \mathbf{X}, Y \in \mathbf{Y}$*, and* $f : V \times F(Y) \to G(Y)$ *in* $\mathbf{Z}$ *there is a specified arrow* $c\{f\} : V \times F'(Y) \to G'(Y)$ *in* $\mathbf{Z}'$ *such that:*

(SC1) for each $f : V \times F(Y) \to G(Y)$ *and* $u : U \to V$*,*

$$\begin{tikzcd} U \times F'(Y) \arrow[d, "u \times id"'] \arrow[dr, "c\{(u \times id); f\}"] & \\ V \times F'(Y) \arrow[r, "c\{f\}"'] & G'(Y) \end{tikzcd}$$

commutes, and

(SC2) for each $f : V \times F(Y) \to G(Y)$*,* $f' : V \times F(Y') \to G(Y')$*, and* $h : V \times Y \to Y'$*, commutativity of*

$$\begin{tikzcd} V \times F(Y) \arrow[r, "\langle p_0, f \rangle"] \arrow[d, "\langle p_0, \theta^F; F(h) \rangle"'] & V \times G(Y) \arrow[d, "\theta^G; G(h)"] \\ V \times F(Y') \arrow[r, "f'"'] & G(Y') \end{tikzcd}$$

implies commutativity of

$$\begin{tikzcd} V \times F'(Y) \arrow[r, "\langle p_0, c\{f\} \rangle"] \arrow[d, "\langle p_0, \theta^{F'}; F'(h) \rangle"'] & V \times G'(Y) \arrow[d, "\theta^{G'}; G'(h)"] \\ V \times F'(Y') \arrow[r, "c\{f'\}"'] & G'(Y') \end{tikzcd}$$

In the type theory of the Charity term logic a strong combinator can be expressed as a rule of the form

$$\frac{\Gamma \vdash f : F(Y) \to G(Y)}{\Gamma \vdash c\{f\} : F'(Y) \to G'(Y).}$$

The property (SC1) means that substitution commutes with application of c. In other words c is preserved (on the nose) by reindexing functors u^*. Type theoretically this means the following derivations are equivalent:

- $$\dfrac{\dfrac{v : V \vdash f : F(Y) \to G(Y) \qquad u : U \vdash s : V}{u : U \vdash f[s/v] : F(Y) \to G(Y)}}{u : U \vdash c\{f[s/v]\} : F'(Y) \to G'(Y)}$$

- $$\dfrac{\dfrac{v : V \vdash f : F(Y) \to G(Y)}{v : V \vdash c\{f\} : F'(Y) \to G'(Y)} \qquad u : U \vdash s : V}{u : U \vdash c\{f\}[s/v] : F'(Y) \to G'(Y)}$$

The property (SC2) expresses the fibred parametricity of c. In the type theory we have

$$\frac{\Gamma \vdash f; G\{h\} = F\{h\}; f'}{\Gamma \vdash c\{f\}; G'\{h\} = F'\{h\}; c\{f'\}}$$

for each $\Gamma \vdash f : F(Y) \to G(Y)$, $\Gamma \vdash f' : F(Y') \to G(Y')$, and $\Gamma \vdash h : Y \to Y'$.

The verification that our strong combinators correspond to the appropriate 2-categorical combinators is given by the following theorem.

Theorem 5.3.2 *The following are equivalent:*

1. $c : F//G \to F'//G'$ *is a strong combinator,*
2. $c : \widetilde{F}//\widetilde{G} \longrightarrow \widetilde{F}'//\widetilde{G}'$ *is a (split) combinator in* $\mathbf{Fib}(\mathbf{X})$.

Proof. (1) $\Rightarrow$ (2). The object map of the split functor c is specified by the arrow construction

$$\frac{f : V \times F(Y) \to G(Y)}{c\{f\} : V \times F'(Y) \to G'(Y).}$$

The arrow map is specified by $(u, h) \mapsto (u, h)$ and is well-defined as

$$\begin{array}{rcl}
\langle p_0, f\rangle ; \theta^G; G(h) & = & \left\langle p_0; u, \theta^F; F(h)\right\rangle ; f' \\
& \Leftrightarrow & \\
\langle p_0, f\rangle ; \theta^G; G(h) & = & \left\langle p_0, \theta^F; F(h)\right\rangle ; (u \times id); f' \\
& \Rightarrow & \text{(SC2)} \\
\langle p_0, c\{f\}\rangle ; \theta^{G'}; G'(h) & = & \left\langle p_0, \theta^{F'}; F'(h)\right\rangle ; c\{(u \times id); f'\} \\
& \Leftrightarrow & \text{(SC1)} \\
\langle p_0, c\{f\}\rangle ; \theta^{G'}; G'(h) & = & \left\langle p_0, \theta^{F'}; F'(h)\right\rangle ; (u \times id); c\{f'\} \\
& \Leftrightarrow & \\
\langle p_0, c\{f\}\rangle ; \theta^{G'}; G'(h) & = & \left\langle p_0; u, \theta^{F'}; F'(h)\right\rangle ; c\{f'\}
\end{array}$$

Preservation of identities, composition, and the cartesian splitting is clear.

(2) $\Rightarrow$ (1). The object map of the split functor gives the arrow construction. (SC1) follows from the preservation of the cartesian splitting. (SC2) follows from the arrow maps of c restricted to each fibre. $\Box$

5.4 Strong Combinators in Charity

With a definition of strong combinator in hand, we return to the question of how to interpret polymorphic Charity functions as strong combinators. Essentially we seek a polymorphic Charity type theory which is a generalization of the term logic type theory of Chapter 2. Recall from Chapter 2 the form of our judgements was

$$v : V \vdash \varphi$$

where $v : V$ is a context of term variables and φ is a function which is well-formed with respect to $v : V$. Adding explicit polymorphism to this type theory would facilitate the derivation of judgements of the form

$$A : \text{type} \mid f : S \to T \mid v : V \vdash \varphi$$

where $A : \text{type}$ is a type variable context, $f : S \to T$ is a function variable context, $v : V$ is a term variable context, and φ is a function well-formed with respect to the combined context. Such a type theory could be obtained by adding type variables and function variables to the term logic type theory of Chapter 2. The next question to consider is how such a type theory could be interpreted categorically.

In Chapter 2, the term logic type theory was shown to have an interpretation in the simple fibration. Our 2-categorical type theory of Chapter 3 (in which contexts contained only type variables and function variables) was shown to have a fibrational interpretation as well. The question of which fibration is appropriate for a polymorphic Charity type theory, however, must be answered more carefully. This is because there are implicit dependencies between term variables and function variables which do not become apparent until substitution of these variables is performed. The structure which is appropriate to capture such dependencies seems to be a layering of fibrations where each additional variable in the context corresponds to an additional layer. At this point, we have not pursued this direction very far but there is a potential for future work here.

Another question which we have considered is how individual polymorphic Charity functions can be interpreted as strong combinators. Using the definition facility of the Charity interpreter, one can define a polymorphic function of the form

$$\begin{aligned} \mathrm{def}\ c\{f : S_0(A) \to T_0(A)\} : S_1(A) \to T_1(A) \\ = \ \{v \mapsto t\} \end{aligned}$$

where $\{v \mapsto t\}$ is a closed abstraction depending only on a choice of function parameter f and type parameter A. In the Charity interpreter, such polymorphic functions are translated to categorical combinators. This translation is based on the original term logic translation defined in [d2], but is extended to handle function parameters. An essential property of these function parameters is that they can depend on potentially *any* term variable context. This dependency is handled in the extended translation by regarding function parameters as functions in context. This subtlety turns out to be exactly what is needed for polymorphic Charity functions to be viewed as strong combinators. In other words, the translation used by the Charity interpreter *is* how polymorphic Charity functions are to be formally interpreted as strong combinators.

Chapter 6

Conclusion

In this final chapter we summarize the contributions of this thesis and consider some potential directions for future work.

6.1 Summary

In this thesis we have shown how categorical combinators like those used in Charity can be expressed 2-categorically. The specific contributions include:

- a type theoretic formulation of the Charity term logic along with a fibrational semantics (Chapter 2),
- a 2-categorical definition of *combinator* based on inserters which captures parametricity for strictly covariant types, and a type theory and fibrational semantics for 2-categorical polymorphic combinator expressions (Chapter 3),
- a formulation of 2-categorical datatypes (rectified from [d1]) which is shown to be compatible with our 2-categorical combinators and parametricity (Chapter 4), and
- a formulation of *strong combinator* based on the instantiation of 2-categorical combinators to morphisms of split fibrations between inserters of strong functors (Chapter 5).

6.2 Future Work

Perhaps the most obvious direction for future work is to continue the pursuit of a polymorphic Charity type theory along with an appropriate fibrational semantics. We have begun to consider this direction in 5.4.

Another possible direction is to consider applications of parametricity. Knowing that all polymorphic functions in Charity satisfy parametricity allows us to generate "free theorems" about these functions just by looking at their types. For example, suppose

$$\mathit{sort}\{p : (A \times A) \to \mathit{Bool}\} : \mathit{List}(A) \to \mathit{List}(A)$$

is a polymorphic Charity function. Then the following theorem must hold: for all $p : (A \times A) \to \mathit{Bool}$, $q : (B \times B) \to \mathit{Bool}$, and $h : A \to B$,

$$p = (h \times h); q \quad \Rightarrow \quad \mathit{sort}\{p\}; \mathit{List}\{h\} = \mathit{List}\{h\}; \mathit{sort}\{q\}$$

In other words, whenever p and q induce related orderings on A and B (i.e. related by a map h), $\mathit{sort}\{p\}$ and $\mathit{sort}\{q\}$ will take related inputs to related outputs. Such free theorems have been investigated for the polymorphic lambda calculus (see e.g. [wad89],[pa93],[acc93]) and are a potential source for investigation in Charity.

A third direction of interest is the relationship between our 2-categorical combinators and those of the lambda calculus and combinatory logic ([hls72], [hin85]). These latter combinators may be viewed as categorical arrow constructions but their parametricity is complicated by the potential mixed variances of the type constructors. This complication may be surmountable for fragments of the polymorphic lambda calculus in which free type variables can only occur positively in type expressions [jay96].

Bibliography

[acc93] M. Abadi, L. Cardelli, and P.L. Curien, Formal parametric polymorphism, *Theoretical Computer Science* **121** (1993) 9–58.

[bfss90] E.S. Bainbridge, P.J. Freyd, A. Scedrov, and P.J. Scott, Functorial polymorphism, *Theoretical Computer Science* **70** (1990) 35–64.

[bor94] F. Borceux, Handbook of Categorical Algebra 1, Basic Category Theory (Cambridge University Press, 1994).

[bw88] R. Bird and P. Wadler, Introduction to Functional Programming (Prentice Hall, 1988).

[cf92] J.R.B. Cockett and T. Fukushima, About Charity, Research Report No. 92/480/18, Dept. of Computer Science, University of Calgary, 1992.

[d1] J.R.B. Cockett and D. Spencer, Strong categorical datatypes I, in: R.A.G. Seely, ed., *International Meeting on Category Theory 1991, Canadian Mathematical Society Proceedings*, Vol. 13, AMS, Montreal (1992) 141–169.

[d2] J.R.B. Cockett and D. Spencer, Strong categorical datatypes II: A term logic for categorical programming, *Theoretical Computer Science* **139** (1995) 69–113.

[ft96] T. Fukushima and C. Tuckey, Charity User Manual (draft), Jan., 1996.

[hag87] T. Hagino, A Categorical Programming Language. Ph.D. Thesis, University of Edinburgh, 1987.

[has94] R. Hasegawa, Categorical data types in parametric polymorphism, *Math. Struct. in Comp. Science* **4** (1994) 71–109.

[hin85] J.R. Hindley, Combinators and Lambda Calculus, in *Combinators and Functional Programming Languages*, Lecture Notes in Computer Science **242** (Springer-Verlag, 1985) 104–122.

[hls72] J.R. Hindley, B. Lercher, and J.P. Seldin, Introduction to Combinatory Logic (Cambridge University Press, 1972).

[jac94] B. Jacobs, Categorical Logic and Type Theory (book in preparation, to appear in 1997).

[jay96] B. Jay, Covariant Types, (unpublished paper, 1996).

[kel89] G.M. Kelly, Elementary observations on 2-categorical limits, *Bulletin of the Australian Mathematical Society* **39** (1989) 301–317.

[ks74] G.M. Kelly and R. Street, Review of the elements of 2-categories, *Lecture Notes in Mathematics* **420** (Springer-Verlag, 1974) 75–103.

[pa93] G. Plotkin and M. Abadi, A Logic for Parametric Polymorphism, *Lecture Notes in Computer Science* **664** (Springer-Verlag, 1993) 361–375.

[rey83] J.C. Reynolds, Types, abstraction, and parametric polymorphism, in: R.E.A. Mason, ed., *Information Processing 83* (North-Holland, Amsterdam, 1983) 513–523.

[rob92] E. Robinson, Notes on the Second-Order Lambda Calculus, University of Sussex, Computer Science Report 4/92, August 1992.

[spe93] D.L. Spencer, Categorical Programming with Functorial Strength. Ph.D. Thesis, Oregon Graduate Institute, 1993.

[tho91] S. Thompson, Type Theory and Functional Programming, (Addison Wesley, 1991).

[wad89] P. Wadler, Theorems for free! in *Proc. 4th Internat. Symp. on Functional Programming Languages and Computer Architecture* (Springer, Berlin, 1989).