A Calculational Approach to Strong Datatypes*

Alberto Pardo†
Technische Hochschule Darmstadt, Germany

Abstract

This paper describes calculational properties of *fold*, a recursive functional on inductive types that extends the standard *catamorphism* [MFP91] to global parameters. As described by Cockett and Spencer [CS91] fold is definable for those inductive types that are *strong* in the sense of being given by an algebra that happens to be *initial with parameters*. The laws for fold are presented in two groups. The first one corresponds to standard laws for catamorphisms now adapted to folds when these are regarded as catamorphisms in a category of so-called X-actions (for X an object of parameters). The second group contains laws that describe the combination of folds and the interaction between folds and catamorphisms.

1 Introduction

In programming semantics, a recursive type is understood as a solution of a recursive type equation. Least fixpoints of covariant recursive type equations correspond to the so-called *inductive types* which contain only finite elements, like natural numbers, lists or trees. On the other hand, infinite data structures, such as streams, are given by the greatest fixpoints and are called *co-inductive types*. From an algebraic (and categorical) perspective least/greatest solutions of recursive type equations correspond to the notions of initial algebras/final coalgebras. This algebraic formulation of recursive types has demonstrated to be very useful not only for semantics, but also for program development (particularly in functional programming). As a consequence of the properties of initiality/finality in the category of algebras/coalgebras it is possible to derive general recursive functionals for datatypes, such as catamorphism [MFP91], primitive recursion [Geu92] or even case analysis [CF92] (and their duals). Programming with a fixed set of recursive functionals derived from type definitions leads to a structural programming discipline which can be exploited to facilitate program calculation. By initiality or finality one can state calculational laws for recursive functionals to be used in the derivation, transformation and general reasoning (see e.g. [Mal90, Fok92, Mee92, Jeu93]).

Following these lines, this work presents the derivation of a battery of calculational laws involving a recursive functional on inductive types called *fold*. A fold is an extension of a *catamorphism*. Catamorphism is another functional on inductive types that corresponds to a generic version of the well-known foldr (or reduce) operator for lists by now standard in functional programming [BW88, Wik87]. Although fold and catamorphism share similar function definitions by structural recursion, the difference between them is that the former manipulates explicitely further constant parameters while the latter doesn't. Indeed, fold may be regarded as a 'catamorphism with parameters'. This minor difference between both functionals becomes more tangible as soon as one tries to represent categorically functions that require parameters. In fact, catamorphisms turn out to be (in general) incapable of representing such functions in a first-order setting. On the other hand, as Cockett and Spencer [CS91] showed one can achieve a definition of fold for any inductive type that is *strong*, i.e. that is given by an algebra which is *initial with parameters* [CS91, Jac95].

*This work has been supported by a DAAD scholarship.
†Fachbereich Informatik, Technische Hochschule Darmstadt, Magdalenenstr. 11c, 64289 Darmstadt, Germany; pardo@informatik.th-darmstadt.de

The theory of strong datatypes has been used as the basis for the design of the (categorical) programming language CHARITY [CF92]. However, a calculational theory for functionals on strong datatypes has still remained undeveloped. In this light, our main contribution consists of the identification of the relevant calculational properties for fold.

Following a dual reasoning it is easy to achieve the derivation of similar calculational properties for *unfold* [CS91], a functional dual of fold defined on co-inductive types. However, in this paper we restrict ourselves due to space limitations to the single treatment of fold. The laws for unfold can be found in [Par97].

The paper is organized as follows: We begin with a preliminary section containing background material on the categorical approach to recursive datatypes and program calculation. Only some familiarity with basic notions of category theory is necessary to follow the paper; most of the concepts used appear in e.g. [BW90]. Section 3 introduces the theory of strong datatypes and the definition of fold. In section 4 we formulate the calculational properties for fold. The laws are grouped in two sets. One set contains the standard laws for catamorphisms now adapted to folds. These laws are obtained from the observation that folds can be regarded as catamorphisms in a category of so-called X-actions, where X is an object of parameters. The other set essentially contains laws that specify how fold interacts with other folds and with catamorphisms. Finally, section 5 concludes the paper and describes further work.

2 Recursive Datatypes Categorically

In this section we explain the relevant notions behind the categorical approach to recursive datatypes [Geu92, How92] and its application to program calculation [Mal90, MFP91, Fok92, Jeu93]. We begin introducing some notation. Subsequently we describe the notions of algebra and inductive type, type functor and polynomial functor.

Preliminaries. Throughout, we shall assume a working category $\mathcal{C}$ with finite products $(\times, 1)$ and finite coproducts $(+, 0)$, where $0/1$ denotes the initial/final object of the category. A typical category with this characteristics is **Sets** —the category of sets and total functions— in which 0 and 1 are given by empty and singleton set, respectively. For the product we write $\mathsf{fst} : A \times B \to A$ and $\mathsf{snd} : A \times B \to B$ to denote the (left and right) projections. The pairing of two arrows $f : C \to A$ and $g : C \to B$ is written $\langle f, g \rangle : C \to A \times B$. In **Sets**, $\langle f, g \rangle\, c = (f\, c, g\, c)$. Product associativity is a natural isomorphism denoted by $a_{A,B,C} : A \times (B \times C) \to (A \times B) \times C$. We write $\mathsf{inl} : A \to A + B$ and $\mathsf{inr} : B \to A + B$ for sum inclusions. For $f : A \to C$ and $g : B \to C$, case analysis is given by $[f, g] : A + B \to C$, such that $[f, g] \circ \mathsf{inl} = f$ and $[f, g] \circ \mathsf{inr} = g$. A natural transformation between endofunctors $F, G : \mathcal{C} \to \mathcal{C}$ (i.e functors from $\mathcal{C}$ to itself) is a family of arrows $\sigma_A : FA \to GA$ indexed by the objects of $\mathcal{C}$ such that, for each $f : A \to B$, $Gf \circ \sigma_A = \sigma_B \circ Ff$ (the validity of this equation will be referred to as the *naturality condition* on σ).

Algebras. The category-theoretic explanation of recursive types is based on the idea that types constitute objects of a category $\mathcal{C}$, and type constructors are functors on $\mathcal{C}$. Given a recursive type definition in a functional language such as Haskell or ML it is always possible to derive an endofunctor $F : \mathcal{C} \to \mathcal{C}$ that captures the recursive shape (or signature) of the datatype. Thus, the recursive type turns out to be an object T such that $T \cong FT$, i.e. a fixpoint of F. Least fixpoints of covariant functors give rise to *inductive types* which can be understood as initial functor-algebras, a generalization of the usual notion of term algebras over a given signature. For an endofunctor F, an F-*algebra* in $\mathcal{C}$ is an object A of $\mathcal{C}$ (called the carrier) together with a morphism $h : FA \to A$ (called the operation). A morphism of algebras, or F-*homomorphism*, between $h : FA \to A$ and $k : FB \to B$ is a morphism $f : A \to B$ between the carriers that commutes with the operations, $f \circ h = k \circ Ff$. The category of F-algebras in $\mathcal{C}$ is formed by considering F-algebras as objects and F-homomorphisms as morphisms. Composition and identities in this category are inherited from $\mathcal{C}$. An inductive type together with an interpretation of its *constructors* is given (up to isomorphism) by an *initial* F-algebra, i.e. an initial object in the category of algebras. A well-known result states that any initial algebra $h : FT \to T$ is an isomorphism. In the sequel we fix a representative of the initial F-algebra that we denote by $in_F : F\,\mu F \to \mu F$.

For instance, given a datatype for natural numbers,

$$\mathsf{Nat} = \mathit{Zero} \mid \mathit{Succ}\ \mathsf{Nat}$$

its signature is captured by a functor $N : \mathcal{C} \to \mathcal{C}$ such that $N\,A = 1 + A$ and $N\,f = \mathsf{id}_1 + f$. It turns out that any N-algebra is a case analysis morphism $[h_Z, h_S] : 1 + A \to A$, with $h_Z : 1 \to A$ and $h_S : A \to A$.[1] In particular, the initial algebra is given by $[\mathit{Zero}, \mathit{Succ}] : 1 + \mathsf{Nat} \to \mathsf{Nat}$ with $\mathit{Zero} : 1 \to \mathsf{Nat}$ and $\mathit{Succ} : \mathsf{Nat} \to \mathsf{Nat}$; Nat stands for μN.

Initiality permits to associate a functional with each inductive type which is used to represent functions defined by structural recursion on that type. Usually called *catamorphism* [MFP91],[2] this functional is given by the unique homomorphism from the initial F-algebra to any other. That is, for any F-algebra $h : FA \to A$, catamorphism is the unique morphism $(\!|h|\!)_F : \mu F \to A$ that enjoys this equation:

$$(\!|h|\!)_F \circ in_F = h \circ F(\!|h|\!)_F$$

Note that it recursively replaces the constructors of the datatype by the target algebra h. For example, for the natural numbers a catamorphism on a N-algebra $[h_Z, h_S] : 1 + A \to A$ is the unique arrow $f : \mathsf{Nat} \to A$ such that, $f \circ \mathit{Zero} = h_Z$ and $f \circ \mathit{Succ} = h_S \circ f$.

In the field of program transformation it has been exploited that catamorphisms impose an inductive structure on the functions they represent. For instance, many useful calculational laws for catamorphisms can be derived by using initiality. The simplest one is called *cata-identity* which states that a catamorphism on the constructors —i.e. on the initial algebra— is the identity.

$$(\!|in_F|\!)_F = \mathsf{id}_{\mu F}$$

A law that plays an important role in program calculation is the so-called *cata-fusion*. It states that the composition of a catamorphism with an algebra homomorphism is again a catamorphism.

$$f \circ h = g \circ F f \;\Rightarrow\; f \circ (\!|h|\!)_F = (\!|g|\!)_F$$

Type Functors. By fixing the first argument of a bifunctor $F : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ one can obtain a (parameterized) functor $F(A, -)$, which shall be denoted as F_A, such that $F_A\,B = F(A, B)$ and $F_A\,f = F(\mathsf{id}_A, f)$. Parameterized functors are useful to model the signature of parameterized datatypes. For instance, lists with elements over A are given by the initial algebra $[\mathit{Nil}, \mathit{Cons}] : 1 + A \times \mathsf{List}\,A \to \mathsf{List}\,A$ of the functor $L_A\,B = 1 + A \times B$, where $\mathit{Nil} : 1 \to \mathsf{List}\,A$ and $\mathit{Cons} : A \times \mathsf{List}\,A \to \mathsf{List}\,A$. A catamorphism on an algebra $[h_N, h_C] : 1 + A \times B \to B$ corresponds to the usual foldr operator as known in the functional programming folklore [BW88]. In **Sets** it is the unique function $f : \mathsf{List}\,A \to B$ such that: $f\ \mathit{Nil} = h_N$ and $f\ (\mathit{Cons}\,(a, as)) = h_C\ (a, f\ as)$.

Like for lists, every parameterized functor F_A gives rise to a parameterized datatype D given by the initial F_A-algebra $in_{F_A} : F_A(DA) \to DA$, where $DA = \mu F_A$. Thus D is a type constructor, which can be made into a functor $D : \mathcal{C} \to \mathcal{C}$ —called a *type functor*— by defining its mapping on arrows: For $f : A \to B$, $Df : DA \to DB$ is given by $Df = (\!|in_{F_B} \circ F(f, \mathsf{id}_{DB})|\!)_{F_A}$. For example, by expanding the definition of $\mathsf{List}\,f = (\!|[\mathit{Nil}, \mathit{Cons} \circ (f \times \mathsf{id})]|\!)_{L_A}$ we get the definition of the usual map function on lists [BW88]: $\mathsf{map}(f)\ \mathit{Nil} = \mathit{Nil}$ and $\mathsf{map}(f)\ (\mathit{Cons}(a, as)) = \mathit{Cons}(f\,a, \mathsf{map}(f)\ as)$.

An interesting property on type functors is the *map-cata-fusion* law which states that the result of mapping a morphism $f : A \to B$ with a type functor D (induced by a bifunctor F), composed with a catamorphism on the algebra $h : F_B\,C \to C$ is again a catamorphism.

$$(\!|h|\!)_{F_B} \circ Df = (\!|h \circ F(f, \mathsf{id}_C)|\!)_{F_A}$$

[1] To understand the functorial representation of the signature it is essential to point out that constants of any type A are regarded as morphisms of the form $1 \to A$ categorically.

[2] It may be found under other names in the literature, like *generalised fold* [MJ95] or *iterator* [How92].

Polynomial Functors. The simplest example of a functor is the identity functor I. $\underline{C}$ denotes the constant functor for each object C, such that $\underline{C}\,A = C$ and $\underline{C}\,f = \mathsf{id}_C$. We can make new functors from others by functor composition which is denoted by juxtaposition. The class of so-called *polynomial functors* is inductively defined by the following grammar:

$$F ::= I \mid \underline{C} \mid F \times F \mid F + F \mid FF$$

where $F \times G$ and $F + G$ stand for the product and coproduct of two functors F and G, respectively, such that $(F \dagger G)\,A = FA \dagger GA$, for $\dagger \in \{\times, +\}$.

3 Strong Datatypes

It is common to find recursive functions on datatypes that require extra *parameters* for their computation. However, these functions can not always be represented as a catamorphism since that only admits the involved datatype as input. It turns out that the ability of representing functions with parameters mainly depends on the features of the setting in which a catamorphism is embedded. For instance, within a **higher-order** language we may use *currying*. A recursive function that requires parameters may be given as a curried definition by recursion on the datatype that yields a function on the parameters as result, something that is common practice in functional programming. In categorical terms, a higher-order language is modelled by a *cartesian closed category* in which, for every pair of objects A and B, there is an exponential (or functional) object $[A \to B]$ satisfying an universal property (see e.g. [BW90]). To see an example let us consider functions on the natural numbers. In **Sets**, given a constant $h_Z : A$ and a function $h_S : A \to A$ the natural numbers catamorphism is the unique function $f : \mathsf{Nat} \to A$ such that, $f\ \mathit{Zero} = h_Z$ and $f\,(\mathit{Succ}\ n) = h_S(f\ n)$. Thus, by a higher-order catamorphism of type $\mathsf{Nat} \to [X \to A]$ we can represent a recursive function that admits an *object of parameters* X and returns a value of type A, for certain $h_Z : [X \to A]$ and $h_S : [X \to A] \to [X \to A]$. For instance, this is the case for addition $\mathit{add} : \mathsf{Nat} \to [\mathsf{Nat} \to \mathsf{Nat}]$ with $X = A = \mathsf{Nat}$, $h_Z = \lambda n.n$, and $h_S(g) = \lambda n.\mathit{Succ}\,(g\ n)$. In lining we obtain the curried recursive definition: $\mathit{add}\ \mathit{Zero}\ n = n$ and $\mathit{add}\ (\mathit{Succ}\ m)\ n = \mathit{Succ}\,(\mathit{add}\ m\ n)$.

On the contrary, working in a **first-order** language, the absence of the currying feature leads to the necessity of modifying the definition of catamorphism in order to explicitly manage parameters. The new functional —called *fold*— is then introduced as a morphism of two variables $f : T \times X \to A$ by recursion on T (the inductive type) that uses the second variable as parameter. For instance, in **Sets**, given the functions $h_Z : X \to A$ and $h_S : A \times X \to A$, the natural numbers fold is the unique function $f : \mathsf{Nat} \times X \to A$ such that:

$$f\,(\mathit{Zero}, x) = h_Z(x) \qquad\qquad f\,(\mathit{Succ}\ n, x) = h_S(f(n,x), x) \tag{1}$$

Now, addition is given by $\mathit{add} : \mathsf{Nat} \times \mathsf{Nat} \to \mathsf{Nat}$ with $X = A = \mathsf{Nat}$, $h_Z(n) = n$, and $h_S(m,n) = \mathit{Succ}\,(m)$. In-lining we obtain the traditional number-theoretic definition: $\mathit{add}\,(\mathit{Zero}, n) = n$ and $\mathit{add}\,(\mathit{Succ}\ m, n) = \mathit{Succ}\,(\mathit{add}\,(m,n))$.

Cockett and Spencer [CS91] addressed the definition of fold in a not necessarily closed category by introducing the concept of strong initiality which, in turn, is based on the concept of a strong functor. In fact, initiality with parameters can be stated only for those datatypes whose signature is captured by a so-called strong functor. In the remainder of this section we build up the defintion of fold that best serves the subsequent derivation of its calculational properties.

X-actions. Any fold $f : T \times X \to A$ on an object of parameters X can be regarded as a special case of a so-called X-*action*, a term that will be used to describe morphisms in $\mathcal{C}$ of the form $A \times X \to B$, for arbitrary objects A and B. In particular, it is possible to recognize a category in which X-actions cohabit.

Definition 3.1 For a fixed object X of $\mathcal{C}$, the category $\mathcal{C}_X$ of X-actions is such that: the objects of $\mathcal{C}_X$ are those of $\mathcal{C}$; morphisms between objects A and B of $\mathcal{C}_X$ correspond to X-actions $A \times X \to B$ in $\mathcal{C}$; composition in $\mathcal{C}_X$ is given by $f \bullet g = f \circ \langle g, \mathsf{snd}\rangle$, for $f : B \times X \to C$ and $g : A \times X \to B$; and the identity on A in $\mathcal{C}_X$ is given by $\mathsf{fst} : A \times X \to A$.

Jacobs [Jac95] refers to this category as the *simple slice category*. Technically speaking, category $\mathcal{C}_X$ corresponds to the *Kleisli category* of the product comonad given by functor $MA = A \times X$ (see e.g. [BS93]). It is possible to establish a *lifting* functor $\widehat{(-)} : \mathcal{C} \to \mathcal{C}_X$ between $\mathcal{C}$ and the category of X-actions in the obvious way. On objects, $\widehat{A} = A$, while on arrows, $\widehat{f} = f \circ \mathsf{fst} : A \times X \to B$, for $f : A \to B$. For example, for each object A of $\mathcal{C}_X$, the identity on A in $\mathcal{C}_X$ is the lifting of the identity on A in $\mathcal{C}$, since $\widehat{\mathsf{id}_A} = \mathsf{id}_A \circ \mathsf{fst} = \mathsf{fst}$. Concerning the composition of X-actions, by product properties it follows that: (i) for any $f : B \times X \to C$ and $g : A \to B$, $f \bullet \widehat{g} = f \circ (g \times \mathsf{id}_X)$; and (ii) for any $f : B \to C$ and $g : A \times X \to B$, $\widehat{f} \bullet g = f \circ g$.

Strong Functors. As Cockett and Spencer [CS91] observed, in order to achieve a definition of fold it is sufficient to assume that: (i) the underlying category $\mathcal{C}$ is cartesian (i.e. has a product and a final object), but not necessarily closed; and (ii) the signature of datatypes is given by strong functors. If $A = (A_1, \ldots, A_m)$ is an object of $\mathcal{C}^m$ and B an object of $\mathcal{C}$, then let us define $A \times B = (A_1 \times B, \ldots, A_m \times B)$.

Definition 3.2 A functor $F : \mathcal{C}^m \to \mathcal{C}$ is *strong* if it is equipped with a natural transformation $\tau^F_{A,X} : FA \times X \to F(A \times X)$, called a *strength*, such that the following two diagrams commute:

$$\begin{tikzcd}
FA \times X \arrow[r, "\tau^F_{A,X}"] \arrow[dr, "\mathsf{fst}"'] & F(A \times X) \arrow[d, "F\,\mathsf{fst}"] \\
& FA
\end{tikzcd}
\qquad
\begin{tikzcd}
FA \times (X \times Y) \arrow[rr, "\tau^F_{A,X\times Y}"] \arrow[d, "a"'] & & F(A \times (X \times Y)) \arrow[d, "Fa"] \\
(FA \times X) \times Y \arrow[r, "\tau^F_{A,X} \times \mathsf{id}_Y"'] & F(A \times X) \times Y \arrow[r, "\tau^F_{A\times X,Y}"'] & F((A \times X) \times Y)
\end{tikzcd}$$

For instance, polynomial functors turn out to be strong under the additional assumption that category $\mathcal{C}$ is distributive. A category is called *distributive* if it has finite products and coproducts, such that binary products distribute over finite coproducts. This means that: (i) the canonical map $[\mathsf{inl} \times \mathsf{id}_C, \mathsf{inr} \times \mathsf{id}_C] : A \times C + B \times C \to (A + B) \times C$ is an isomorphism whose inverse is the natural transformation denoted by $d_{A,B,C} : (A+B) \times C \to A \times C + B \times C$; and (ii) for each object A, the unique canonical map $0 \to 0 \times A$ is an isomorphism. Again, a typical example is **Sets**.

By reasoning inductively on functor construction, below we present a definition of strength for any polynomial functor.

Definition 3.3

$$\begin{array}{lll}
\tau^C_{A,X} & = \mathsf{fst} & : C \times X \to C \\
\tau^I_{A,X} & = \mathsf{id}_{A\times X} & : A \times X \to A \times X \\
\tau^{F\times G}_{A,X} & = (\tau^F_{A,X} \times \tau^G_{A,X}) \circ \langle \mathsf{fst} \times \mathsf{id}_X, \mathsf{snd} \times \mathsf{id}_X \rangle & : (FA \times GA) \times X \to F(A \times X) \times G(A \times X) \\
\tau^{F+G}_{A,X} & = (\tau^F_{A,X} + \tau^G_{A,X}) \circ d_{FA,GA,X} & : (FA + GA) \times X \to F(A \times X) + G(A \times X) \\
\tau^{FG}_{A,X} & = F\tau^G_{A,X} \circ \tau^F_{GA,X} & : FGA \times X \to FG(A \times X)
\end{array}$$

It is easy to check that the τ's above defined indeed satisfy the diagrams of Definition 3.2. A definition of strength for type functors can also be specified, but it requires the use of fold. So, we come back to this issue after having introduced fold.

Strong functors can be lifted to work on X-actions. Given a strong functor $F : \mathcal{C} \to \mathcal{C}$, a *lifting* $F^X : \mathcal{C}_X \to \mathcal{C}_X$ is a functor given by, on objects $F^X A = FA$, while on arrows $F^X f = Ff \circ \tau^F_{A,X} : FA \times X \to FB$, for $f : A \times X \to B$.

Definition 3.4 A *strong natural transformation* between two strong functors F and G is a natural transformation $\sigma_A : FA \to GA$ that behaves consistently with respect to the strengths. That is, $\tau^G_{A,X} \circ (\sigma_A \times \mathsf{id}_X) = \sigma_{A\times X} \circ \tau^F_{A,X}$.

Lemma 3.5 For X fixed, the strength $\tau^F_{A,X} : FA \times X \to F(A \times X)$ of any strong functor F turns out to be a strong natural transformation from the functor $F \times \underline{X}$ to $F(I \times \underline{X})$.

Strongly Initial Datatypes. To define fold we proceed in a similar way as we did with catamorphism, but now requiring initiality in a category of algebras in $\mathcal{C}_X$. Given a functor $G : \mathcal{C}_X \to \mathcal{C}_X$ we can consider the category of G-algebras in $\mathcal{C}_X$. In particular, let us take the lifting F^X of a strong functor for G in the sequel. Thus, a F^X-algebra in $\mathcal{C}_X$ is a X-action $h : FA \times X \to A$.[3] A F^X-homomorphism between $h : FA \times X \to A$ and $k : FB \times X \to B$ is a X-action $f : A \times X \to B$ such that, $f \bullet h = k \bullet F^X f$. In $\mathcal{C}$, this corresponds to the equation, $f \circ \langle h, \mathsf{snd}\rangle = k \circ \langle Ff \circ \tau^F_{A,X}, \mathsf{snd}\rangle$. Having arrived at this point it is natural to ask for initiality in the category of F^X-algebras in $\mathcal{C}_X$. But it is even more interesting to ask for so-called strong initiality.

Definition 3.6 For F strong, an initial F-algebra in_F is said to be *strongly initial* if for each object X, its lifting $\widehat{in_F} = in_F \circ \mathsf{fst} : F\mu F \times X \to \mu F$ happens to be an initial algebra in the category of F^X-algebras.

It is easy to verify that $\widehat{in_F}$ is an isomorphism, implied by the fact that in_F is an isomorphism as well. Consequently, it can safely play the role of an initial F^X-algebra. The satisfaction of strong initiality means the existence and uniqueness of a F^X-homomorphism denoted by $\mathsf{fold}_F(h) : \mu F \times X \to A$ from $\widehat{in_F}$ to any other F^X-algebra $h : FA \times X \to A$. That is, if it exists, $\mathsf{fold}_F(h)$ is the unique X-action that makes this equation in $\mathcal{C}_X$ hold:

$$\mathsf{fold}_F(h) \bullet \widehat{in_F} = h \bullet F^X \mathsf{fold}_F(h)$$

or equivalently, the following diagram in $\mathcal{C}$ commutes.

$$\begin{tikzcd}
F\mu F \times X \arrow[r, "in_F \times \mathsf{id}_X"] \arrow[d, "\langle F\,\mathsf{fold}_F(h) \circ \tau^F_{\mu F, X}, \mathsf{snd}\rangle"'] & \mu F \times X \arrow[d, "\mathsf{fold}_F(h)"] \\
FA \times X \arrow[r, "h"'] & A
\end{tikzcd}$$

Observe that fold and the standard catamorphism of $\mathcal{C}_X$ coincide by definition. But the keypoint with strong initality relies on the fact that the definition of fold results to be directly expressed in terms of the datatype constructors given by in_F. Moreover, fold gives rise to the desired 'catamorphism with parameters' when viewed as an arrow in $\mathcal{C}$. Note also the role played by the strength τ^F in the definition of fold: it distributes the global values stored in the parameters to the recursive calls.

For example, recall that the signature of natural numbers is captured by a functor N such that, functorially expressed, $N = \underline{1} + I$. Thus, $\tau^N_{A,X} = (\mathsf{fst} + \mathsf{id}) \circ d : (1 + A) \times X \to 1 + A \times X$ by Definition 3.3. So, for any object X and for any algebra $h = [h_Z, h_S] \circ d : (1 + A) \times X \to A$, with $h_Z : 1 \times X \to A$ and $h_S : A \times X \to A$, fold is the unique arrow $f = \mathsf{fold}_N(h)$ such that, $f \circ ([\mathit{Zero}, \mathit{Succ}] \times \mathsf{id}_X) = h \circ \langle (\mathsf{fst} + f) \circ d, \mathsf{snd}\rangle$. We can rewrite the right hand side of this equation and reverse d (since it is an isomorphism) arriving at these equations:

$$f \circ (\mathit{Zero} \times \mathsf{id}_X) = h_Z \qquad\qquad f \circ (\mathit{Succ} \times \mathsf{id}_X) = h_S \circ \langle f, \mathsf{snd}\rangle$$

In **Sets** they give rise to the equations (1) on page 4.

Using fold we can give a definition of a strength for any type functor induced by a bifunctor that is strong componentwise: A bifunctor $F : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ is called *bistrong* [CF92] if the functors $F(A, -)$ and $F(-, B)$ are strong.[4]

Definition 3.7 Let D be a strongly initial parameterized datatype induced by a bistrong functor F. Then, the strength of the corresponding type functor D is given by,

$$\tau^D_{A,X} \stackrel{\text{def}}{=} \mathsf{fold}_{F_A}(\phi_{A,X}) : DA \times X \to D(A \times X)$$

[3] We can simply write $FA \times X \to A$ instead of $F^X A \times X \to A$ as F^X and F coincide on objects by definition.

[4] Note that the strength of the functor $F(-, B)$ is such that, $\tau^{F(-,B)}_{A,X} : F(A, B) \times X \to F(A \times X, B)$.

where $\phi_{A,X} : F(A, D(A \times X)) \times X \to D(A \times X)$ is given by,

$$\phi_{A,X} = in_{F_{A \times X}} \circ \tau^{F(-,D(A\times X))}_{A,X}$$

The proof that $\tau^D_{A,X}$ is indeed a strength is shown in Proposition 4.12. For example, let us consider lists in **Sets**. For each algebra $h = [h_N, h_C] \circ d : (1 + A \times B) \times X \to B$, one can see that fold is the unique function $f : \mathsf{List}\, A \times X \to B$ such that:

$$f\ (Nil, x) = h_N(x) \qquad\qquad f\ (Cons\ (a, as), x) = h_C(a, f(as, x), x)$$

Consequently, the strength of the List type functor is given by:

$$\tau^{\mathsf{List}}(Nil, x) = Nil \qquad\qquad \tau^{\mathsf{List}}(Cons\ (a, as), x) = Cons\ ((a, x), \tau^{\mathsf{List}}(as, x))$$

since in this case,

$$\phi_{A,X} = [Nil \circ \mathsf{fst}, Cons \circ \langle \mathsf{fst} \times \mathsf{id}_X, \mathsf{snd} \circ \mathsf{fst}\rangle] \circ d$$

That is, τ^{List} maps the pair $([a_1, \ldots, a_n], x)$ to the list $[(a_1, x), \ldots, (a_n, x)]$.

Finally, the following proposition guarantees the existence of categories where strong initiality holds.

Proposition 3.8 ([CS91]) If $\mathcal{C}$ is cartesian closed, then every initial algebra is strongly initial.

4 Calculational Laws

In this section we turn to the study of calculational laws involving folds and catamorphisms. The algebraic laws we specify are grouped in two sets. That fold and the catamorphism of $\mathcal{C}_X$ coincide can be used to derive a set formed by usual standard laws for catamorphisms now stated for folds. The other set consists of laws that show the interaction between folds and between folds and catamorphisms.

4.1 Standard Laws

Recall that for catamorphism we presented an identity and a fusion law. We specify corresponding laws for fold which now follow (easily) by strong initiality. *Fold-identity* states that a fold on the lifting of an initial F-algebra is the identity X-action $\mathsf{fst} : \mu F \times X \to \mu F$.

Proposition 4.1 $\mathsf{fold}_F(\widehat{in_F}) = \mathsf{fst}$.

Fold-fusion states that the composition of a fold with a F^X-homomorphism is again a fold.

Proposition 4.2 For $h : FA \times X \to A$, $k : FB \times X \to B$ and $f : A \times X \to B$,

$$f \bullet h = k \bullet F^X f \ \Rightarrow\ f \bullet \mathsf{fold}_F(h) = \mathsf{fold}_F(k)$$

Concerning type functors we can specify a *map-fold-fusion* law that corresponds to map-cata-fusion which considers the composition of a type functor lifting with a fold.

Proposition 4.3 Let D be a type functor induced by a bistrong bifunctor F and $h : F_A C \times X \to C$ a F^X_A-algebra. Then, for each $f : A \to B$,

$$\mathsf{fold}_{F_B}(h) \bullet D^X f = \mathsf{fold}_{F_A}(h \bullet (F(f, \mathsf{id}_C) \circ \tau^{F(-,C)}_{A,X}))$$

This law is an immediate consequence of a more general one —presented in Proposition 4.10 below— which states that the composition of folds is again a fold, under certain conditions. (In fact, Proposition 4.10 applies in this case because $D^X f$ is a fold, like states Corollary 4.7 below.)

4.2 Further Laws

The first law confirms an obvious result: any catamorphism can be formulated as a fold that does not make use of the parameters.

Proposition 4.4 For every $h : FA \to A$, $\widehat{\llparenthesis h \rrparenthesis_F} = \mathsf{fold}_F(\widehat{h})$.

The rest of the laws are essentially fusion laws. We first analyze a *fold-cata-fusion* law. It states that the composition of a certain class of folds with a catamorphism is again a fold.

Proposition 4.5 Let F be a strong functor and X a fixed object. Let $\sigma_A : FA \times X \to GA$ be a natural transformation and $h : GA \to A$ an arbitrary G-algebra for a functor G. Then,

$$\llparenthesis h \rrparenthesis_G \circ \mathsf{fold}_F(in_G \circ \sigma_{\mu G}) = \mathsf{fold}_F(h \circ \sigma_A)$$

Proof As a consequence of the definitions of catamorphism and fold and naturality of σ we can draw the following commuting diagram, where we write f for $\mathsf{fold}_F(in_G \circ \sigma_{\mu G})$.

$$\begin{tikzcd}
F\mu F \times X \arrow[rr, "in_F \times \mathsf{id}_X"] \arrow[d, "\langle F f \circ \tau^F_{\mu F, X}, \mathsf{snd}\rangle"'] & & \mu F \times X \arrow[d, "f"] \\
F\mu G \times X \arrow[r, "\sigma_{\mu G}"] \arrow[d, "F\llparenthesis h \rrparenthesis_G \times \mathsf{id}_X"'] & G\mu G \arrow[r, "in_G"] \arrow[d, "G\llparenthesis h \rrparenthesis_G"'] & \mu G \arrow[d, "\llparenthesis h \rrparenthesis_G"] \\
FA \times X \arrow[r, "\sigma_A"'] & GA \arrow[r, "h"'] & A
\end{tikzcd}$$

Since $(F\llparenthesis h \rrparenthesis_G \times \mathsf{id}_X) \circ \langle F f \circ \tau^F_{\mu F,X}, \mathsf{snd}\rangle = \langle F(\llparenthesis h \rrparenthesis_G \circ f) \circ \tau^F_{\mu F,X}, \mathsf{snd}\rangle$ by functoriality of F and a law of product, it follows that the outermost commuting polygon states that $\llparenthesis h \rrparenthesis_G \circ f$ is a F^X-homomorphism from $\widehat{in_F}$ to $h \circ \sigma_A$. This homomorphism is equal to $\mathsf{fold}_F(h \circ \sigma_A)$ by strong initiality. $\square$

Since the action of a type functor on arrows is given by a catamorphism, we can state a law that says that the composition of a (certain class of) fold with the mapping of an arrow with a type functor is again a fold.

Corollary 4.6 Let D be a type functor induced by bifunctor H. Let $\sigma_C : FC \times X \to H_A C$ be a natural transformation only indexed on C. Then, for each $f : A \to B$,

$$Df \circ \mathsf{fold}_F(in_{H_A} \circ \sigma_{DA}) = \mathsf{fold}_F(in_{H_B} \circ H(f, \mathsf{id}_{DB}) \circ \sigma_{DB})$$

Proof Recall that $Df = \llparenthesis in_{H_B} \circ H(f, \mathsf{id}_{DB}) \rrparenthesis_{H_A}$, for $f : A \to B$. Taking $G = H_B$, and $h = in_{H_B} \circ H(f, \mathsf{id}_{DB})$ we are in the hypothesis of Proposition 4.5, following the equation. $\square$

Corollary 4.7 Let D be a type functor induced by a (bistrong) bifunctor F. Then, for each $f : A \times X \to B$, the lifting of type functor D, i.e $D^X f = Df \circ \tau^D_{A,X} : DA \times X \to DB$, is such that

$$D^X f = \mathsf{fold}_{F_A}(in_{F_B} \circ F(f, \mathsf{id}_{DB}) \circ \tau^{F(-,DB)}_{A,X})$$

Now we turn to the analysis of a *cata-fold-fusion* law which expresses that the lifting of a (certain class of) catamorphism followed by a fold is again a fold.

Proposition 4.8 Let $\sigma_A : FA \to GA$ be a strong natural transformation between two strong functors F and G. Let $h : F\mu G \to \mu G$ be the F-algebra given by $h = in_G \circ \sigma_{\mu G}$, and $k : GA \times X \to A$ an arbitary G^X-algebra for a given X. Then,

$$\mathsf{fold}_G(k) \bullet \widehat{\llparenthesis h \rrparenthesis_F} = \mathsf{fold}_F(k \bullet \widehat{\sigma_A})$$

Proof First, observe that $\mathsf{fold}_G(k) \bullet \widehat{(\!|h|\!)_F} = \mathsf{fold}_G(k) \circ ((\!|h|\!)_F \times \mathsf{id}_X)$ as a consequence of the definition of X-action composition. Now let us consider the following composed diagram:

$$\begin{tikzcd}
F\mu F \times X \arrow[rr, "in_F \times \mathsf{id}_X"] \arrow[d, "F(\!|h|\!)_F \times \mathsf{id}_X"'] & (I) & \mu F \times X \arrow[d, "(\!|h|\!)_F \times \mathsf{id}_X"] \\
F\mu G \times X \arrow[r, "\sigma_{\mu G} \times \mathsf{id}_X"] \arrow[d, "\langle F^X \mathsf{fold}_G(k), \mathsf{snd}\rangle"'] \arrow[dr, phantom, "(II)"] & G\mu G \times X \arrow[r, "in_G \times \mathsf{id}_X"] \arrow[d, "\langle G^X \mathsf{fold}_G(k), \mathsf{snd}\rangle"] \arrow[dr, phantom, "(III)"] & \mu G \times X \arrow[d, "\mathsf{fold}_G(k)"] \\
FA \times X \arrow[r, "\sigma_A \times \mathsf{id}_X"'] & GA \times X \arrow[r, "k"'] & A
\end{tikzcd}$$

The pentagon (I) commutes by definition of catamorphism while (III) by definition of fold. (II) commutes as a consequence of vertically composing these two diagrams:

$$\begin{tikzcd}
F\mu G \times X \arrow[r, "\sigma_{\mu G} \times \mathsf{id}_X"] \arrow[d, "\langle \tau^F_{\mu G,X}, \mathsf{snd}\rangle"'] & G\mu G \times X \arrow[d, "\langle \tau^G_{\mu G,X}, \mathsf{snd}\rangle"] \\
F(\mu G \times X) \times X \arrow[r, "\sigma_{\mu G \times X} \times \mathsf{id}_X"] \arrow[d, "F\,\mathsf{fold}_G(k) \times \mathsf{id}_X"'] & G(\mu G \times X) \times X \arrow[d, "G\,\mathsf{fold}_G(k) \times \mathsf{id}_X"] \\
FA \times X \arrow[r, "\sigma_A \times \mathsf{id}_X"'] & GA \times X
\end{tikzcd}$$

The top square follows by combining the strong naturality condition of σ (Definition 3.4) with naturality of snd, whereas the bottom square commutes by naturality of $\sigma \times \mathsf{id}$. By the application of the equality $\langle F^X \mathsf{fold}_G(k), \mathsf{snd}\rangle \circ (F(\!|h|\!)_F \times \mathsf{id}_X) = \langle F(\mathsf{fold}_G(k) \circ ((\!|h|\!)_F \times \mathsf{id}_X)) \circ \tau^F_{\mu F,X}, \mathsf{snd}\rangle$ to the outermost polygon in the first diagram above, we obtain that $\mathsf{fold}_G(k) \circ ((\!|h|\!)_F \times \mathsf{id}_X)$ is a homomorphism between $\widehat{in_F}$ and $k \bullet \widehat{\sigma_A}$. The desired equation thus follows by strong initiality. □

Corollary 4.9 Let D be a type functor induced by a bistrong bifunctor H. Let $k : H_B C \times X \to C$ be any H_B^X-algebra. Then, for each $f : A \to B$,

$$\mathsf{fold}_{H_B}(k) \bullet \widehat{D\,f} = \mathsf{fold}_{H_A}(k \circ (H(f, \mathsf{id}_C) \times \mathsf{id}_X))$$

Proof Recall that $Df = (\!|in_{H_B} \circ H(f, \mathsf{id}_{DB})|\!)_{H_A}$. If we consider $F = H_A$, $G = H_B$, and $\sigma_C = H(f, \mathsf{id}_C) : FC \to GC$ (natural on C), then we are in the hypothesis of Proposition 4.8, because: (i) F and G are strong by the fact that H is bistrong; and (ii) σ is a strong natural transformation since $H(f, \mathsf{id}_{DB\times X}) \circ \tau^{H_A}_{DB,X} = \tau^{H_B}_{DB,X} \circ (H(f, \mathsf{id}_{DB}) \times \mathsf{id}_X)$ holds by naturality of τ^{H_A} with respect to A. □

Now we present two laws that express the interaction between folds. The first one is concerned with the composition of two folds on the same parameters.

Proposition 4.10 Let F and G be two strong functors. Let $\sigma_A : FA \times X \to GA$ be a strong natural transformation between the functors $F \times \underline{X}$ and G. Let $h : F\mu G \times X \to \mu G$ be the F^X-algebra given by, $h = in_G \circ \sigma_{\mu G}$, and $k : GA \times X \to A$ an arbitrary G^X-algebra. Then,

$$\mathsf{fold}_G(k) \bullet \mathsf{fold}_F(h) = \mathsf{fold}_F(k \bullet \sigma_A)$$

On the other hand, for folds on different parameters composition takes place with the help of product associativity.

Proposition 4.11 Let F and G be two strong functors. Let $\sigma_A : FA \times X \to GA$ be a strong natural transformation between the functors $F \times \underline{X}$ and G. Let $h : F\mu G \times X \to \mu G$ be the F^X-algebra given by, $h = in_G \circ \sigma_{\mu G}$, and $k : GA \times Y \to A$ an arbitrary G^Y-algebra. Then,

$$\mathsf{fold}_G(k) \circ (\mathsf{fold}_F(h) \times \mathsf{id}_Y) \circ a_{\mu F,X,Y} = \mathsf{fold}_F(k \circ (\sigma_A \times \mathsf{id}_Y) \circ a_{FA,X,Y})$$

To conclude, we present the following proposition as an illustration of the use of some of the preceding calculational laws.

Proposition 4.12 Let D be a type functor induced by a bistrong functor F. Then, the morphism $\tau^D_{A,X}$ given by Definition 3.7 is indeed a strength.
Proof We show that $\tau^D_{A,X}$ verifies both diagrams in the definition of strength (see Definition 3.2).
(i)

$$\begin{aligned}
& D\,\mathsf{fst} \circ \tau^D_{A,X} \\
= \quad & \{\ \text{Corollary 4.6}\ \} \\
& \mathsf{fold}_{F_A}(in_{F_A} \circ F(\mathsf{fst}, \mathsf{id}_{DA}) \circ \tau^{F(-,DA)}_{A,X}) \\
= \quad & \{\ \text{definition of strength and lifting}\ \} \\
& \mathsf{fold}_{F_A}(\widehat{in_{F_A}}) \\
= \quad & \{\ \text{Proposition 4.1}\ \} \\
& \mathsf{fst}
\end{aligned}$$

(ii) We omit some subscripts in order to improve readability.

$$\begin{aligned}
& \tau^D_{A\times X,Y} \circ (\tau^D_{A,X} \times \mathsf{id}_Y) \circ a \\
= \quad & \{\ \text{Proposition 4.11}\ \} \\
& \mathsf{fold}_{F_A}(in \circ \tau^{F(-,D((A\times X)\times Y))}_{A\times X,Y} \circ (\tau^{F(-,D((A\times X)\times Y))}_{A,X} \times \mathsf{id}_Y) \circ a) \\
= \quad & \{\ \text{definition of strength}\ \} \\
& \mathsf{fold}_{F_A}(in_{F_{(A\times X)\times Y}} \circ F(a, \mathsf{id}) \circ \tau^{F(-,D((A\times X)\times Y))}_{A,X\times Y}) \\
= \quad & \{\ \text{Proposition 4.5}\ \} \\
& (\!|in \circ F(a, \mathsf{id})|\!) \circ \mathsf{fold}_{F_A}(in \circ \tau^{F(-,D(A\times(X\times Y)))}_{A,X\times Y}) \\
= \quad & \{\ \text{definition of type functor and } \tau^D\ \} \\
& D\,a \circ \tau^D_{A,X\times Y}
\end{aligned}$$

5 Conclusions and Further Work

This paper presented the first results in an attempt to consolidate a calculational framework for functionals on strong datatypes. We introduced fold as the unique arrow obtained by initiality in a category of algebras in $\mathcal{C}_X$. This construction showed to be especially suitable for the whole development. It permitted to regard folds as catamorphisms in $\mathcal{C}_X$ causing the formulation of various laws to be quite immediate. Also, it made some proofs easier, since most of the composite diagrams formed on the basis of definitions of functionals were more easily constructed in $\mathcal{C}_X$ and then translated to $\mathcal{C}$.

There are some possible directions for future work. First of all, as mentioned in the introduction, we can easily derive similar properties for *unfold* [CS91, CF92] by following a dual reasoning. Unfold corresponds to an extension (with parameters) of a standard functional on co-inductive types called *anamorphism* [MFP91, Fok92]. Furthermore, we might also consider an extension with parameters for other functionals like primitive recursion [Geu92, Mee92] and its dual, co-recursion [Geu92]. Such an extension is not hard to be formulated. However, the interesting part seems to be the study of their calculational properties (We guess that their definition as unique arrows in $\mathcal{C}_X$ will be helpful in this case as well).

Finally, another direction of interest concerns the study of *accumulations* [Gib93, HIT96], a class of recursive functionals on inductive types that use extra parameters to accumulate intermediate results. For example, in [HIT96] they are presented as higher-order catamorphisms. Our approach is to see accumulations as functionals that take extra parameters whose value may change during computation. Assuming that the signature of datatypes is given by strong functors, preliminary investigations have shown that a definition of accumulations may be achieved proceeding in a similar way as we did with fold. Concretely, we have found two possible formulations of accumulations. One formulation requires only a little modification to the definition of functor lifting. For the natural numbers, it corresponds to a function $f : \mathsf{Nat} \times X \to A$ such that, $f(\mathit{Zero}, x) = h_Z(x)$ and $f(\mathit{Succ} n, x) = h_C(f(n, g(x)), x)$, for $g : X \to X$, $h_Z : X \to A$ and $h_S : A \times X \to A$. The other formulation is a bit more complex requiring the modification of the definition of homomorphism between algebras of a lifted functor F^X.

Acknowledgements. I would like to thank the Organization of the 8th Nordic Workshop on Programming Theory for the financial support that made my participation in this event possible. Diagrams were constructed using Paul Taylor's diagrams package.

References

[BS93] S. Brookes and K. Van Stone. Monads and Comonads in Intensional Semantics. Technical Report CMU-CS-93-140, School of Computer Science, Carnegie Mellon University, 1993.

[BW88] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice-Hall, 1988.

[BW90] M. Barr and C. Wells. *Category Theory for Computing Science.* Prentice-Hall, 1990.

[CF92] R. Cockett and T. Fukushima. About Charity. Technical Report 92/480/18, University of Calgary, June 1992.

[CS91] R. Cockett and D. Spencer. Strong Categorical Datatypes I. In *Category Theory 1991,* volume 13 of *Canadian Math. Society Conference Proceedings*, pages 141–169, 1991.

[Fok92] M.M. Fokkinga. *Law and Order in Algorithmics.* PhD thesis, Universiteit Twente, 1992.

[Geu92] H. Geuvers. Inductive and Coinductive Types with Iteration and Recursion. In *Workshop on Types for Proofs and Programs*, pages 193–217, Båstad, Sweden, June 1992.

[Gib93] J. Gibbons. Upwards and Downwards Accumulations on Trees. In *Mathematics of Program Construction,* LNCS 669. Springer-Verlag, 1993.

[HIT96] Z. Hu, H. Iwasaki, and M. Takeichi. Calculating Accumulations. Technical Report METR 96-03, Faculty of Engineering, University of Tokyo, March 1996.

[How92] B.T. Howard. *Fixed Points and Extensionality in Typed Functional Programming Languages.* PhD thesis, Stanford University, 1992.

[Jac95] B. Jacobs. Parameters and Parameterization in Specification, using distributive categories. *Fundamenta Informaticae*, 24(3):209–250, 1995.

[Jeu93] J. Jeuring. *Theories for Algorithm Calculation.* PhD thesis, Utrecht University, 1993.

[Mal90] G. Malcolm. Data Structures and Program Transformation. *Science of Computer Programming*, 14:255–279, 1990.

[Mee92] L. Meertens. Paramorphisms. *Formal Aspects of Computing*, 4:413–424, 1992.

[MFP91] E. Meijer, M. Fokkinga, and R. Paterson. Functional Programming with Bananas, Lenses, Envelopes and Barbed Wire. In *5th FPCA,* LNCS 523. Springer-Verlag, 1991.

[MJ95] E. Meijer and J. Jeuring. Merging Monads and Folds for Functional Programming. In *Advanced Functional Programming,* LNCS 925. Springer-Verlag, 1995.

[Par97] A. Pardo. forthcoming PhD Thesis. Technische Hochschule Darmstadt, 1997.

[Wik87] Å. Wikström. *Functional Programming using Standard ML.* Prentice-Hall, 1987.