Proof Principles for Datatypes with Iterated Recursion*

Ulrich Hensel
Inst. Theor. Inf., TU Dresden
D-01062 Dresden,
Germany.
hensel@tcs.inf.tu-dresden.de

Bart Jacobs
Dep. Comp. Sci., Univ. Nijmegen,
P.O. Box 9010, 6500 GL Nijmegen,
The Netherlands.
bart@cs.kun.nl

Abstract. Data types like trees which are finitely branching and of (possibly) infinite depth are described by iterating initial algebras and terminal coalgebras. We study proof principles for such data types in the context of categorical logic, following and extending the approach of [14, 15]. The technical contribution of this paper involves a description of initial algebras and terminal coalgebras in total categories of fibrations for lifted "datafunctors". These lifted functors are used to formulate our proof principles. We test these principles by proving some elementary results for four kinds of trees (with finite or infinite breadth or depth) using the proof tool PVS.

1 Introduction

Algebras and coalgebras are of well-established importance in computer science, notably in the theory of datatypes, where especially *initial* algebras and *terminal* coalgebras play a distinguished rôle. Over the past decade there is more and more interest in the logic associated with initial algebras and terminal coalgebras. It is known for a long time [10] that induction is the appropriate proof method to reason about functions which are defined on initial algebras. More recently, bisimulation is recognised as the appropriate proof method to reason about (definable) functions taking values in terminal coalgebras [1, 29, 28]. In this context it became clear that there is an equivalent "binary" induction principle, stating that all congruences[2] on an initial algebra contain the equality relation (*i.e.* are reflexive). This induction principle for relations is completely dual to the bisimulation proof principle, which says that every bisimulation on a terminal coalgebra is contained in the equality relation. Unary predicates on coalgebras are also important, namely as "invariants" [17], or as "subcoalgebras" [30].

---

* This paper was written during a visit of Ulrich Hensel to the Computing Science Institute of the University of Nijmegen. It is to appear in the LNCS Proceedings of *Category Theory and Computer Science 1997*.

[2] We mean a relation which is closed under the constructors of the algebra, but which is not necessarily an equivalence relation. For example, a congruence $R \subseteq \mathbb{N} \times \mathbb{N}$ on the natural numbers satisfies $R(0,0)$ and $R(x,y) \Rightarrow R(S(x),S(y))$ for all $x, y \in \mathbb{N}$.

This (recent) interest in the logic of initial algebras and terminal coalgebras (see for example [6, 2, 22, 26, 7, 28, 9, 8, 12, 27]) is also motivated by the increasing use of powerful and expressive proof tools, like PVS, ISABELLE, HOL, LARCH, LEGO, COQ *etc.*, in which users may introduce their own inductively (and sometimes also coinductively) defined type. And, this is of most relevance, the tool[3] then automatically generates several standard definitions and results for such types, like "map" definitions (or functoriality) and (co)inductive proof principles. In practice the available (theoretical) possibilities are not yet fully exploited by such tools.

The concepts of initial algebra and terminal coalgebra form the basis for the experimental programming language CHARITY [5, 4]. It exploits initiality and terminality as the explicit definition schemas and demonstrates the remarkable power of these notions. CHARITY only involves a term language for writing programs, but no logic to reason about these programs.

It is the aim of this paper to contribute to the logical theory behind inductively and coinductively defined types, by developing proof principles for types which are obtained by iterated mixing of initial and terminal fixed points. Ideally, this should form a basis for a suitable blend of programming languages like CHARITY and proof tools like PVS, combining specification, programming and verification.

Interestingly, we expect that the theory that we are about to develop will (eventually) have practical applications. But the actual development of this theory takes place at a very abstract and remote level, using (technical) notions and techniques from categorical logic. Once we have formalised the relevant notions at a suitably abstract level, the proof principles that we seek take standard formulations (see Definition 6). In the present paper we only develop the technical details, and illustrate their usefulness in several typical examples. Space restrictions do not allow us to introduce all relevant notions from categorical logic, and we refer the interested reader to [3, II, 8.5] or [15, 20, 21] (or to the forthcoming book [18]) for background information.

This paper is organised as follows. To start, we identify the kind of "datafunctors" that we shall study, and present several examples of trees that will be used later. Then, in Section 3 we show how to obtain a fixed point for a functor on a total category of a fibration by using fixed points in the base category and in the fibre categories. This extends standard approaches to obtain such global structure [13]. Subsequently in Section 4 we formulate our proof principles as preservation properties, and establish their validity in the presence of comprehension and quotients (following [14, 15]). We pay special attention to handling mixed proof obligations in Section 5, and finally illustrate the use of these principles for four kinds of trees (with finite or infinite breadth or depth), each involving a composition operation. We prove in each case, using PVS, that composition with the empty tree yields the identity. To do so, we have to formalise our proof principles in (a suitable theory in) PVS.

[3] There is considerable difference between the various tools in the extend to which they do this.

Throughout, the data types and their proof principles involve various kinds of fixed points. In this (version of the) paper we are not concerned with the actual existence of these fixed points (which is a non-trivial topic in itself), but only with the derived formal properties.

2 Datafunctors

In this section we introduce the kind of functors that we will consider. These functors are built up from multiple arguments with constants, finite products and coproducts and, most importantly, with fixed points arising both from initial algebras and from terminal coalgebras. This allows us to iterate data type constructions, as in the examples below.

Definition 1. For a sufficiently complete and cocomplete category $\mathbb{C}$, the class of **datafunctors on** $\mathbb{C}$ is the least class of functors $\mathbb{C}^n \to \mathbb{C}$ (for $n \in \mathbb{N}$) containing:

1. the projection functors $\Pi_i\colon \mathbb{C}^n \to \mathbb{C}$ for $1 \leq i \leq n$, the constant functors $A\colon \mathbf{1} = \mathbb{C}^0 \to \mathbb{C}$ for $A \in \mathbb{C}$, and the (binary) product and coproduct functors $\times\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$, $+\colon \mathbb{C} \times \mathbb{C} \to \mathbb{C}$;
2. the composite $S \circ (T_1, \ldots, T_m)\colon \mathbb{C}^n \to \mathbb{C}$, for datafunctors $T_i\colon \mathbb{C}^n \to \mathbb{C}$ and $S\colon \mathbb{C}^m \to \mathbb{C}$;
3. the initial algebra carrier $\mu X. T(-, X)\colon \mathbb{C}^n \to \mathbb{C}$, and also the terminal coalgebra carrier $\nu X. T(-, X) : \mathbb{C}^n \to \mathbb{C}$, for a datafunctor $T\colon \mathbb{C}^{n+1} \to \mathbb{C}$.

The completeness and cocompleteness which is assumed in this definition refers to the existence of the initial algebras and terminal coalgebras mentioned in point 3. The class of datafunctors we use is akin to the one in [19], but does not involve "shape". Also, at this stage, we do not consider the concept of "strength" (see [5, 23, 16], although it forms an essential part of the complete story of these functors.

Explicitly, the functor $\mu X. T(-, X)\colon \mathbb{C}^n \to \mathbb{C}$ in the third clause maps a sequence of objects $\vec{Y} \in \mathbb{C}^n$ to the carrier $\mu X. T(\vec{Y}, X) \in \mathbb{C}$ of the initial algebra $\alpha_{\vec{Y}}\colon T(\vec{Y}, \mu X. T(\vec{Y}, X)) \xrightarrow{\cong} \mu X. T(\vec{Y}, X)$ of the functor $T(\vec{Y}, -)\colon \mathbb{C} \to \mathbb{C}$. And it maps a sequence of morphisms $\vec{f}\colon \vec{Y} \to \vec{Z}$ in $\mathbb{C}^n$ to the unique morphism of algebras $\mu X. T(\vec{f}, X)\colon \mu X. T(\vec{Y}, X) \to \mu X. T(\vec{Z}, X)$ in the following diagram.

$$
\begin{tikzcd}
T(\vec{Y}, \mu X. T(\vec{Y}, X)) \arrow[r, dashed, "{T(id, \mu X. T(\vec{f}, X))}"] \arrow[dd, "\alpha_{\vec{Y}}"', "\cong"] & T(\vec{Y}, \mu X. T(\vec{Z}, X)) \arrow[d, "{T(\vec{f}, id)}"] \\
& T(\vec{Z}, \mu X. T(\vec{Z}, X)) \arrow[d, "\cong"', "\alpha_{\vec{Z}}"] \\
\mu X. T(\vec{Y}, X) \arrow[r, dashed, "{\mu X. T(\vec{f}, X)}"'] & \mu X. T(\vec{Z}, X)
\end{tikzcd}
$$

The terminal coalgebra functor $\nu X.\, T(-, X): \mathbb{C}^n \to \mathbb{C}$ is defined in a dual manner. Often we simply write μT and νT for $\mu X.\, T(-, X)$ and $\nu X.\, T(-, X)$. For an algebra $f: T(\vec{Y}, Z) \to Z$ we sometimes write $\mathsf{reduce}(f): \mu X.\, T(\vec{Y}, X) \to Z$ for the unique homomorphism of algebras resulting from initiality. Similarly, we write $\mathsf{coreduce}(g): Z \to \nu X.\, T(\vec{Y}, X)$ for the unique homomorphism of coalgebras induced by a coalgebra $g: Z \to T(\vec{Y}, Z)$. Hence the map $\mu X.\, T(\vec{f}, X)$ in the above diagram is $\mathsf{reduce}(\alpha_{\vec{Z}} \circ T(\vec{f}, id))$.

Example 1. (i) The polynomial functor

$$F_A : \mathbb{C} \times \mathbb{C} \longrightarrow \mathbb{C} \qquad \text{given by} \qquad (Y, X) \mapsto (A \times Y \times X) + 1$$

is a data functor built up with rules 1. and 2. in Definition 1. It will form the basis for the next four examples.

(ii) The functor $\mathsf{List}(A \times -) = \mu X.\, F_A(-, X): \mathbb{C} \to \mathbb{C}$ sends a (parameter) object Y to the carrier of the initial $F_A(Y, -)$-algebra $\mathsf{List}(A \times Y)$, with constructors $\mathsf{nil}: 1 \to \mathsf{List}(A \times Y)$ and $\mathsf{cons}: (A \times Y) \times \mathsf{List}(A \times Y) \to \mathsf{List}(A \times Y)$. The elements of $\mathsf{List}(A \times Y)$ are generated by these constructors and can thus be seen as finite sequences (of pairs).

(iii) The functor $\mathsf{Colist}(A \times -) = \nu X.\, F_A(-, X): \mathbb{C} \to \mathbb{C}$ sends a (parameter) object Y to the carrier of the terminal $F_A(Y, -)$-coalgebra $\mathsf{Colist}(A \times Y)$, with destructor $\mathsf{next}: \mathsf{Colist}(A \times Y) \to ((A \times Y) \times \mathsf{Colist}(A \times Y)) + 1$. The elements of $\mathsf{Colist}(A \times Y)$ are seen as finite or infinite sequences of elements of $A \times Y$.

(iv) A signature functor for finitely branching trees is

$$\mathsf{Tree}(A) : \mathbb{C} \longrightarrow \mathbb{C} \qquad \text{sending} \qquad X \mapsto \mathsf{List}(A \times X).$$

A $\mathsf{Tree}(A)$-algebra is a morphism $\mathsf{List}(A \times X) \to X$. It constructs a new node in X out of a list of subtrees in $\mathsf{List}(A \times X)$ which are all labeled with a value from A. The initial $\mathsf{Tree}(A)$-algebra contains all A-labeled finitely branching trees with finite depth. Note that the empty list of subtrees corresponds to a leaf. In the sequel we write the initial algebra as $\mathsf{br}: \mathsf{List}(A \times \mathsf{FTreeF}(A)) \xrightarrow{\cong} \mathsf{FTreeF}(A)$.

A $\mathsf{Tree}(A)$-coalgebra, is given by a morphism $X \to \mathsf{List}(A \times X)$ which decomposes a value in X into a list of successor states paired with a "visible" output in A. The carrier of the terminal $\mathsf{Tree}(A)$-coalgebra contains all A-labeled, finitely branching trees with possibly infinite depth. We use the same name br as before for the structure map of the terminal coalgebra $\mathsf{br}: \mathsf{FTreeI}(A) \xrightarrow{\cong} \mathsf{List}(A \times \mathsf{FTreeI}(A))$.

(v) The signature functor for possibly infinitely branching trees is

$$\mathsf{Cotree}(A) : \mathbb{C} \longrightarrow \mathbb{C} \qquad \text{given by} \qquad X \mapsto \mathsf{Colist}(A \times X).$$

The initial $\mathsf{Cotree}(A)$-algebra $\mathsf{ITreeF}(A)$ contains all possibly infinitely branching trees with finite depth. And the terminal $\mathsf{Cotree}(A)$-coalgebra $\mathsf{ITreeI}(A)$ additionally contains those with infinite depth. We use br to denote both the structure maps of these fixed points.

Notice that the examples (iv) and (v) describe the four combinatorial possibilities of finite versus possibly infinite in breadth or depth for trees, written as FTreeF, FTreeI, ITreeF and ITreeI. These four examples will reappear in Section 6.

3 Initial algebras and terminal coalgebras in total categories

From the perspective of categorical logic, the datafunctors from the previous section are functors $\mathbb{B}^n \to \mathbb{B}$ on a base category $\mathbb{B}$ of types and terms. In order to reason about such functors, one needs a fibration on top of this base category, where the objects in the fibres are seen as predicates on the underlying objects (see [18] for more information about this perspective). Appropriate reasoning principles are obtained by lifting the data functors on the base category to "logical" functors on the total category, by induction on the structure of the data functors. Essentially the same structure (of products and coproducts) is used in the total categories (derived from logical structure in fibre categories), see [14, 15] for more details. In this section we develop the theory which is required to extend this approach to data functors involving iterated fixed points. It will be the categorical contribution of this paper. The extension will involve an explanation of how to obtain fixed points for functors on total categories from fixed points in base categories, plus fixed points in the fibres. In later sections this theory will be applied to the particular situations that we are interested in.

Our starting point is the following situation. We have fibrations $\begin{smallmatrix}\mathbb{E}\\ \downarrow p\\ \mathbb{B}\end{smallmatrix}$ and $\begin{smallmatrix}\mathbb{D}\\ \downarrow q\\ \mathbb{A}\end{smallmatrix}$ together with a functor $T: \mathbb{A} \times \mathbb{B} \to \mathbb{B}$ between base categories, and over T a fibred functor $H: \mathbb{D} \times \mathbb{E} \to \mathbb{E}$ between total categories, in a (commuting) square:

$$\begin{tikzcd}
\mathbb{D} \times \mathbb{E} \arrow[r, "H"] \arrow[d, "q \times p"'] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{A} \times \mathbb{B} \arrow[r, "T"'] & \mathbb{B}
\end{tikzcd} \tag{1}$$

In this situation we think of p as the fibration which captures the logic that we are working in, and of q as a "parameter fibration"; often q will be p^n, for some $n \in \mathbb{N}$. The functor T between the base categories is a datafunctor, and H between total categories is a certain lifting of T. We shall assume that for each object $A \in \mathbb{A}$, the functor $T(A, -): \mathbb{B} \to \mathbb{B}$ has an initial algebra and a terminal coalgebra, written as

$$T(A, \mu T(A)) \xrightarrow[\cong]{\alpha_A} \mu T(A) \qquad \text{and} \qquad \nu T(A) \xrightarrow[\cong]{\beta_A} T(A, \nu T(A))$$

We thus get two functors $\mu T, \nu T: \mathbb{A} \to \mathbb{B}$, as described after Definition 1. Our aim in this section is to construct similar functors $\mu H, \nu H: \mathbb{D} \to \mathbb{E}$ over $\mu T, \nu T$

(respectively). They will be constructed via two auxiliary functors between fibres: for $X \in \mathbb{D}_A$, we define

$$\begin{array}{ccccccc} \mathbb{E}_{\mu T(A)} & \xrightarrow{H_X} & \mathbb{E}_{\mu T(A)} & \qquad & \mathbb{E}_{\nu T(A)} & \xrightarrow{H^X} & \mathbb{E}_{\nu T(A)} \\ Y & \longmapsto & \left(\alpha_A^{-1}\right)^* (H(X,Y)) & & Y & \longmapsto & (\beta_A)^* (H(X,Y)) \end{array} \tag{2}$$

The following result from [15] will play an important rôle.

Lemma 2. *A natural transformation $\sigma: SU \Rightarrow UT$ in a situation*

$$\begin{tikzcd} \mathbb{A} \arrow[r, "S"] & \mathbb{A} \\ \mathbb{B} \arrow[u, "U"] \arrow[r, "T"'] \arrow[ur, phantom, "\stackrel{\sigma}{\Longrightarrow}"] & \mathbb{B} \arrow[u, "U"'] \end{tikzcd} \qquad \textit{induces a functor} \qquad \begin{tikzcd} \mathrm{Alg}(S) \\ \mathrm{Alg}(T) \arrow[u, "\mathrm{Alg}(U)"'] \end{tikzcd}$$

by $(TX \xrightarrow{f} X) \mapsto (SUX \xrightarrow{\sigma_X} UTX \xrightarrow{Uf} UX)$.

And if σ is an isomorphism, then a right adjoint G to U

$$\begin{tikzcd} \mathbb{A} \arrow[d, bend left, "G"] \\ \mathbb{B} \arrow[u, bend left, "U"] \arrow[u, phantom, "\dashv"] \end{tikzcd} \qquad \textit{induces a right adjoint} \qquad \begin{tikzcd} \mathrm{Alg}(S) \arrow[d, bend left, "\mathrm{Alg}(G)"] \\ \mathrm{Alg}(T) \arrow[u, bend left, "\mathrm{Alg}(U)"] \arrow[u, phantom, "\dashv"] \end{tikzcd}$$

where the functor $\mathrm{Alg}(G)$ *arises from* $\tau: TG \Rightarrow GS$, *the adjoint transpose of* $UTG \cong SUG \stackrel{S\varepsilon}{\Longrightarrow} S$.

Dually, a natural transformation $\tau: UT \Rightarrow SU$ *gives a functor* $\mathrm{CoAlg}(U)$: $\mathrm{CoAlg}(T) \to \mathrm{CoAlg}(S)$ *between categories of coalgebras. Furthermore, if this τ is an isomorphism, then, a left adjoint $F \dashv U$ induces a left adjoint* $\mathrm{CoAlg}(F) \dashv \mathrm{CoAlg}(U)$. □

We now turn to initial algebras and terminal coalgebras of functors $H(X, -)$ as above.

Theorem 3. *Consider the situation described in (1) above, with auxiliary functors H_X and H^X as in (2). Assuming that the initial algebras and terminal coalgebras in fibre categories in (i) and (ii) below exist, we can define for each $X \in \mathbb{D}$ over $A \in \mathbb{A}$ an initial algebra $\mu H(X)$ over $\mu T(A)$ and a terminal coalgebra $\nu H(X)$ over $\nu T(A)$ for the functor $H(X, -): \mathbb{E} \to \mathbb{E}$ as follows.*

(i) Let $\mu H(X)$ be the carrier of the initial algebra

$$\begin{array}{ccc} H_X(\mu H(X)) & \xrightarrow[\cong]{a_X} & \mu H(X) \\ \| & & \\ (\alpha_A^{-1})^* H(X, \mu H(X)) & & \end{array}$$

of the functor H_X in the fibre category $\mathbb{E}_{\mu T(A)}$. It gives rise to an $H(X,-)$-algebra

$$H(X, \mu H(X)) \dashrightarrow (\alpha_A^{-1})^* H(X, \mu H(X)) \xrightarrow{a_X} \mu H(X)$$

over the initial $T(A,-)$-algebra $\alpha_A: T(A, \mu T(A)) \xrightarrow{\cong} \mu T(A)$, where the dashed arrow is the unique one over α_A for which the composite $H(X, \mu H(X)) \to (\alpha_A^{-1})^* H(X, \mu H(X)) \to H(X, \mu H(X))$ is the identity. This $H(X,-)$-algebra is the initial one.

(ii) Assume now that p is a bifibration, i.e. *that substitution functors u^* have left adjoints $\coprod_u$, and let $\nu H(X)$ be the carrier of the terminal H^X-coalgebra in the fibre $\mathbb{E}_{\nu T(A)}$:*

$$\begin{tikzcd} \nu H(X) \arrow[r, "b_X", "\cong"'] & H^X(\nu H(X)) \arrow[d, equal] \\ & \beta_A^* H(X, \nu H(X)) \end{tikzcd}$$

It yields as terminal $H(X,-)$-coalgebra the map:

$$\nu H(X) \xrightarrow{b_X} \beta_A^* H(X, \nu H(X)) \longrightarrow H(X, \nu H(X))$$

over the terminal $T(A,-)$-coalgebra $\beta_A: \nu T(A) \xrightarrow{\cong} T(A, \nu T(A))$.

In case the fibration p in this result is a fibred preorder modelling some logic involving provability, rather than some type theory with proof-objects, the initial algebras and terminal coalgebras in the fibres are simply least and greatest fixed points of monotone functions. We shall see such examples later. But we present this result in its general—non-preordered—form.

Proof. (i) Assume an algebra $f: H(X,Y) \to Y$ in $\mathbb{E}$, say over $u: T(A,B) \to B$ in $\mathbb{B}$—with $f': H(X,Y) \to u^*(Y)$ as the vertical part of f. Initiality of the algebra $\alpha_A: T(A, \mu T(A)) \xrightarrow{\cong} \mu T(A)$ gives a unique map of algebras $v = \mathsf{reduce}(u): \mu T(A) \to B$ with $v \circ \alpha_A = u \circ T(id_A, v)$. Using that H is a fibred functor, we can construct an H_X-algebra $g: H_X(v^*(Y)) \to Y$ on $v^*(Y) \in \mathbb{E}_{\mu T(A)}$ as composite:

$$\begin{tikzcd} (\alpha_A^{-1})^* H(X, v^*(Y)) \arrow[r, "\cong"] & (\alpha_A^{-1})^* T(id_A, v)^* H(X,Y) \arrow[d, "(\alpha_A^{-1})^* T(id_A, v)^*(f')"] \\ & (\alpha_A^{-1})^* T(id_A, v)^* u^*(Y) \arrow[r, "\cong"'] & v^*(Y) \end{tikzcd}$$

By initiality of $\mu H(X)$ we get a unique (vertical) map $h: \mu H(X) \to v^*(Y)$ with $h \circ a_X = H_X(h) \circ g$. The required mediating $H(X,-)$-algebra map is then the composite $\mu H(X) \to v^*(Y) \to Y$ (over v) of h and the cartesian map $v^*(Y) \to Y$.

(ii) For an arbitrary coalgebra $f: Y \to H(X,Y)$ over $u: B \to T(A,B)$—with vertical part $f': Y \to u^* H(X,Y)$—we get by terminality a unique map $v = \mathsf{coreduce}(u): B \to \nu T(A)$ in $\mathbb{B}$ with $\beta_A \circ v = T(id_A, v) \circ u$. We can define a functor $K: \mathbb{E}_B \to \mathbb{E}_B$ by $Z \mapsto u^* H(X,Z)$. It yields a diagram commuting up-to-isomorphism:

$$\begin{tikzcd}
\mathbb{E}_B \arrow[r, "K"] & \mathbb{E}_B \\
\mathbb{E}_{\nu T(A)} \arrow[u, "v^*"] \arrow[r, "H^X"'] \arrow[ur, phantom, "\cong"] & \mathbb{E}_{\nu T(A)} \arrow[u, "v^*"']
\end{tikzcd}$$

since

$$\begin{aligned} K v^*(Y) &= u^* H(X, v^*(Y)) \cong u^* T(id_A, v)^* H(X,Y) \\ &\cong v^* \beta_A^* H(X,Y) = v^* H^X(Y). \end{aligned}$$

The left adjoint $\coprod_v \dashv v^*$ now gives by Lemma 2 (ii) a left adjoint to the induced functor $\mathrm{CoAlg}(v^*): \mathrm{CoAlg}(H^X) \to \mathrm{CoAlg}(K)$. As a consequence, this functor $\mathrm{CoAlg}(v^*)$ preserves terminal coalgebras. Hence the terminal K-algebra is the composite

$$v^*(\nu H(X)) \xrightarrow[\cong]{v^*(b_X)} v^* H^X(\nu H(X)) \xrightarrow{\cong} K(v^*(\nu H(X)))$$

The vertical part $f': Y \to u^* H(X,Y) = K(Y)$ of our original map f is a K-coalgebra, and thus gives rise to a unique (vertical) map of K-coalgebras $g: Y \to v^*(\nu H(X))$. Composing this g with the cartesian map $v^*(\nu H(X)) \to \nu H(X)$ yields the required map of $H(X,-)$-coalgebras $Y \to \nu H(X)$ (over v). □

Proposition 4. *In the situation of the previous theorem, assume that the substitution functors u^* of the fibration p have both a left adjoint $\coprod_u$ and a right adjoint $\prod_u$. The resulting functors $\mu H, \nu H: \mathbb{D} \to \mathbb{E}$ are then fibred functors in diagrams:*

$$\begin{tikzcd}
\mathbb{D} \arrow[r, "\mu H"] \arrow[d, "q"'] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{A} \arrow[r, "\mu T"'] & \mathbb{B}
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathbb{D} \arrow[r, "\nu H"] \arrow[d, "q"'] & \mathbb{E} \arrow[d, "p"] \\
\mathbb{A} \arrow[r, "\nu T"'] & \mathbb{B}
\end{tikzcd}$$

Proof. Assume a morphism $w: B \to A$ in $\mathbb{A}$ and an object $X \in \mathbb{D}_A$. We get two morphisms $\mu T(w): \mu T(B) \to \mu T(A)$ and $\nu T(w): \nu T(B) \to \nu T(A)$ in $\mathbb{B}$, determined by the equations

$$\mu T(w) \circ \alpha_B = \alpha_A \circ T(w, \mu T(w)) \quad \text{and} \quad \beta_A \circ \nu T(w) = T(w, \nu T(w)) \circ \beta_B.$$

We also get two isomorphisms

$$\begin{tikzcd} \mathbb{E}_{\mu T(B)} \arrow[r, "H_{w^*(X)}"] & \mathbb{E}_{\mu T(B)} \\ \mathbb{E}_{\mu T(A)} \arrow[u, "\mu T(w)^*"] \arrow[r, "H_X"'] \arrow[ur, phantom, "\cong"] & \mathbb{E}_{\mu T(A)} \arrow[u, "\mu T(w)^*"'] \end{tikzcd} \qquad \begin{tikzcd} \mathbb{E}_{\nu T(B)} \arrow[r, "H^{w^*(X)}"] & \mathbb{E}_{\nu T(B)} \\ \mathbb{E}_{\nu T(A)} \arrow[u, "\nu T(w)^*"] \arrow[r, "H^X"'] \arrow[ur, phantom, "\cong"] & \mathbb{E}_{\nu T(A)} \arrow[u, "\nu T(w)^*"'] \end{tikzcd}$$

Since

$$\begin{aligned} H_{w^*(X)}(\mu T(w))^*(Y) &= (\alpha_B^{-1})^* H(w^*(X), (\mu T(w))^*(Y)) \\ &\cong (\alpha_B^{-1})^* T(w, \mu T(w))^*(H(X, Y) \\ &\cong (\mu T(w))^*(\alpha_A^{-1})^* H(X, Y) \\ &= (\mu T(w))^* H_X(Y). \\ H^{w^*(X)}(\nu T(w))^*(Y) &= \beta_B^* H(w^*(X), (\nu T(w))^*(Y)) \\ &\cong \beta_B^* T(w, \nu T(w))^* H(X, Y) \\ &\cong (\nu T(w))^* \beta_A^* H(X, Y) \\ &= (\nu T(w))^* H^X(Y). \end{aligned}$$

By Lemma 2, the two adjoints $(\mu T(w))^* \dashv \prod_{\mu T(w)}$ and $\coprod_{\nu T(w))} \dashv (\nu T(w))^*$ give rise to right and left adjoints:

$$\begin{tikzcd}[column sep=small] & \mathrm{Alg}(H_{w^*(X)}) \arrow[d, bend left, ""{name=R}] \\ \mathrm{Alg}((\mu T(w))^*) & \mathrm{Alg}(H_X) \arrow[u, bend left, ""{name=L}] \arrow[from=L, to=R, phantom, "\dashv"] \end{tikzcd} \qquad \begin{tikzcd}[column sep=small] \mathrm{CoAlg}(H^{w^*(X)}) \arrow[d, bend right, ""{name=A}] & \\ \mathrm{CoAlg}(H^X) \arrow[u, bend right, ""{name=B}] \arrow[from=A, to=B, phantom, "\dashv"] & \mathrm{CoAlg}((\nu T(w))^*) \end{tikzcd}$$

Hence the functor $\mathrm{Alg}((\mu T(w))^*) \colon \mathrm{Alg}(H_X) \to \mathrm{Alg}(H_{w^*(X)})$ preserves initial objects, and the functor $\mathrm{CoAlg}((\nu T(w))^*) \colon \mathrm{CoAlg}(H^X) \to \mathrm{CoAlg}(H^{w^*(X)})$ preserves terminal objects. In particular, this means that

$$(\mu T(w))^*(\mu H(X)) \cong \mu H(w^*(X)) \quad \text{and} \quad (\nu T(w))^*(\nu H(X)) \cong \nu H(w^*(X))$$

showing that μH and νH are fibred functors as required. $\square$

Remark. As already mentioned, one is often interested in preorder fibrations describing some logic. Then the initial H_X-algebra and terminal H^X-coalgebra in Theorem 3 in the *preorder* fibre categories $\mathbb{E}_{\mu T(A)}$ and $\mathbb{E}_{\nu T(A)}$ (respectively) can be computed as least and greatest fixed points of the monotone functions H_X, H^X. The standard formulas are:

$$\mu H(X) = \bigwedge \{y \mid H_X(y) \leq y\} \qquad \text{and} \qquad \nu H(X) = \bigvee \{y \mid y \leq H^X(y)\}.$$

This leads to the familiar properties: $H_X(y) \leq y \Rightarrow \mu H(X) \leq y$ and $y \leq H^X(y) \Rightarrow y \leq \nu H(X)$. In this preorder setting, the latter implication can be strengthened to the so-called *strong coinduction* rule (see *e.g.* [27, 3.2]):

$$y \leq H^X(y) \vee \nu H(X) \qquad \text{implies} \qquad y \leq \nu H(X).$$

This rule is easily derived, since the assumption yields $y \vee \nu H(X) \leq H^X(y \vee \nu H(X))$.

4 Relational proof principles for datafunctors

In the previous section we have studied a fibred functor H over a functor T between base categories of fibrations. Here we shall instantiate this situation by taking T to be a datafunctor and H a "lifting" $\mathrm{Rel}(T)$ of T to a fibred category of relations $\mathrm{Rel}(\mathbb{E})$. Formally, this category $\mathrm{Rel}(\mathbb{E})$ is the total category of a fibration $\begin{smallmatrix}\mathrm{Rel}(\mathbb{E})\\ \downarrow\\ \mathbb{B}\end{smallmatrix}$ of relations, which is obtained from a fibration $\begin{smallmatrix}\mathbb{E}\\ \downarrow\\ \mathbb{B}\end{smallmatrix}$ by change-of-base (or pullback) along the functor $\mathbb{B} \to \mathbb{B}$ given by $A \mapsto A \times A$. Throughout this section we shall assume that the fibration $\begin{smallmatrix}\mathbb{E}\\ \downarrow\\ \mathbb{B}\end{smallmatrix}$ admits the structure required for the constructions that we perform. In particular we shall assume that there is an equality functor $\mathrm{Eq}: \mathbb{B} \to \mathrm{Rel}(\mathbb{E})$, were $\mathrm{Eq}(A) = \coprod_\delta(1A)$ with $1: \mathbb{B} \to \mathbb{E}$ a terminal object functor for the fibration.

We shall define for a datafunctor $T: \mathbb{B}^n \to \mathbb{B}$ a lifting $\mathrm{Rel}(T): \mathrm{Rel}(\mathbb{E})^n \to \mathrm{Rel}(\mathbb{E})$ by induction on the structure of T, adapting the approach of [14, 15]. That is, if T is a projection functor $\mathbb{B}^n \to \mathbb{B}$, then $\mathrm{Rel}(T): \mathrm{Rel}(\mathbb{E})^n \to \mathrm{Rel}(\mathbb{E})$ is the same projection; if T is a constant $A: \mathbf{1} \to \mathbb{B}$, then $\mathrm{Rel}(T)$ is the associated equality constant $\mathrm{Eq}(A) = \coprod_\delta(1A): \mathbf{1} \to \mathrm{Rel}(\mathbb{E})$; if T is $\times$ or $+$ from $\mathbb{B} \times \mathbb{B}$ to $\mathbb{B}$, then $\mathrm{Rel}(T)$ is $\times$ or $+$ from $\mathrm{Rel}(\mathbb{E}) \times \mathrm{Rel}(\mathbb{E})$ to $\mathrm{Rel}(\mathbb{E})$—given by canonical formulas $R \times S = (\pi \times \pi)^*(R) \wedge (\pi' \times \pi')^*(S)$, and $R + S = \coprod_{\kappa \times \kappa}(R) \vee \coprod_{\kappa' \times \kappa'}(S)$, using $\wedge$ and $\vee$ for products and coproducts in the fibres; if T is $S \circ (T_1, \ldots, T_m)$, then $\mathrm{Rel}(T)$ is $\mathrm{Rel}(S) \circ (\mathrm{Rel}(T_1), \ldots, \mathrm{Rel}(T_n))$; and finally, if T is μS or νS, then $\mathrm{Rel}(T)$ is $\mu\mathrm{Rel}(S)$ or $\nu\mathrm{Rel}(S)$, using the fixed points of fibred functors as described in the previous section. Under mild requirements, the lifted functors $\mathrm{Rel}(T)$ are fibred over T (using Proposition 4).

The following definition formalises appropriate interaction between T and $\mathrm{Rel}(T)$. The property that is defined can be proved for all non-recursive data functors, see [15], but in the presence of recursion it has to be required explicitly.

Definition 5. A datafunctor $T: \mathbb{B}^n \to \mathbb{B}$ is said to *respect equality* if there is a (canonical) natural isomorphism

$$\mathrm{Eq} \circ T \overset{\cong}{\Longrightarrow} \mathrm{Rel}(T) \circ \mathrm{Eq}^n.$$

(The canonicity of this isomorphism depends on the structure of T, but does not really play a rôle in what follows.)

A functor $T: \mathbb{B}^{n+1} \to \mathbb{B}$ respecting equality induces by lemma 2 functors between categories of algebras and of coalgebras, namely

$$\begin{array}{ccc} \mathrm{Alg}(T(\vec{X}, -)) & & \mathrm{CoAlg}(T(\vec{X}, -)) \\ {\scriptstyle \mathrm{Alg}(\mathrm{Eq})}\big\downarrow & \text{and} & \big\downarrow{\scriptstyle \mathrm{CoAlg}(\mathrm{Eq})} \\ \mathrm{Alg}(\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), -)) & & \mathrm{CoAlg}(\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), -)) \end{array}$$

for each tuple $\vec{X} \in \mathbb{B}^n$. These functors both use the isomorphism in the above definition, but in different directions. The first functor Alg(Eq) sends an algebra $\alpha: T(\vec{X}, Y) \to Y$ to the composite

$$\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), \mathrm{Eq}(Y)) \xrightarrow{\cong} \mathrm{Eq}(T(\vec{X}, Y)) \xrightarrow{\mathrm{Eq}(\alpha)} \mathrm{Eq}(Y)$$

And the second functor CoAlg(Eq) sends a coalgebra $\beta: Y \to T(\vec{X}, Y)$ to

$$\mathrm{Eq}(Y) \xrightarrow{\mathrm{Eq}(\beta)} \mathrm{Eq}(T(\vec{X}, Y)) \xrightarrow{\cong} \mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), \mathrm{Eq}(Y))$$

In terms of these functors Alg(Eq) and CoAlg(Eq) we define induction and coinduction principles in a logic of a fibration (essentially as in [14, 15]).

Definition 6. Consider a fibration $\genfrac{}{}{0pt}{}{\mathbb{E}}{\downarrow \atop \mathbb{B}}$ with a datafunctor $T: \mathbb{B}^n \times \mathbb{B} \to \mathbb{B}$ respecting equality on its base category, together with the associated lifted functor $\mathrm{Rel}(T): \mathrm{Rel}(\mathbb{E})^n \times \mathrm{Rel}(\mathbb{E}) \to \mathrm{Rel}(\mathbb{E})$. The fibration satisfies

1. the *induction principle w.r.t.* T if for all objects $\vec{X} \in \mathbb{B}^n$ the functor
$$\mathrm{Alg}(T(\vec{X}, -)) \xrightarrow{\mathrm{Alg}(\mathrm{Eq})} \mathrm{Alg}(\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), -))$$
preserves initial objects.
2. the *coinduction principle w.r.t.* T if, similarly, for all $\vec{X} \in \mathbb{B}^n$ the functor
$$\mathrm{CoAlg}(T(\vec{X}, -)) \xrightarrow{\mathrm{CoAlg}(\mathrm{Eq})} \mathrm{CoAlg}(\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), -))$$
preserves terminal objects.

Lemma 7. *If* $\genfrac{}{}{0pt}{}{\mathbb{E}}{\downarrow \atop \mathbb{B}}$ *satisfies induction/coinduction* w.r.t. $T: \mathbb{B}^n \times \mathbb{B} \to \mathbb{B}$ *then both* $\mu T, \nu T : \mathbb{B}^n \to \mathbb{B}$ *respect equality.*

Proof. For each $\vec{X} \in \mathbb{B}$ the induction principle tells us that the carrier of the initial algebra $\mu\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}))$ of the functor $\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X}), -)$ is $\mathrm{Eq}(\mu T(\vec{X}))$. As a result, there is a (canonical) isomorphism $\mu\mathrm{Rel}(T)(\mathrm{Eq}^n(\vec{X})) \cong \mathrm{Eq}(\mu T(\vec{X}))$ between two initial objects. The same holds in the coalgebraic case. $\square$

The next result ensures the validity of induction and coinduction principles in the presence of comprehension and quotients, like in [15].

Theorem 8. *If our fibration admits both comprehension and quotients, then it satisfies the induction and coinduction principles w.r.t. all data functors.*

Proof. Recall that $\mathrm{Eq} = \coprod_\delta \circ 1$. The existence of comprehension as a right adjoint to 1 makes Eq a left adjoint. In the presence of quotients Eq becomes, by definition, a right adjoint. If a datafunctor T respects equality, then, by Lemma 2 the functor Alg(Eq) is a left and CoAlg(Eq) is a right adjoint preserving initial and terminal objects respectively. By induction on the structure of the datafunctors T, one establishes that T respects equality and therefore that induction and coinduction principles hold *w.r.t.* T. $\square$

5 Handling mixed proof obligations

In this section we focus on the special case where we have an initial algebra α or a terminal coalgebra β of the form

$$\nu F(A) \xrightarrow[\cong]{\alpha} A \qquad \text{and} \qquad B \xrightarrow[\cong]{\beta} \mu F(B)$$

involving initial algebras and terminal coalgebras

$$\nu F(A) \xrightarrow{\cong} F(A, \nu F(A)) \qquad \text{and} \qquad F(B, \mu F(B)) \xrightarrow{\cong} \mu F(B)$$

where F is a datafunctor $\mathbb{C} \times \mathbb{C} \to \mathbb{C}$. For convenience, we assume that F has only two parameters instead of $n+1$, since there is no fundamental difference. The problem with the above α and β is that the fixed points $\nu F(A)$ and $\mu F(B)$ are "on the wrong side".

The same inconvenience occurs at a logical level. We shall explain this in a setting where we have a logical preorder structure $\vdash$ representing entailment, instead of a type theoretical (non-preorder) structure $\to$ describing explicit proof-terms. In this setting we assume that both the induction and coinduction principles hold *w.r.t* F.

We first consider the second situation with terminal coalgebra $\beta: B \xrightarrow{\cong} \mu F(B)$. Assume that we have a relation R on B, and that we wish to show by coinduction that R is contained in the equality relation $\mathrm{Eq}(B)$ on B (*i.e.* that $R \vdash \mathrm{Eq}(B)$). By Definition 6 we know that $\mathrm{Eq}(B)$ is the terminal coalgebra of the lifted functor $\mathrm{Rel}(\mu F)$, so it suffices to show that R carries a $\mathrm{Rel}(\mu F)$ coalgebra structure $R \to \mathrm{Rel}(\mu F)(R)$ over β: such a coalgebra results in a unique coalgebra homomorphism $R \to \mathrm{Eq}(B)$ over $\mathsf{coreduce}(\beta) = id$, see Theorem 3 (and its proof). It thus yields $R \vdash \mathrm{Eq}(B)$ as required (since we assumed the fibres to be preorders).

Hence in order to prove $R \vdash \mathrm{Eq}(B)$ we seek a coalgebra structure $R \to \mathrm{Rel}(\mu F)(R)$ over β. Equivalently, an entailment $R \vdash (\beta \times \beta)^*(\mathrm{Rel}(\mu F))(R)$ over B, or an entailment $\coprod_{\beta\times\beta}(R) \vdash \mathrm{Rel}(\mu F)(R)$ over $\mu F(B)$. Since $\mu F(B)$ is an initial algebra, we may try to prove the latter by induction. We can make this explicit by unraveling the left hand side $\coprod_{\beta\times\beta}(R)$. It can be described as

$$\begin{aligned}&\textstyle\coprod_{\beta\times\beta}(R)\\ &\quad= \textstyle\coprod_{\pi'\times\pi'} \left(\pi^*(\beta\times id)^*\mathrm{Eq}(\mu F(B)) \wedge \pi'^*(\beta\times id)^*\mathrm{Eq}(\mu F(B)) \wedge (\pi\times\pi)^*(R)\right).\end{aligned}$$

Hence, what we have to prove is an entailment over $\mu F(B)$:

$$\begin{aligned}&\textstyle\coprod_{\pi'\times\pi'}(\pi^*(\beta\times id)^*\mathrm{Eq}(\mu F(B)) \wedge \pi'^*(\beta\times id)^*\mathrm{Eq}(\mu F(B)) \wedge (\pi\times\pi)^*(R))\\ &\quad\vdash \mathrm{Rel}(\mu F)(R).\end{aligned}$$

It is equivalent to an entailment over $B \times \mu F(B)$:

$$\begin{aligned}&\pi^*(\beta\times id)^*\mathrm{Eq}(\mu F(B)),\ \pi'^*(\beta\times id)^*\mathrm{Eq}(\mu F(B))\\ &\quad\vdash (\pi'\times\pi')^*\left((\pi\times\pi)^*(R) \Rightarrow \mathrm{Rel}(\mu F)(R)\right),\end{aligned}$$

where the comma on the left hand side of the turnstile describes the meet $\wedge$, and where $\Rightarrow$ on the right hand side is the associated implication (or exponent). Such an entailment can be proved by a "double" induction over the initial algebra $\mu F(B)$. Describing the precise details of this in categorical terms requires us to formally deal with strength at a logical level (see also [15]), leading us too far from the main points of the present paper. Also, we skip these details because this presents no problem in a proof assistant like PVS.

We turn to the initial algebra $\alpha: \nu F(A) \xrightarrow{\cong} A$, described in the beginning of this section. Suppose now that we wish to prove that a relation R on A is reflexive[4]. We can try to do so by induction, since A is an inductive type. This means that we wish to show $\mathrm{Eq}(A) \vdash R$ in the fibre over $A \times A$, using the binary induction principle. Since $\mathrm{Eq}(A)$ is by Definition 6 the initial algebra of $\mathrm{Rel}(\nu F)$, it suffices to show that R carries a $\mathrm{Rel}(\nu F)$-algebra $\mathrm{Rel}(\nu F)(R) \to R$ over α. Equivalently, this involves proving an entailment

$$\mathrm{Rel}(\nu F)(R) \vdash (\alpha \times \alpha)^*(R) \tag{3}$$

over $\nu F(A)$. In this case we do not know of a general way of massaging this goal into an equivalent obligation which we can prove by coinduction. But, in the special case where the right hand side is of suitable form, there is more we can do.

Let us assume that the right hand side $(\alpha \times \alpha)^*(R)$ of (3) is a conjunction of implications with substituted equations on terminal coalgebras as conclusions, *i.e.* that

$$(\alpha \times \alpha)^*(R) = \bigwedge\nolimits_i S_i \Rightarrow (u_i \times v_i)^*(\mathrm{Eq}(Z_i))$$

where u_i, v_i are parallel maps $\nu F(A) \to Z_i$ to the carrier of some terminal coalgebra $\gamma_i: Z_i \xrightarrow{\cong} T_i(Z_i)$. The above proof obligation (3) is then equivalent to the collection of proof obligations

$$\mathrm{Rel}(\nu F)(R) \vdash S_i \Rightarrow (u_i \times v_i)^*(\mathrm{Eq}(Z_i)).$$

Or, equivalently, to

$$\textstyle\coprod_{u_i \times v_i}(\mathrm{Rel}(\nu F)(R) \wedge S_i) \vdash \mathrm{Eq}(Z_i).$$

The latter can be proved by coinduction (on the terminal coalgebras $\gamma_i: Z_i \xrightarrow{\cong} T_i(Z_i)$), namely, by proving that we have $\mathrm{Rel}(T_i)$-coalgebra structures on the left hand side $\coprod_{u_i \times v_i}(\mathrm{Rel}(\nu F)(R) \wedge S_i)$ given by entailments

$$\textstyle\coprod_{u_i \times v_i}(\mathrm{Rel}(\nu F)(R) \wedge S_i) \vdash (\gamma_i \times \gamma_i)^*(\mathrm{Rel}(T_i)(\coprod_{u_i \times v_i}(\mathrm{Rel}(\nu F)(R) \wedge S_i))).$$

These two approaches for handling successive fixed points of different kinds are used in the next section.

[4] Often, the relation R will be of the form $\coprod_\delta(P)$ for a predicate P on A. Showing that R is reflexive is then the same as showing that P holds on A (since $\coprod_\delta(P)(x, y) \Leftrightarrow x = y \wedge P(x)$). In fact, in this preorder situation the case $R = \coprod_\delta(P)$ describes the general case: showing $\mathrm{Eq}(A) \vdash R$ is then equivalent to showing $\mathrm{Eq}(A) \vdash R \wedge \mathrm{Eq}(A)$. But by Frobenius $R \wedge \mathrm{Eq}(A) = R \wedge \coprod_\delta(1A) = \coprod_\delta(\delta^*(R) \wedge 1(A)) = \coprod_\delta(\delta^*(R))$, so that we may take as predicate $P = \delta^*(R)$.

6 Proof principles in use

Having described the general proof principles for iterated datafunctors in the previous sections we proceed with the presentation of some sample proofs. This should enable the reader to grasp the differences between pure induction and coinduction on the one hand and the iterated mixing of them on the other.

We define a composition operation $(x, y) \mapsto \mathsf{comp}(x, y)$ for each of the tree examples from Section 2, by appending a tree y at all leaves of the tree x. A leaf, hereby, is a tree which has no subtrees. This composition operation can be defined for all types of trees. We then prove, using appropriate proof principles for the different types of trees, that appending the empty tree τ yields the given tree itself, *i.e.* that $\mathsf{comp}(x, \tau) = x$. This is a non–trivial matter to prove formally.

These proofs are carried out using the specification and verification tool PVS [25, 24]. PVS is a tool which comprises a specification language based on higher order logic with predicate subtyping and dependent types, an interactive theorem prover and several tools for handling modular specifications, pretty printing, and generating readable LaTeX specification and proof files.

The formalisation of the employed proof principles involves the specification of coinductive sequences (colists) and the full range of our example trees. These specifications have some value in themself because they illustrate how PVS (or a similar tool) could (in future versions) incorporate iterated data structures and their proof principles.

We closely follow the abstract categorical development in this paper. It is an indispensable road map for our implementation in PVS. PVS itself supplies inductive (non iterated) abstract datatypes including the required lists and coproducts. Colists, however, have to be specified explicitly by providing the structure map next, describing the notion of homomorphism, and axiomatising terminality, as described in [11]. The (co)inductive trees are then formalised in a similar style in terms of the underlying (co)lists.

Relations (objects in the total category) are defined using predicates or, equivalently, characteristic functions. A fibre is a poset category with relations on a common type as objects and implication as (vertical) morphisms. The logic of PVS provides for the logical connectives yielding the appropriate fibred structure. An endofunctor on a fibre is then an implication preserving mapping from relations to relations (on the same type). We define general theories for fixed points of such endofunctors. Having introduced these basic definitions we formalise the specific endofunctors H_R and H^R from section 3 for lists and colists. Then we require the existence of fixed points in the fibres by suitable axioms and obtain the relational lifting.

The bisimulation and induction rules are introduced in terms of these fixed points (the polynomial liftings are "programmed" directly). PVS has a notion of comprehension via predicate subtypes and quotients can be implemented, so that the induction and coinduction principle can be proved within PVS. For all datatypes we provide strength, functorial properties, and the unrolling of the proof rules as used below.

The following subsections explain the sample proofs in more detail. We use a notion of a goal oriented proof tree starting with the goal sequence at the bottom and developing the proof in a bottom-up fashion. Moreover we simplify the injections for the coproduct in the functor $F(Y, X) = F_A(Y, X) = A \times Y \times X + 1$ from Example 1 by writing a triple $(a, y, x) \in A \times Y \times X$ for an element given by the left inclusion and $* \in 1$ for the right inclusion. The lifting $\mathrm{Rel}(F)$ of F to a (binary) functor on relations is then described as follows. For a relation R on Y and a relation S on X,

$$\begin{aligned}
&\mathrm{Rel}(F)(R, S) \subseteq (A \times Y \times X + 1) \times (A \times Y \times X + 1) \\
&\quad = \{\langle (a, y, x), (a', y', x') \rangle \mid a = a' \text{ and } R(y, y') \text{ and } S(x, x')\} \cup \{\langle *, * \rangle\}.
\end{aligned}$$

For this functor F we consider the initial algebra and terminal coalgebra

$$\mu F = \mu X.\, F(-, X) = \mathsf{List}(A \times -) \quad \text{and} \quad \nu F = \nu X.\, F(-, X) = \mathsf{Colist}(A \times -),$$

with structure maps

$$A \times Y \times \mathsf{List}(A \times Y) + 1 \xrightarrow[\cong]{[\mathsf{cons}, \mathsf{nil}]} \mathsf{List}(A \times Y)$$

$$\mathsf{Colist}(A \times Y) \xrightarrow[\cong]{\mathsf{next}} A \times Y \times \mathsf{Colist}(A \times Y) + 1$$

The associated functors on the fibres—as in (2)—are as follows. For R a relation on Y, and S a relation on $\mathsf{List}(A \times Y)$,

$$\begin{aligned}
&\mathrm{Rel}(F)_R(S) \subseteq \mathsf{List}(A \times Y) \times \mathsf{List}(A \times Y) \\
&\quad = \{\langle \mathsf{cons}(a, x, \ell), \mathsf{cons}(b, y, k) \rangle \mid a = b \text{ and } R(x, y) \text{ and } S(\ell, k)\} \cup \{\langle \mathsf{nil}, \mathsf{nil} \rangle\}.
\end{aligned}$$

And for S a relation on $\mathsf{Colist}(A \times Y)$,

$$\begin{aligned}
&\mathrm{Rel}(F)^R(S) \subseteq \mathsf{Colist}(A \times Y) \times \mathsf{Colist}(A \times Y) \\
&\quad = \{\langle \ell, k \rangle \mid \mathsf{next}(\ell) = (a, x, \ell'), \mathsf{next}(k) = (b, y, k') \text{ with} \\
&\qquad\quad a = b \text{ and } R(x, y) \text{ and } S(\ell', k')\} \cup \{\langle \ell, k \rangle \mid \mathsf{next}(\ell) = \mathsf{next}(k) = *\}.
\end{aligned}$$

By definition, there are equalities

$$\begin{aligned}
\mathrm{Rel}(\mu F)(R) &= (\mu \mathrm{Rel}(F))(R) = \mu(\mathrm{Rel}(F)_R) \\
\mathrm{Rel}(\nu F)(R) &= (\nu \mathrm{Rel}(F))(R) = \nu(\mathrm{Rel}(F)^R).
\end{aligned}$$

Induction for $\mathsf{List}(A \times Y)$ takes the form of an equality (between initial algebras):

$$\mu(\mathrm{Rel}(F)_{\mathrm{Eq}(Y)}) = \mathrm{Eq}(\mathsf{List}(A \times Y)).$$

And coinduction for $\mathsf{Colist}(A \times Y)$ amounts to:

$$\nu(\mathrm{Rel}(F)^{\mathrm{Eq}(Y)}) = \mathrm{Eq}(\mathsf{Colist}(A \times Y)).$$

Below we will consider the following two initial algebras and two terminal coalgebras.

$$\begin{aligned}
\mathsf{br}\colon \mathsf{List}(A \times \mathsf{FTreeF}(A)) &\xrightarrow{\cong} \mathsf{FTreeF}(A) \\
\mathsf{br}\colon \mathsf{Colist}(A \times \mathsf{ITreeF}(A)) &\xrightarrow{\cong} \mathsf{ITreeF}(A) \\
\mathsf{br}\colon \mathsf{FTreeI}(A) &\xrightarrow{\cong} \mathsf{List}(A \times \mathsf{FTreeI}(A)) \\
\mathsf{br}\colon \mathsf{ITreeI}(A) &\xrightarrow{\cong} \mathsf{Colist}(A \times \mathsf{ITreeI}(A))
\end{aligned}$$

The associated induction and coinduction principles then take the following form.

$$\begin{array}{ll}
\text{Induction for } \mathsf{FTreeF}(A): & \mu\mathrm{Rel}(\mu F) = \mathrm{Eq}(\mathsf{FTreeF}) \\
\text{Induction for } \mathsf{ITreeF}(A): & \mu\mathrm{Rel}(\nu F) = \mathrm{Eq}(\mathsf{ITreeF}) \\
\text{Coinduction for } \mathsf{FTreeI}(A): & \nu\mathrm{Rel}(\mu F) = \mathrm{Eq}(\mathsf{FTreeI}) \\
\text{Coinduction for } \mathsf{ITreeI}(A): & \nu\mathrm{Rel}(\nu F) = \mathrm{Eq}(\mathsf{ITreeI}).
\end{array}$$

We formalise these principles in PVS (via axioms), and use them to obtain the results in the next four Subsections 6.1–6.4. This formalisation makes use of our earlier formalisation of colists (or sequences), see [11].

We emphasise that these principles can be generated automatically from the form of the functor (or signature) describing the relevant datatype. The formulation of our proof principles is entirely syntax-driven, and is thus suitable for incorporation in a proof-tool.

6.1 Finitely branching trees of finite depth

Finitely branching trees with finite depth are purely inductive. We briefly sketch the definition of composition and the required proof. In order not to complicate matters unnecessarily we avoid the usage of strength in inductive definitions and proofs. Therefore the composition for inductive trees (finite depth) here and in subsection 6.2 is unary by fixing the second argument.

Composition is defined inductively as $\mathsf{comp}_y = \mathsf{reduce}(\mathsf{comp\text{-}struct}_y)$ where

$$\begin{aligned}
&\mathsf{comp\text{-}struct}_y : \mathsf{List}(A \times \mathsf{FTreeF}(A)) \to \mathsf{FTreeF}(A); \\
&\quad \ell \mapsto \begin{cases} y & \text{if } \ell = \mathsf{nil} \\ \mathsf{br}(\ell) & \text{else} \end{cases}
\end{aligned}$$

The empty tree τ is constructed from the empty list as $\mathsf{br}(\mathsf{nil})$. In order to prove the equality $\mathsf{comp}_\tau(x) = x$ we show that $R = \{(x,x) \,|, \mathsf{comp}_\tau(x) = x\}$ contains the equality relation on trees. Using that $\mathrm{Rel}(\mu F)(R)$ is a least fixed point of $\mathrm{Rel}(F)_R$ we obtain

$$\dfrac{\dfrac{\mathrm{Rel}(F)_R((\mathsf{br} \times \mathsf{br})^*(R)) \vdash (\mathsf{br} \times \mathsf{br})^*(R)}{\mathrm{Rel}(\mu F)(R) \vdash (\mathsf{br} \times \mathsf{br})^*(R)}}{\mathrm{Eq} \vdash R}$$

The upper obligation amounts to two cases, namely $R(\mathsf{br}(\mathsf{nil}), \mathsf{br}(\mathsf{nil}))$ and

$$R(x,y) \wedge R(\mathsf{br}(\ell), \mathsf{br}(k)) \vdash R(\mathsf{br}(\mathsf{cons}((a,x),\ell)), \mathsf{br}(\mathsf{cons}((a,y),k))).$$

For our particular relation $R = \{(x, x) \mid, \mathsf{comp}_{\mathcal{T}}(x) = x\}$ the first case yields

$$\mathsf{comp}_{\mathcal{T}}(\mathsf{br}(\mathsf{nil})) = \mathsf{br}(\mathsf{nil})$$

and the second becomes

$$\begin{aligned} & x = y \wedge \mathsf{comp}_{\mathcal{T}}(x) = x \wedge \mathsf{br}(\ell) = \mathsf{br}(k) \wedge \mathsf{comp}_{\mathcal{T}}(\mathsf{br}(\ell)) = \mathsf{br}(\ell) \\ & \quad \vdash \mathsf{br}(\mathsf{cons}((a, x), \ell)) = \mathsf{br}(\mathsf{cons}((a, y), k)) \\ & \qquad \wedge \mathsf{comp}_{\mathcal{T}}(\mathsf{br}(\mathsf{cons}((a, x), \ell))) = \mathsf{br}(\mathsf{cons}((a, x), \ell)). \end{aligned}$$

This can be easily discharged using above definitions and the fact that br is an isomorphism.

6.2 Possibly infinitely branching trees of finite depth

Here we expect a mixed argument for our proof. According to the structure of $\mathsf{ITreeF}(A)$ we start by induction, the inner proof is then performed by coinduction.

Composition is defined inductively as in section 6.1 but it involves colists instead of lists.

$$\begin{aligned} & \mathsf{comp\text{-}struct}_y : \mathsf{Colist}(A \times \mathsf{ITreeF}(A)) \to \mathsf{ITreeF}(A); \\ & \quad \ell \mapsto \begin{cases} y & \text{if } \mathsf{next}(\ell) = * \\ \mathsf{br}(\ell) & \text{else} \end{cases} \end{aligned}$$

The empty tree is constructed from the empty colist. The relation to be proved reflexive is again $R = \{(x, x) \mid, \mathsf{comp}_{\mathcal{T}}(x) = x\}$, a predicate transformed into a relation. $\mathsf{ITreeF}(A)$ yields the following obligation

$$\frac{\mathrm{Rel}(\nu F)(R) \vdash (\mathsf{br} \times \mathsf{br})^*(R)}{\mathrm{Eq} \vdash R}$$

Fortunately we can simplify $(\mathsf{br} \times \mathsf{br})^*(R)$ to a conjunction of substituted equalities over colists as explained in section 5.

$$\begin{aligned} (\mathsf{br} \times \mathsf{br})^*(R) &= \{(\ell, k) \mid, \mathsf{br}(\ell) = \mathsf{br}(k) \wedge \mathsf{comp}_{\mathcal{T}}(\mathsf{br}(\ell)) = \mathsf{br}(\ell)\} \\ &= \{(\ell, k) \mid, \ell = k \wedge \mathsf{br}^{-1}(\mathsf{comp}_{\mathcal{T}}(\mathsf{br}(\ell))) = k\} \\ &= \mathrm{Eq}(\mathsf{Colist}(A \times \mathsf{ITreeF}(A))) \\ &\qquad \wedge ((\mathsf{br}^{-1} \circ \mathsf{comp}_{\mathcal{T}} \circ \mathsf{br}) \times id)^*(\mathrm{Eq}(\mathsf{Colist}(A \times \mathsf{ITreeF}(A)))) \end{aligned}$$

This yields the obligations over $\mathsf{Colist}(A \times \mathsf{ITreeF}(A))$:

$$\mathrm{Rel}(\nu F)(R) \vdash \mathrm{Eq} \quad \text{and} \quad \mathrm{Rel}(\nu F)(R) \vdash ((\mathsf{br}^{-1} \circ \mathsf{comp}_{\mathcal{T}} \circ \mathsf{br}) \times id)^*(\mathrm{Eq})$$

which can be discharged by coinduction on colists, based on the fact that equality $\mathrm{Eq}(\mathsf{Colist}(A \times \mathsf{ITreeF}(A))$ on $A \times \mathsf{ITreeF}(A)$ is the terminal $\mathrm{Rel}(F)^{\mathrm{Eq}}$-coalgebra. The first obligation amounts to

$$\mathrm{Rel}(\nu F)(R) \vdash (\mathsf{next} \times \mathsf{next})^*(\mathrm{Rel}(F)(\mathrm{Eq}(A \times \mathsf{ITreeF}(A)), \mathrm{Rel}(\nu F)(R)))$$

which is trivially true as $R(x,y)$ implies $x = y$. The second one can be massaged as follows, for $f = \mathsf{br}^{-1} \circ \mathsf{comp}_\tau \circ \mathsf{br}$.

$$\cfrac{\cfrac{\cfrac{\mathrm{Rel}(\nu F)(R) \vdash (f \times id)^*(\mathrm{Rel}(F)^{\mathrm{Eq}}(\coprod_{f \times id}(\nu \mathrm{Rel}(F)^R)))}{\coprod_{f \times id}(\mathrm{Rel}(\nu F)(R)) \vdash \mathrm{Rel}(F)^{\mathrm{Eq}}(\coprod_{f \times id}(\mathrm{Rel}(\nu F)(R)))}}{\coprod_{f \times id}(\mathrm{Rel}(\nu F)(R)) \vdash \mathrm{Eq}} \begin{pmatrix}\text{coinduction} \\ \text{for } \mathsf{Colist}\end{pmatrix}}{\mathrm{Rel}(\nu F)(R) \vdash (f \times id)^*(\mathrm{Eq})}$$

The upper sequence of this proof tree remains to be verified. Suppose $(\ell, k) \in \nu\mathrm{Rel}(F)^R = \mathrm{Rel}(\nu F)(R)$ and $\mathsf{next}(\ell) = \mathsf{next}(k) = *$ then

$$\mathsf{next}(\mathsf{br}^{-1}(\mathsf{comp}_\tau(\mathsf{br}(\ell)))) = \mathsf{next}(\mathsf{br}^{-1}(\tau)) = *$$

and therefore (ℓ, k) satisfies the right hand side of the sequence. Suppose now $\mathsf{next}(\ell) = (a, x, \ell')$ and $\mathsf{next}(k) = (b, y, k')$ then $f(\ell) = \mathsf{Colist}(id \times \mathsf{comp}_\tau)(\ell)$ and

$$\mathsf{next}(f(\ell)) = (a, \mathsf{comp}_\tau(x), \mathsf{Colist}(id \times \mathsf{comp}_\tau)(\ell'))$$

But by assumption we have $a = b$, $\mathsf{comp}_\tau(x) = x = y$, and

$$(\mathsf{Colist}(id \times \mathsf{comp}_\tau)(\ell'), k') \in \textstyle\coprod_{f \times id}(\nu\mathrm{Rel}(F)^R)).$$

ensuring that (ℓ, k) satisfies the right hand side of the sequence.

6.3 Finitely branching trees of possibly infinite depth

Composition for $\mathsf{FTreeI}(A)$ needs to be defined coinductively, while the required structure map is defined inductively on lists.

$$\begin{array}{l} \mathsf{comp\text{-}struct} : \mathsf{FTreeI}(A) \times \mathsf{FTreeI}(A) \to \mathsf{List}(A \times (\mathsf{FTreeI}(A) \times \mathsf{FTreeI}(A))); \\ \quad (x, y) \mapsto \begin{cases} \mathsf{merge\text{-}middle}(\mathsf{br}(y), x) & \text{if } \mathsf{br}(x) = \mathsf{nil} \\ \mathsf{merge\text{-}last}(\mathsf{br}(x), y) & \text{else} \end{cases} \end{array}$$

The functions $\mathsf{merge\text{-}middle} = \mathsf{List}(id \times \mathsf{c}) \circ \theta$ and $\mathsf{merge\text{-}last} = \theta$ make use of the *strength* $\theta : \mathsf{List}(A) \times B \to \mathsf{List}(A \times B)$[5], the commutativity of products $\mathsf{c} : A \times B \to B \times A$, and of the functor properties of the List–datatype. The empty tree τ is obtained by applying the inverse of the destructor br to the empty list nil.

The proof of our equation $\mathsf{comp}(x, \tau) = x$ requires a mixed argument. We start off by coinduction, then we need to switch to induction on lists. The coinduction principle for $\mathsf{FTreeI}(A)$ states that the equation holds if there exists a relation R on trees such that $R(\mathsf{comp}(x, \tau), x)$ and

$$R \vdash (\mathsf{br} \times \mathsf{br})^*(\mathrm{Rel}(\mu F)(R)) \quad \text{or equivalently} \quad \textstyle\coprod_{\mathsf{br} \times \mathsf{br}}(R) \vdash \mathrm{Rel}(\mu F)(R).$$

[5] Defined be induction.

The latter obligation lies in the fibre over lists and reads

$$\begin{aligned}&\forall \ell, k : \mathsf{List}(A \times \mathsf{FTreeI}(A)), x, y : \mathsf{FTreeI}(A).\\ &\quad R(x, y) \wedge \mathsf{br}(x) = \ell \wedge \mathsf{br}(y) = k \Rightarrow \mathrm{Rel}(\mu F)_R(\ell, k).\end{aligned}$$

We take $R = \{(\mathsf{comp}(x, \tau), x) \mid x \in \mathsf{FTreeI}(A)\}$ and proceed with induction on the lists ℓ and k. If both ℓ and k are nil then unfolding the fixed point $\mathrm{Rel}(\mu F)(R) = \mu\mathrm{Rel}(F)_R \cong \mathrm{Rel}(F)_R(\mu\mathrm{Rel}(F)_R)$ does the job. If either $\ell = \mathsf{nil}$ and $k = \mathsf{cons}(b, y', k')$ or vice versa then for all pairs (x, y) with $\mathsf{br}(x) = \ell$ $\mathsf{br}(y) = k$ the relation $R(x, y)$ does not hold because this would require $x = \mathsf{comp}(y, \tau)$. But we have $\mathsf{br}(x) = \mathsf{nil}$ and $\mathsf{br}(\mathsf{comp}(y, \tau)) \neq \mathsf{nil}$. For the case $\ell = \mathsf{cons}(a, x', \ell')$ and $k = \mathsf{cons}(b, y', k')$ the induction hypothesis holds for ℓ' and k' Assume $R(x, y)$ for all $\mathsf{br}(x) = \mathsf{cons}(a, x', \ell')$ and $\mathsf{br}(y) = \mathsf{cons}(b, y', k')$. It remains to show that $a = b$, $R(x', y')$, and $\mu\mathrm{Rel}(F)_R(\ell', k')$ but

$$\mathsf{br}(\mathsf{comp}(y, \tau)) = \mathsf{cons}((b, \mathsf{comp}(y', \tau)), \mathsf{List}(id \times \mathsf{comp})(\mathsf{merge\text{-}last}(k', \tau)))$$

which shows the first two cases. For $\mu\mathrm{Rel}(F)_R(\ell', k')$, we use the induction hypothesis and have to show, that there is a pair (x, y) in R such that $\mathsf{br}(x) = \ell'$ and $\mathsf{br}(y) = k'$. The inverse of br applied to ℓ' and k' does the job as

$$\ell' = \mathsf{tl}(\mathsf{br}(x)) = \mathsf{tl}(\mathsf{br}(\mathsf{comp}(y, \tau))) = \mathsf{List}(id \times \mathsf{comp})(\mathsf{merge\text{-}last}(k', \tau))$$

and therefore $\ell' = \mathsf{br}(\mathsf{comp}(\mathsf{br}^{-1}(k'), \tau))$.

6.4 Infinitely branching—infinite depth

The composition for the datatype of possibly infinite trees with possibly infinite branching is defined coinductively.

$$\begin{aligned}&\mathsf{comp\text{-}struct}_t : \mathsf{ITreeI}(A) \times \mathsf{ITreeI}(A) \to \mathsf{Colist}(A \times (\mathsf{ITreeI}(A) \times \mathsf{ITreeI}(A)));\\ &\quad (x, y) \mapsto \begin{cases} \mathsf{merge\text{-}middle}(\mathsf{br}(y), x) & \text{if } \mathsf{next}(\mathsf{br}(x)) = * \\ \mathsf{merge\text{-}last}(\mathsf{br}(x), y) & \text{else} \end{cases}\end{aligned}$$

The functions $\mathsf{merge\text{-}middle}$ and $\mathsf{merge\text{-}last}$ resemble those for the finitely branching case, but make use of the strength and functorial properties of Colist (which are of coinductive nature). The required composition is then defined as $\mathsf{comp} = \mathsf{coreduce}(\mathsf{comp\text{-}struct})$. The empty tree is defined via the inverse of br and the empty colist ϵ, namely $\tau = \mathsf{br}^{-1}(\epsilon)$. In order to prove $\mathsf{comp}(x, \tau) = x$ it suffices to show that $R = \{(\mathsf{comp}(x, \tau), x) \mid, x \in \mathsf{ITreeI}(A)\} \subseteq \mathsf{ITreeI}(A) \times \mathsf{ITreeI}(A)$ is a bisimulation *w.r.t.* the lifted datatype functor, *i.e.*

$$\frac{R \vdash (\mathsf{br} \times \mathsf{br})^* \mathrm{Rel}(\nu F)(R)}{R \vdash \mathrm{Eq}}$$

The relational lifting of the colist functor $\mathrm{Rel}(\nu F)(R)$ is by definition the fixed point $\nu\mathrm{Rel}(F)^R$. Therefore

$$\cfrac{\cfrac{\cfrac{R \vdash (\mathsf{br} \times \mathsf{br})^*(\mathrm{Rel}(F)^R(\coprod_{\mathsf{br}\times\mathsf{br}}(R)))}{\coprod_{\mathsf{br}\times\mathsf{br}}(R) \vdash \mathrm{Rel}(F)^R(\coprod_{\mathsf{br}\times\mathsf{br}}(R))}}{\coprod_{\mathsf{br}\times\mathsf{br}}(R) \vdash \nu\mathrm{Rel}(F)^R}}{R \vdash (\mathsf{br} \times \mathsf{br})^*(\nu\mathrm{Rel}(F)^R)}$$

by coinduction. The upper line spells out as:

$$R \vdash \left\{ (x, y) \left| \begin{array}{l} \mathsf{next}(\mathsf{br}(x)) = \mathsf{next}(\mathsf{br}(y)) = * \vee \\ \mathsf{next}(\mathsf{br}(x)) = (a, t, \ell) \wedge \mathsf{next}(\mathsf{br}(y)) = (b, s, k) \\ \quad \wedge\ a = b \wedge R(t, s) \wedge \coprod_{\mathsf{br}\times\mathsf{br}}(R)(\ell, k) \end{array} \right. \right\}$$

Suppose now $R(x, y)$, *i.e.* $x = \mathsf{comp}(y, \tau)$ and, for the upper case, $\mathsf{next}(\mathsf{br}(y)) = *$ (y is the empty tree). Then

$$\begin{aligned} \mathsf{next}(\mathsf{br}(\mathsf{comp}(y.\tau))) &= \mathsf{next}(\mathsf{Colist}(id \times \mathsf{comp})(\mathsf{comp\text{-}struct}(y, \tau))) \\ &= \mathsf{next}(\mathsf{Colist}(id \times \mathsf{comp})(\epsilon)) \\ &= * \end{aligned}$$

For the lower case assume that $\mathsf{next}(\mathsf{br}(y)) = (b, s, k)$. Then, by applying the definitions,

$$\mathsf{next}(\mathsf{br}(\mathsf{comp}(y, \tau))) = (b, \mathsf{comp}(s, \tau), \mathsf{Colist}(id \times \mathsf{comp})(\mathsf{merge\text{-}last}(k, \tau)))$$

As, by assumption, $R(\mathsf{comp}(s, \tau), s)$ is valid, it remains to prove the last clause, namely the membership

$$\begin{aligned} &(\mathsf{Colist}(id \times \mathsf{comp})(\mathsf{merge\text{-}last}(k, \tau)), k) \\ &\quad \in \{(\ell, k) \mid, \exists (x, y). R(x, y) \wedge \mathsf{br}(x) = \ell \wedge \mathsf{br}(y) = k\}. \end{aligned}$$

So far we simply rewrote our definitions, the last obligation, however, requires a suitable instantiation of the existential quantifier. In our simple case there exists only the instantiation via the inverse of br, which yields the obligation

$$R(\mathsf{br}^{-1}(\mathsf{Colist}(id \times \mathsf{comp})(\mathsf{merge\text{-}last}(k, \tau))), \mathsf{br}^{-1}(k))$$

that is

$$\mathsf{Colist}(id \times \mathsf{comp})(\mathsf{merge\text{-}last}(k, \tau)) = \mathsf{br}(\mathsf{comp}(\mathsf{br}^{-1}(k), \tau))$$

which is shown by rewriting the right hand side.

7 Conclusion

We have shown how to obtain proof principles for datatypes with iterated recursion from a categorical analysis, and how to apply these principles using a modern proof tool.

References

1. P. Aczel. *Non-well-founded sets.* CSLI Lecture Notes 14, Stanford, 1988.
2. R.C. Backhouse, P. Chisholm, G. Malcolm, and E. Saaman. Do-it-yourself type theory. *Formal Aspects of Comp.*, 1:19–84, 1989.
3. F. Borceux. *Handbook of Categorical Algebra*, volume 50, 51 and 52 of *Encyclopedia of Mathematics.* Cambridge Univ. Press, 1994.
4. J.R.B. Cockett and T. Fukushima. About charity. Technical Report 92/480/18, Dep. Comp. Sci., Univ. Calgary, 1992.
5. J.R.B. Cockett and D. Spencer. Strong categorical datatypes II: A term logic for categorical programming. *Theor. Comp. Sci.*, 139:69–113, 1995.
6. Th. Coquand and Ch. Paulin. Inductively defined types. In P. Martin-Löf and G. Mints, editors, *COLOG 88 International conference on computer logic*, number 417 in Lect. Notes Comp. Sci., pages 50–66. Springer, Berlin, 1988.
7. P. Dybjer. Inductive families. *Formal Aspects of Comp.*, 6:440–465, 1994.
8. M.P. Fiore. A coinduction principle for recursive data types based on bisimulation. *Inf. & Comp.*, 127(2):186–198, 1996.
9. E. Giménez. Implementation of co-inductive types in Coq: an experiment with the Alternating Bit Protocol. In S. Berardi and M. Coppo, editors, *Types for Proofs and Programs*, number 1158 in Lect. Notes Comp. Sci., pages 135–152. Springer, Berlin, 1996.
10. J.A. Goguen, J. Thatcher, and E. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology*, pages 80–149. Prentice Hall, 1978.
11. U. Hensel and B. Jacobs. Coalgebraic theories of sequences in PVS. Techn. Rep. CSI-R9708, Comput. Sci. Inst., Univ. of Nijmegen, 1997.
12. U. Hensel and D. Spooner. A view on implementing processes: Categories of circuits. In M. Haveraaen, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Type Specification*, number 1130 in Lect. Notes Comp. Sci., pages 237–254. Springer, Berlin, 1996.
13. C. Hermida. Some properties of Fib as a fibred 2-category. *Journ. Pure & Appl. Algebra*, 1997, to appear.
14. C. Hermida and B. Jacobs. An algebraic view of structural induction. In L. Pacholski and J. Tiuryn, editors, *Computer Science Logic 1994*, number 933 in Lect. Notes Comp. Sci., pages 412–426. Springer, Berlin, 1995.
15. C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. Full version of [14], 1996.
16. B. Jacobs. Parameters and parametrization in specification using distributive categories. *Fund. Informaticae*, 24(3):209–250, 1995.
17. B. Jacobs. Invariants, bisimulations and the correctness of coalgebraic refinements. Techn. Rep. CSI-R9704, Comput. Sci. Inst., Univ. of Nijmegen, 1997.
18. B. Jacobs. *Categorical Logic and Type Theory.* 1998, to appear.

19. B. Jay. Data categories. In M.E. Houle and P.Eades, editors, *Computing: The Australasian Theory Symposium Proceedings*, number 18 in Australian Comp. Sci. Comm., pages 21–28, 1996.
20. M. Makkai. The fibrational formulation of intuitionistic predicate logic I: completeness according to Gödel, Kripke, and Läuchli. Part 1. *Notre Dame Journ. Formal Log.*, 34(3):334–377, 1993.
21. M. Makkai. The fibrational formulation of intuitionistic predicate logic I: completeness according to Gödel, Kripke, and Läuchli. Part 2. *Notre Dame Journ. Formal Log.*, 34(4):471–499, 1993.
22. N.P. Mendler. Inductive types and type constraints in second-order lambda calculus. *Ann. Pure & Appl. Logic*, 51(1/2):159–172, 1991.
23. E. Moggi. Notions of computation and monads. *Inf. & Comp.*, 93(1):55–92, 1991.
24. S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M. Srivas. PVS: Combining specification, proof checking, and model checking. In R. Alur and T.A. Henzinger, editors, *Computer Aided Verification*, number 1102 in Lect. Notes Comp. Sci., pages 411–414. Springer, Berlin, 1996.
25. S. Owre, J.M. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Trans. on Softw. Eng.*, 21(2):107–125, 1995.
26. Ch. Paulin-Mohring. Inductive definitions in the system Coq. Rules and properties. In M. Bezem and J.F. Groote, editors, *Typed Lambda Calculi and Applications*, number 664 in Lect. Notes Comp. Sci., pages 328–345. Springer, Berlin, 1993.
27. L.C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *Journ. of Logic and Computation*, 7:175–204, 1997.
28. A.M. Pitts. A co-induction principle for recursively defined domains. *Theor. Comp. Sci.*, 124(2):195–219, 1994.
29. J. Rutten and D. Turi. Initial algebra and final coalgebra semantics for concurrency. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *A Decade of Concurrency*, number 803 in Lect. Notes Comp. Sci., pages 530–582. Springer, Berlin, 1994.
30. J.J.M.M. Rutten. Universal coalgebra: a theory of systems. CWI Report CS-R9652, 1996.

This article was processed using the LaTeX macro package with LLNCS style