Foundational, Compositional (Co)datatypes for Higher-Order Logic
Category Theory Applied to Theorem Proving

Dmitriy Traytel Andrei Popescu Jasmin Christian Blanchette
Technische Universität München, Germany
{traytel,popescua,blanchette}@in.tum.de

***Abstract*—Higher-order logic (HOL) forms the basis of several popular interactive theorem provers. These follow the definitional approach, reducing high-level specifications to logical primitives. This also applies to the support for datatype definitions. However, the internal datatype construction used in HOL4, HOL Light, and Isabelle/HOL is fundamentally noncompositional, limiting its efficiency and flexibility, and it does not cater for codatatypes.**

We present a fully modular framework for constructing (co)datatypes in HOL, with support for mixed mutual and nested (co)recursion. Mixed (co)recursion enables type definitions involving both datatypes and codatatypes, such as the type of finitely branching trees of possibly infinite depth. Our framework draws heavily from category theory. The key notion is that of a *rich type constructor*—a functor satisfying specific properties preserved by interesting categorical operations. Our ideas are formalized in Isabelle and implemented as a new definitional package, answering a long-standing user request.

***Keywords*—Category theory, higher-order logic, interactive theorem proving, (co)datatypes, cardinals**

I. Introduction

Higher-order logic (HOL, Sect. II) forms the basis of several popular interactive theorem provers, notably HOL4 [9], HOL Light [15], and Isabelle/HOL [25]. Its straightforward semantics, which interprets types as sets (collections) of elements, makes it an attractive choice for many computer science and mathematical formalizations.

The theorem provers belonging to the HOL family traditionally encourage their users to adhere to the definitional approach, whereby new types and constants are defined in terms of existing constructs rather than introduced axiomatically. Following the LCF philosophy [10], theorems can be generated only within a small inference kernel, reducing the amount of code that must be trusted. As a result, HOL-based provers are widely considered trustworthy.

The definitional approach is a harsh taskmaster. At the primitive level, a new type is defined by carving out an isomorphic subset from an existing type. Higher-level mechanisms are also available, but behind the scenes they reduce the user-supplied specification to primitive type definitions.

The most important high-level mechanism is undoubtedly the datatype package, which automates the derivation of (freely generated inductive) datatypes. Melham [23] devised such a definitional package already two decades ago. His approach, considerably extended by Gunter [12], [13] and simplified by Harrison [14], now lies at the heart of the implementations in HOL4, HOL Light, and Isabelle/HOL.

Despite having withstood the test of time, the Melham–Gunter approach suffers from a few limitations that impair its usefulness. The most pressing issue is probably its ignorance of codatatypes (the coinductive pendant of datatypes). Lacking a definitional package to automate the definition of codatatypes, users face an unappealing choice between tedious manual constructions and risky axiomatizations [8].

Creating a monolithic codatatype package to supplement the datatype package is not an attractive prospect, because many applications need to mix and match datatypes and codatatypes, as in the following nested-(co)recursive specification of finitely branching trees of possibly infinite depth:

$$\textbf{datatype}\ \alpha\,\mathsf{list} = \mathsf{Nil} \mid \mathsf{Cons}\,\alpha\,(\alpha\,\mathsf{list})$$
$$\textbf{codatatype}\ \alpha\,\mathsf{tree_l} = \mathsf{Node}\,\alpha\,((\alpha\,\mathsf{tree_l})\,\mathsf{list})$$

Ideally, users should also be allowed to define (co)datatypes with (co)recursion through well-behaved non-free type constructors, such as the finite set constructor fset:

$$\textbf{codatatype}\ \alpha\,\mathsf{tree_l} = \mathsf{Node}\,\alpha\,((\alpha\,\mathsf{tree_l})\,\mathsf{fset})$$

This paper presents a fully compositional framework for defining datatypes and codatatypes in HOL, including mutual and nested (co)recursion through an arbitrary combination of datatypes, codatatypes, and other well-behaved type constructors (Sect. III). The underlying mathematical apparatus for specifying and structuring datatypes is taken from category theory. From this perspective, our type constructors are functors satisfying specific semantic properties, and we call them *rich type constructors* (*RTCs*). Unlike all previous approaches implemented in HOL-based provers, our framework imposes no syntactic restrictions on the type constructors that can participate in nested (co)recursion.

The main mathematical contribution of this paper is a novel class of functors—the RTCs—that is closed under the initial algebra, final coalgebra, and composition operations and that allows initial and final constructions in a sufficiently "local" way (Sect. IV). Cardinality reasoning with canonical membership-based well-orders lies beyond HOL's expressive power, so we need a theory of cardinals that circumvents this limitation. Performing global categorical constructions in a weak, "local" formalism arguably constitutes the logical equivalent of walking on a tightrope.

We have formalized the development in Isabelle/HOL and

Supported by the DFG project Ni 491/13–1 (part of the DFG priority program RS[3]).

are proceeding to implement a fully automatic definitional package for (co)datatypes based on these ideas to supplant the existing datatype package (Sect. V).

II. Higher-Order Logic (HOL)

By HOL we mean classical higher-order logic with Hilbert choice, the axiom of infinity, and ML-style polymorphism. HOL is based on Church's simple type theory [2], [7]. It is the logic of Gordon's original HOL system [9] and of its many successors and emulators. To keep the discussion focused on the relevant issues, we depart from tradition and present HOL not as a formal system but rather as a framework for expressing mathematics, much in the way that set theory is employed by working mathematicians.

A. Basics

The standard semantics of HOL relies on a universe $\mathscr{U}$ of *types*, ranged over by α, β, γ, which we view as nonempty collections of elements. Membership of an element a in a type α is written $a : \alpha$. The type unit consists of a single element written $()$, bool is the Boolean type, and nat is the type of natural numbers. Fixed elements of types, such as $() : \mathsf{unit}$, are called *constants*. Given α and β, we can form the type $\alpha \to \beta$ of (total) functions from α to β. If $f : \alpha \to \beta$ and $a : \alpha$, then $f\ a : \beta$ is the result of applying f to a. The types $\alpha + \beta$ and $\alpha \times \beta$ are the disjoint sum and the product of α and β, respectively. For functions taking n arguments, we generally prefer the curried form $f : \alpha_1 \to \cdots \to \alpha_n \to \beta$ to the tuple form $f : (\alpha_1 \times \cdots \times \alpha_n) \to \beta$.

HOL supports a restrictive, simply typed flavor of set theory. We write $\alpha\,\mathsf{set}$ for the powertype of α, consisting of sets of α elements; it is isomorphic to $\alpha \to \mathsf{bool}$. The *universe set of* α, $\mathsf{U}_\alpha : \alpha\,\mathsf{set}$, is the set consisting of all the elements of α. For notational convenience, we sometimes write α instead of U_α. Given an element $a : \alpha$ and a set $A : \alpha\,\mathsf{set}$, $a \in A$ tests whether a belongs to A. Although the two concepts are related, set membership is not to be confused with type membership. Given a type α and a predicate $\varphi : \alpha \to \mathsf{bool}$, we can form by comprehension the set $\{a{:}\alpha.\ \varphi\ a\}$ of type $\alpha\,\mathsf{set}$. Russell's paradox is avoided, because elements of $\alpha\,\mathsf{set}$ cannot be elements of α.

While unit, bool, and nat are types in their own right, set, $\to$, $+$, and $\times$ are *type constructors*, i.e., functions on the universe of types. The first of these is unary, and the last three are binary. Types are a special case of type constructors, with arity 0. We can introduce new type constructors by combining existing type constructors and comprehension; for example, we can define the ternary type constructor $(\alpha_1, \alpha_2, \alpha_3)\ \mathsf{F}$ as $(\alpha_2 + \alpha_1) \times (\alpha_3\ \mathsf{set})$. Except for infix operators, type constructor application is written in postfix notation (e.g., $\alpha\ \mathsf{F}$), whereas function application is written in prefix notation (e.g., $f\ a$).

Depending on the context, $(\alpha_1, \ldots, \alpha_n)\ \mathsf{F}$ either denotes the application of F to $(\alpha_1, \ldots, \alpha_n)$ or simply indicates that F is an n-ary type constructor. We abbreviate $(\alpha_1, \ldots, \alpha_n)\ \mathsf{F}$ to $\overline{\alpha}\ \mathsf{F}$. Given a binary type constructor $(\alpha_1, \alpha_2)\ \mathsf{F}$ and a fixed type β, $(\_, \beta)\ \mathsf{F}$ denotes the unary type constructor sending an arbitrary type α to $(\alpha, \beta)\ \mathsf{F}$, and similarly for $(\beta, \_)\ \mathsf{F}$.

As the main primitive way of introducing custom types, HOL lets us carve out from a type α the type corresponding to a nonempty set comprehension $A = \{a{:}\alpha.\ \varphi\ a\}$, yielding a type β and an injective function $f : \beta \to \alpha$ whose image is A.

Where Church's simple type theory only offers monomorphic types, HOL features ML-style (rank-1) polymorphism and type inference. Polymorphic constants can be regarded as families of constants indexed by types. For example, the identity function $\mathsf{id} : \alpha \to \alpha$ is defined for any type α and corresponds to a family $(\mathsf{id}_\alpha)_{\alpha \in \mathscr{U}}$. $\mathsf{Id} : (\alpha \times \alpha)\ \mathsf{set}$ is the identity relation. Function composition $\circ$ has type $(\alpha \to \beta) \to (\beta \to \gamma) \to \alpha \to \gamma$. Type arguments can be indicated by a subscript (e.g., U_α) if needed.

B. Expressiveness

HOL is significantly weaker than the set theories popular as foundations of mathematics, such as Zermelo–Fraenkel with the axiom of choice (ZFC). Some standard mathematical constructions cannot be performed in HOL, notably those dealing with proper classes or families of unboundedly large sets (not containable in any fixed set). A typical example is the representation of the HOL semantics, which is impossible in HOL due to the unbounded nature of the simple type hierarchy. Another example is the standard (membership-based) theory of ordinals and cardinals, which involves the well-ordered class of ordinals.

Nonetheless, many standard mathematical constructions are *local*, meaning that they are performed within an arbitrary but fixed universe set. These are particularly well suited to (polymorphic) HOL. Examples include basic algebra and analysis, formal language theory, and structural operational semantics. Indeed, a large body of mathematics can be expressed adequately in HOL, as witnessed by the extensive library developments in HOL-based provers.

III. Datatypes in HOL

The limitations of HOL mentioned above may seem exotic and contrived. Yet our application—datatype definitions—is precisely one of those areas where HOL's lack of expressiveness is most painfully felt. Category theory offers a powerful, modular methodology for constructing (co)datatypes, but filling the gap between theoretical category theory and theorem proving in HOL, with its simply typed set theory, is challenging; indeed, it is the main concern of this paper.

A. The Melham–Gunter Approach

Melham's original datatype package [23] is based on a manually defined polymorphic datatype of finite labeled trees, from which simple datatypes are carved out as subsets. Gunter [12] generalized the package to support mutually recursive datatypes. She also showed how to reduce specifications with nested recursion to mutually recursive specifications. A typical example is the recursive occurrence of $\alpha\ \mathsf{tree_F}$ nested in the list type constructor in the definition of finite trees. To define such a type, Gunter unfolds the definition of list, resulting in a mutually recursive definition of trees ($\alpha\ \mathsf{tree_F}$) and "lists-of-trees" ($\alpha\ \mathsf{tree_F\_list}$):

$$\begin{aligned}&\textbf{datatype}\ \alpha\ \mathsf{tree_F} = \mathsf{Node}\ \alpha\ (\alpha\ \mathsf{tree_F\_list})\\ &\textbf{and}\ \alpha\ \mathsf{tree_F\_list} = \mathsf{Nil} \mid \mathsf{Cons}\ (\alpha\ \mathsf{tree_F})\ (\alpha\ \mathsf{tree_F\_list})\end{aligned}$$

Exploiting an isomorphism, the package translates occurrences of $\alpha\ \mathsf{tree_F\_list}$ to $(\alpha\ \mathsf{tree_F})$ list, maintaining to a large extent the illusion of nested recursion. Orthogonally, Gunter [13] extended Melham's labeled trees with infinite branching, to support positive recursion through functions.

The handling of mutual and nested recursion has several disadvantages, all related to its nonmodularity. Most importantly, it is not clear how to extend the approach to nested recursion and corecursion or to non-free constructors. In addition, some of the internal aspects of the construction are visible to the user (e.g., in the type of the iterator used to define primitive recursive function). Finally, replaying recursive definitions and transferring results via isomorphisms is prohibitive slow for datatypes with many layers of nesting.

B. Bringing HOL Closer to Category Theory

Let $\alpha\ \mathsf{F}$ be a unary type constructor. Category theory has elegant devices to define, based on F, the associated datatype and codatatype by solving the equation $\alpha \cong \alpha\ \mathsf{F}$ (up to isomorphism) in a minimal and maximal way, obtaining the initial F-algebra and final F-coalgebra, respectively. However, this requires F to be complemented by an *action on functions between types*, usually called a "map."

The universe of types $\mathscr{U}$ naturally forms a category where the objects are types and the morphisms are functions between types. We are interested in type constructors $(\alpha_1, \ldots, \alpha_n)\ \mathsf{F}$ that are also *functors* on $\mathscr{U}$, i.e., that are equipped with an action on morphisms commuting with identities and composition. Taking advantage of polymorphism, this action can be expressed as a constant $\mathsf{Fmap} : (\alpha_1 \to \beta_1) \to \ldots \to (\alpha_n \to \beta_n) \to \overline{\alpha}\ \mathsf{F} \to \overline{\beta}\ \mathsf{F}$ satisfying

- $\mathsf{Fmap}\ \mathsf{id} = \mathsf{id}$;
- $\mathsf{Fmap}\ (g_1 \circ f_1)\ \ldots\ (g_n \circ f_n) = (\mathsf{Fmap}\ \overline{g}) \circ (\mathsf{Fmap}\ \overline{f})$.

Let us review some basic functors.

(n,α)-constant functor $(\mathsf{C}_{n,\alpha}, \mathsf{Cmap}_{n,\alpha})$*:* The (n,α)-constant functor $(\mathsf{C}_{n,\alpha}, \mathsf{Cmap}_{n,\alpha})$ is the n-ary functor consisting of the constant type constructor $(\beta_1, \ldots, \beta_n)\ \mathsf{C}_{n,\alpha} = \alpha$ and the constant map function $\mathsf{Cmap}_{n,\alpha}\ f_1\ \ldots\ f_n = \mathsf{id}$.

Sum functor $(+, \oplus)$*:* $\alpha_1 + \alpha_2$ consists of a copy $\mathsf{Inl}\ a_1$ of each element $a_1 : \alpha_1$ and a copy $\mathsf{Inr}\ a_2$ of each element $a_2 : \alpha_2$. Given $f_1 : \alpha_1 \to \beta$ and $f_2 : \alpha_2 \to \beta$, let $[f_1, f_2] : \alpha_1 + \alpha_2 \to \beta$ be the function sending $\mathsf{Inl}\ a_1$ to $f_1\ a_1$ and $\mathsf{Inr}\ a_2$ to $f_2\ a_2$. Given $f_1 : \alpha_1 \to \beta_1$ and $f_2 : \alpha_2 \to \beta_2$, let $f_1 \oplus f_2 : \alpha_1 + \alpha_2 \to \beta_1 + \beta_2$ be $[\mathsf{Inl} \circ f_1, \mathsf{Inr} \circ f_2]$.

Product functor $(\times, \otimes)$*:* Let $\mathsf{fst} : \alpha_1 \times \alpha_2 \to \alpha_1$ and $\mathsf{snd} : \alpha_1 \times \alpha_2 \to \alpha_2$ denote the two standard projection functions. Given $f_1 : \alpha \to \beta_1$ and $f_2 : \alpha \to \beta_2$, let $\langle f_1, f_2\rangle : \alpha \to \beta_1 \times \beta_2$ be the function $a \mapsto (f_1\ a, f_2\ a)$. Given $f_1 : \alpha_1 \to \beta_1$ and $f_2 : \alpha_2 \to \beta_2$, let $f_1 \otimes f_2 : \alpha_1 \times \alpha_2 \to \beta_1 \times \beta_2$ be $\langle f_1 \circ \mathsf{fst}, f_2 \circ \mathsf{snd}\rangle$.

α-Function space functor $(\mathsf{func}_\alpha, \mathsf{comp}_\alpha)$*:* Given a type α, let $\beta\ \mathsf{func}_\alpha = \alpha \to \beta$. For all $f : \beta_1 \to \beta_2$, we define $\mathsf{comp}_\alpha\ f : \beta_1\ \mathsf{func}_\alpha \to \beta_2\ \mathsf{func}_\alpha$ as $\mathsf{comp}_\alpha\ f\ g = f \circ g$.

Powertype functor $(\mathsf{set}, \mathsf{image})$*:* The function $\mathsf{image}\ f : \alpha\ \mathsf{set} \to \beta\ \mathsf{set}$ sends each set A to the image of A through the function $f : \alpha \to \beta$.

k-Powertype functor $(\mathsf{set}_k, \mathsf{image}_k)$*:* Given a cardinal k, for all types α, we define the type $\alpha\ \mathsf{set}_k$ by comprehension, carving out from $\alpha\ \mathsf{set}$ only those sets of cardinality $< k$.

While specific map functions are heavily used in HOL theories (e.g., map, image), the theorem provers traditionally do not record the functorial structure Fmap of F or take advantage of it when defining datatypes. The next examples illustrate the benefits of keeping such additional structure.

Finite lists: The unary type constructor list, which sends each type α to the type α list of lists of α elements, is categorically given as the initial algebra on the second argument of the binary functor (F, Fmap), where $(\alpha, \beta)\ \mathsf{F} = \mathsf{unit} + \alpha \times \beta$ and $\mathsf{Fmap}\ f\ g = \mathsf{id} \oplus f \otimes g$. More precisely, there exists a (polymorphic) *folding bijection* $\mathsf{fld} : (\alpha, \alpha\ \mathsf{list})\ \mathsf{F} \to \alpha\ \mathsf{list}$ making $(\mathsf{fld}, \alpha\ \mathsf{list})$ the initial algebra for the unary functor $(\alpha, \_)\ \mathsf{F}$. Here, $\mathsf{fld} = \langle \mathsf{Nil}, \mathsf{Cons}\rangle$, where Nil and Cons are the familiar list operations. The initial algebra property corresponds to the availability of the standard iterator for lists. Then (list, map) is itself a unary functor.

Finitely branching trees of finite depth: The ability to define lists is hardly a spectacular achievement. It is the *abstract interface* to lists that makes category theory relevant: (list, map) is simply another functor available for nesting in (co)datatype definitions. Assume we want to define finitely branching trees of finite depth. This involves taking the initial algebra $\alpha\ \mathsf{tree_F}$ on the second argument of the functor (G, Gmap), where $(\alpha, \beta)\ \mathsf{G} = \alpha \times \beta\ \mathsf{list}$ and $\mathsf{Gmap}\ f\ g = f \otimes \mathsf{map}\ g$. The resulting iterator iter has the polymorphic type $(\alpha \times \beta\ \mathsf{list} \to \beta) \to \alpha\ \mathsf{tree_F} \to \beta$, and its characteristic equation is $\mathsf{iter}\ s \circ \mathsf{fld} = s \circ (\mathsf{id} \otimes \mathsf{map}\ (\mathsf{iter}\ s))$, where fld is the folding bijection associated to $\alpha\ \mathsf{tree_F}$ (Fig. 1). Thus, the "contract" of tree iteration reads as follows: Given tree-like structure on β as the function $s : \alpha \times \beta\ \mathsf{list} \to \beta$ (viewing β as consisting of "abstract trees," featuring an abstract tree constructor s), provide a function $\mathsf{iter}\ s$ such that $\mathsf{iter}\ s\ (\mathsf{fld}\ (a, \mathit{trl})) = s\ (a, \mathsf{map}\ (\mathsf{iter}\ s)\ \mathit{trl})$ for all $a : \alpha$ and $\mathit{trl} : (\alpha\ \mathsf{tree_F})$ list. The characteristic equation of iter abstracts away completely from the definition of lists, using instead the map interface for accessing lists, thereby allowing truly modular nesting of recursive types inside recursive definitions of larger types. Moreover, the categorical approach gracefully handles nested recursion through corecursion, as the next examples illustrate.

$$\begin{tikzcd}
\alpha \times (\alpha\ \mathsf{tree_F})\ \mathsf{list} \arrow[r, "\mathsf{fld}"] \arrow[d, "\mathsf{id} \otimes \mathsf{map}\ (\mathsf{iter}\ s)"'] & \alpha\ \mathsf{tree_F} \arrow[d, "\mathsf{iter}\ s"] \\
\alpha \times \beta\ \mathsf{list} \arrow[r, "s"] & \beta
\end{tikzcd}$$

Fig. 1. Iterator for finitely branching trees of finite depth

$$\begin{tikzcd}
\beta \arrow[r, "s"] \arrow[d, "\mathsf{coiter}\ s"'] & \alpha \times \beta\ \mathsf{list} \arrow[d, "\mathsf{id} \otimes \mathsf{map}\ (\mathsf{coiter}\ s)"] \\
\alpha\ \mathsf{tree_I} \arrow[r, "\mathsf{unf}"] & \alpha \times (\alpha\ \mathsf{tree_I})\ \mathsf{list}
\end{tikzcd}$$

Fig. 2. Coiterator for finitely branching trees of possibly infinite depth

Finitely branching trees of possibly infinite depth: To define trees of possibly infinite depth, we can take the final coalgebra $\alpha\ \mathsf{tree_I}$ on the second argument of the functor $(\mathsf{G}, \mathsf{Gmap})$ defined above. The resulting coiterator coiter has polymorphic type $(\beta \to \alpha \times \beta\ \mathsf{list}) \to \beta \to \alpha\ \mathsf{tree_I}$, and its characteristic equation is $\mathsf{unf} \circ \mathsf{coiter}\ s \cong (\mathsf{id} \otimes \mathsf{map}\ (\mathsf{coiter}\ s)) \circ s$, where unf is the *unfolding bijection* associated to $\alpha\ \mathsf{tree_I}$ (Fig. 2). Normally, we would split unf in two functions as $\mathsf{unf} = \langle \mathsf{lab}, \mathsf{sub} \rangle$, where, for any $tr : \alpha\ \mathsf{tree_I}$, $\mathsf{lab}\ tr : \alpha$ is the label of the root and $\mathsf{sub}\ tr$ is the list of its subtrees. Then, also splitting any $s : \beta \to \alpha \times \beta\ \mathsf{list}$ similarly to unf in two functions L and C, the contract of tree coiteration reads as follows: Given a tree-like structure on β consisting of functions $L : \beta \to \alpha$ and $C : \beta \to \beta\ \mathsf{list}$, yield a function $\mathsf{coiter}_{\langle C,L \rangle}$ such that $\mathsf{lab}\ (\mathsf{coiter}_{\langle C,L \rangle}\ b) = L\ b$ and $\mathsf{sub}\ (\mathsf{coiter}_{\langle C,L \rangle}\ b) = \mathsf{map}\ \mathsf{coiter}_{\langle C,L \rangle}\ (C\ b)$ for all $b : \beta$.

Unordered finitely branching trees of possibly infinite depth: Assume that we want our finitely branching trees to be unordered. Instead of lists, we can employ finite sets (or even finite multisets). We can then define $\alpha\ \mathsf{tree_I}$ as the final coalgebra of the functor $(\mathsf{H}, \mathsf{Hmap})$, where $(\alpha, \beta)\ \mathsf{H} = \alpha \times \beta\ \mathsf{fset}$ and $\mathsf{Hmap}\ f\ g = f \otimes \mathsf{fimage}\ g$.

C. Bringing Category Theory Closer to HOL

Next we focus on devising a proper categorical setting to accommodate (co)datatype definitions. Here is the system of constraints for our desired class $\mathscr{K}$ of functors (perhaps with additional structure) on the universe of types:

- C1 $\mathscr{K}$ contains basic functors, including at least the constant, sum, product, and function-space functors.
- C2 All functors in $\mathscr{K}$ *admit* both (a) initial algebras and (b) final coalgebras.
- C3 Class $\mathscr{K}$ is *closed under* (a) initial algebras; (b) final coalgebras; and (c) composition.
- C4 The initial algebra and final coalgebra operations over $\mathscr{K}$ are *expressible* in HOL.

In addition to the above nonnegotiable requirements, we formulate a desideratum:

- D $\mathscr{K}$ contains interesting non-free functors, such as the bounded sets and multisets.

Among the basic functors mentioned in C1, constants, $+$, and $\times$ are needed for constructing even simple datatypes, whereas func_α enables infinite branching. The non-free functors mentioned in D further extend (co)datatypes with permutative structures, among which finite sets and multisets are especially useful in computer science formalizations (e.g., semantics of programming languages).

In C3, closure under initial algebras means the following, say, for binary functors $((\alpha, \beta)\ \mathsf{F}, \mathsf{Fmap})$. If we fix an argument, say, the first, then, by C2, for each fixed type α, there exists the initial F-algebra on the second argument, $\alpha\ \mathsf{IF}$, for which we can define a map operator IFmap. C3 requires that the unary functor $(\mathsf{IF}, \mathsf{IFmap})$ be in $\mathscr{K}$. And similarly for closure under final coalgebras.

C4 is required because we are committed to a definitional framework. Otherwise, we could simply postulate the types corresponding to initial and final coalgebras, together with the necessary (co)iterators and their properties.

The literature does not appear to provide a complete solution for the above system of constraints. An obvious candidate, the class of ω-bicontinuous functors [22], satisfies C1–C3 but not C4, because the associated limit construction requires a logic that can express infinite type families (e.g., $(\mathsf{unit}\ \mathsf{F}^n)_n$ for the final coalgebra).

Many results from the literature are concerned only with a given type of construction, and only with admissibility (C2), ignoring closure (C3). Rutten's monograph [31] focuses on coalgebras. It describes a general class of functors on sets, namely, those that preserve weak pullbacks and have a set of generators, or, sufficiently, preserve weak pullbacks and are bounded (in that there exists a cardinal upper bound for the coalgebras generated by any singleton in any of their coalgebras). The main issue with this class of functors is admissibility of initial algebras (C2-a). Closure properties (C3), which Rutten omits to discuss, might also be an issue.

Also focusing on coalgebra, Barr [4], [5] proves the existence of a final coalgebra for accessible functors on sets (i.e., functors preserving k-filtered colimits for some k). This result is an internalization to sets of Aczel and Mendler's final coalgebra theorem [1] stated for set-based functors on classes. Moreover, Barr produces a bound for the size of the final coalgebra, assuming the existence of a certain large cardinal. However, k-filtered colimits are incompatible with C4 for the same reason ω-limit constructions do, and internalizing the construction to a sufficiently large type using the provided cardinal bound is also infeasible, because it requires large cardinals whose existence is not provable in HOL or even ZFC. (C2-a and C3 might also be problematic.)

A different result from Barr [4] states that any quotient functor of an ω-bicontinuous functor admits a weakly final coalgebra obtained from any weakly final coalgebra of the latter. A subclass of ω-bicontinuous that admits HOL-

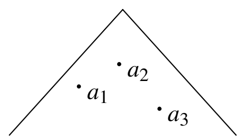

Fig. 3. An element x of α F with $\mathsf{Fset}\ x = \{a_1, a_2, a_3\}$

expressible (co)datatype constructions could prove to be an answer to C1–C4 via this result. In fact, the class $\mathscr{K}$ we adopt in this paper includes the class $\mathscr{K}'$ of functors F that are quotients of Fbd-function-space functors, with Fbd a cardinal number depending on F. Whether $\mathscr{K}'$ is also a solution to C1–C4 remains for us an open question.

Finally, Hensel and Jacobs [16] propose a modular development of (co)datatypes for datafunctors, namely, functors obtained from constants, $+$, and $\times$ by repeated application of composition, initial algebra, and final coalgebra. Datafunctors satisfy C1–C3 but ostensibly not C4, because the arguments, which employ abstract results on categorical logic and fibrations [17], rely on (co)limits.

IV. Rich Type Constructors

To accommodate constraints C1–C4 in HOL, we must work in a strict cardinal-bounded fashion, always keeping in sight a universe type able to host the necessary construction. However, to stay flexible and not commit to a syntactically predetermined class of functors, we cannot a priori fix a universe type, as required by the Melham–Gunter approach. For example, there is no type that can accommodate an arbitrary iteration of the countable powertype construction. Consequently, our functors will carry their cardinal bounds with themselves.

A useful means to keep cardinality under control is the consideration of a natural "atom" structure potentially available for the HOL type constructors in addition to the map structure. Namely (assuming F is unary), we consider a polymorphic constant $\mathsf{Fset} : \alpha\ \mathsf{F} \to \alpha\ \mathsf{set}$, where $\mathsf{Fset}\ x$ consists of all "atoms" of x; for example, if F is list, Fset returns the set of elements in the list.

We think of the elements x of α F as consisting of a *shape* together with a *content* that fills the shape with elements of α, with $\mathsf{Fset}\ x$ returning this content in flattened format, as a set (Fig. 3). This suggests that Fset should be a natural transformation between the functors $(\mathsf{F}, \mathsf{Fmap})$ and $(\mathsf{set}, \mathsf{image})$ (diagram in Fig. 4 commutative for all $f : \alpha \to \beta$). Fset allows us to internalize the type constructor F to sets of elements of given types α. Namely, we define $\mathsf{Fin} : \alpha\ \mathsf{set} \to (\alpha\ \mathsf{F})\ \mathsf{set}$ by $\mathsf{Fin}\ A = \{x : \alpha\ \mathsf{F}.\ \mathsf{Fset}\ x \subseteq A\}$. The generalization to n-ary functors is straightforward, with $\mathsf{Fin}\ A_1\ \ldots\ A_n = \{x : (\alpha_1, \ldots, \alpha_n)\ \mathsf{F}.\ \bigwedge_i \mathsf{Fset}_i\ x \subseteq A_i\}$. In particular, $\mathsf{Fin}\ \alpha_1\ A_2 = \{x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \mathsf{Fset}_2\ x \subseteq A_2\}$ (where the first occurrence of α_1 abbreviates U_{α_1}).

Combining the map and set operators and suitable cardinal bounds, we obtain the following key notion, presented here

$$\begin{tikzcd} \alpha\ \mathsf{F} \arrow[r, "\mathsf{Fset}"] \arrow[d, "\mathsf{Fmap}\ f"'] & \alpha\ \mathsf{set} \arrow[d, "\mathsf{image}\ f"] \\ \beta\ \mathsf{F} \arrow[r, "\mathsf{Fset}"] & \beta\ \mathsf{set} \end{tikzcd}$$

Fig. 4. The "set" natural transformation

for the binary case. A binary *rich type constructor* (*RTC*) is a tuple $(\mathsf{F}, \mathsf{Fmap}, \mathsf{Fset}, \mathsf{Fbd})$, where

- F is a binary type constructor,
- $\mathsf{Fmap} : (\alpha_1 \to \beta_1) \to (\alpha_2 \to \beta_2) \to (\alpha_1, \alpha_2)\ \mathsf{F} \to (\beta_1, \beta_2)\ \mathsf{F}$,
- $\mathsf{Fset}_i : (\alpha_1, \alpha_2)\ \mathsf{F} \to \alpha_i\ \mathsf{set}$ for $i \in \{1, 2\}$,
- Fbd is an infinite cardinal number,

satisfying the following properties:

- FUNC $(\mathsf{F}, \mathsf{Fmap})$ is a binary functor.
- NAT$_1$ For all α_1, Fset_1 is a natural transformation between $((\alpha_1, \_)\ \mathsf{F}, \mathsf{Fmap})$ and $(\mathsf{set}, \mathsf{image})$.
- NAT$_2$ For all α_2, Fset_2 is a natural transformation between $((\_, \alpha_2)\ \mathsf{F}, \mathsf{Fmap})$ and $(\mathsf{set}, \mathsf{image})$.
- WP $(\mathsf{F}, \mathsf{Fmap})$ preserves weak pullbacks.
- CONG If $\forall a \in \mathsf{Fset}_i\ x.\ f_i\ a = g_i\ a$ for all $i \in \{1, 2\}$, then $\mathsf{Fmap}\ f_1\ f_2\ x = \mathsf{Fmap}\ g_1\ g_2\ x$.
- CBD The following cardinal-bound conditions hold:
  a. $\forall x : (\alpha_1, \alpha_2)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \leq \mathsf{Fbd}$ for $i \in \{1,2\}$;
  b. $|\mathsf{Fin}\ A_1\ A_2| \leq (|A_1| + |A_2| + 2)^{\mathsf{Fbd}}$.

Binary functors suffice to illustrate the functorial structure of the initial and final algebras, a structure that would be trivial if we started with unary functors. (The definition of n-ary RTCs is given in the appendix.)

Among the above conditions, FUNC and NAT$_i$ were already explained and motivated. WP is a technical condition allowing a smooth treatment of bisimilarity relations, relevant for coinduction and corecursion [31]; unlike other (weak) limits, weak pullbacks involve a finite number of types and are hence expressible in HOL. CONG states that $\mathsf{Fmap}\ f_1\ f_2\ x$ is uniquely determined by the action of f_i on the atoms of x, $\mathsf{Fset}_i\ x$—it ensures that Fmap behaves well with respect to Fin. Finally, the cardinality conditions put bounds on the branching (CBD-a) and on the number of elements (CBD-b) of the functor $(\mathsf{F}, \mathsf{Fmap})$, and can be understood in terms of shape and content. Thus, CBD-a states that the F-shapes have no more than Fbd slots for contents. Moreover, CBD-b states that shapes are not too redundant, so that all possible combinations of shape and content do not exceed the number of assignments of contents to slots, $A_1 + A_2 \to \mathsf{Fbd}$. (The $+2$ addition is a technicality that covers the case where $A_1 = A_2 = \emptyset$). We are now ready to state the main theoretical result of this paper:

Theorem 1: The class of RTCs satisfies constraints C1–C4 and desideratum D.

Proof sketch: We must show that certain basic type constructors form RTCs and that the operations of composition, initial algebra and final coalgebra exist in HOL and have themselves an RTC structure. Sects. A–F below are dedicated to these tasks. ■

A. Basic Type Constructors

Sect. III-B described the basic constructors' map structure. We now present their set structure and cardinal bound, guided by our "shape and content" intuition.

- $\mathsf{F} = \mathsf{C}_{n,\alpha}$: $\mathsf{Fset}\ x = \emptyset$; $\mathsf{Fbd} = \aleph_0$.
- $\mathsf{F} = +$: $\mathsf{Fset}_1\ (\mathsf{Inl}\ a_1) = \{a_1\}$, $\mathsf{Fset}_2\ (\mathsf{Inl}\ a_1) = \emptyset$, $\mathsf{Fset}_1\ (\mathsf{Inr}\ a_2) = \emptyset$, $\mathsf{Fset}_2\ (\mathsf{Inr}\ a_2) = \{a_2\}$; $\mathsf{Fbd} = \aleph_0$.
- $\mathsf{F} = \times$: $\mathsf{Fset}_1\ (a_1, a_2) = \{a_1\}$, $\mathsf{Fset}_2\ (a_1, a_2) = \{a_2\}$; $\mathsf{Fbd} = \aleph_0$.
- $\mathsf{F} = \mathsf{func}_\alpha$: $\mathsf{Fset}_1\ g = \mathsf{image}\ g\ \alpha$; $\mathsf{Fbd} = \mathsf{max}\ (|\alpha|\ ,\ \aleph_0)$.
- $\mathsf{F} = \mathsf{set}$: $\mathsf{Fset}\ x = x$; set is not an RTC though, due to the absence of a proper bound.
- $\mathsf{F} = \mathsf{set}_k$: $\mathsf{Fset}\ x$ is the set corresponding to x via the embedding of $\alpha\ \mathsf{set}_k$ into $\alpha\ \mathsf{set}$; $\mathsf{Fbd} = \mathsf{max}\ (k,\ \aleph_0)$.

B. Composition

For composition, we focus on the binary–unary case (without loss of generality). Given unary RTCs $\mathscr{F}_i = (\mathsf{F}^i, \mathsf{Fmap}^i, \mathsf{Fset}^i, \mathsf{Fbd}^i)$ with $i \in \{1, 2\}$ and a binary RTC $\mathscr{G} = (\mathsf{G}, \mathsf{Gmap}, \mathsf{Gset}, \mathsf{Gbd})$, their composition is the unary RTC $\mathscr{H} = \mathscr{G} \circ (\mathscr{F}_1, \mathscr{F}_2)$ defined as follows:

- $(\mathsf{H}, \mathsf{Hmap})$ is the functorial composition of $(\mathsf{G}, \mathsf{Gmap})$ with $(\mathsf{F}^i, \mathsf{Fmap}^i)$;
- $\mathsf{Hset}\ y = \bigcup_{x \in \mathsf{Gset}_1\ y} \mathsf{Fset}^1\ x \cup \bigcup_{x \in \mathsf{Gset}_2\ y} \mathsf{Fset}^2\ x$;
- $\mathsf{Hbd} = \mathsf{Gbd} * (\mathsf{Fbd}^1 + \mathsf{Fbd}^2)$.

Although we seldom emphasize its role, composition is a pervasive auxiliary operation in interesting (co)datatype definitions. For example, the list-defining RTC $(\alpha, \beta)\ \mathsf{F}$ discussed in Sect. III-B is a composition of basic RTCs.

C. Relators

A key insight due to Rutten [30] is that, thanks to WP, the functor $(\mathsf{F}, \mathsf{Fmap})$ has a natural extension to a *relator*, i.e., a functor on the category of types and binary relations, denoted $\mathscr{R}$. We can express the relator action of F as a polymorphic constant $\mathsf{Frel} : (\alpha_1 \times \alpha_2)\ \mathsf{set} \to (\beta_1 \times \beta_2)\ \mathsf{set} \to ((\alpha_1, \alpha_2)\ \mathsf{F} \times (\beta_1, \beta_2)\ \mathsf{F})\ \mathsf{set}$ defined as $\mathsf{Frel}\ Q\ R = \{(\mathsf{Fmap}\ \mathsf{fst}\ \mathsf{fst}\ z, \mathsf{Fmap}\ \mathsf{snd}\ \mathsf{snd}\ z).\ z \in \mathsf{Fin}\ Q\ R\}$.

For reasoning in HOL, it is convenient to take an alternative (equivalent) view of Frel, as an action on curried binary predicates $\mathsf{Fpred} : (\alpha_1 \to \alpha_2 \to \mathsf{bool}) \to (\beta_1 \to \beta_2 \to \mathsf{bool}) \to (\alpha_1, \alpha_2)\ \mathsf{F} \to (\beta_1, \beta_2)\ \mathsf{F} \to \mathsf{bool}$. $\mathsf{Fpred}\ \varphi\ \psi$ should be regarded as the *componentwise extension* of the predicates φ and ψ. For example:

- if F is the product functor, $\mathsf{Fpred}\ \varphi_1\ \varphi_2\ (a_1, a_2)\ (b_1, b_2) \Leftrightarrow \varphi_1\ a_1\ b_1 \wedge \varphi_2\ a_2\ b_2$;
- if F is the sum functor, $\mathsf{Fpred}\ \varphi_1\ \varphi_2\ a\ b \Leftrightarrow (\exists a_1\ b_1.\ a = \mathsf{Inl}\ a_1 \wedge b = \mathsf{Inl}\ b_1 \wedge \varphi_1\ a_1\ b_1) \vee (\exists a_2\ b_2.\ a = \mathsf{Inr}\ a_2 \wedge b = \mathsf{Inr}\ b_2 \wedge \varphi_2\ a_2\ b_2)$.

$$
\begin{tikzcd}
\mathsf{Fin}\,\beta\,A_1 \arrow[r, "s_1"] \arrow[d, "\mathsf{Fmap}\ \mathsf{id}\ f"'] & A_1 \arrow[d, "f"] \\
\mathsf{Fin}\,\beta\,A_2 \arrow[r, "s_2"] & A_2
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{Fin}\,\beta\,A_1 \arrow[d, "\mathsf{Fmap}\ \mathsf{id}\ f"'] & A_1 \arrow[l, "s_1"'] \arrow[d, "f"] \\
\mathsf{Fin}\,\beta\,A_2 & A_2 \arrow[l, "s_2"']
\end{tikzcd}
$$

Fig. 5. Algebra morphism (left) and coalgebra morphism (right)

D. The Categories of (Co)algebras

For this and the next two subsections, we fix a binary RTC $\mathscr{F} = (\mathsf{F}, \mathsf{Fmap}, \mathsf{Fset}, \mathsf{Fbd})$. We first show how to construct in HOL the initial algebra (or, dually, the final coalgebra) on the second argument—that is, the minimal solution $\alpha\ \mathsf{IF}$ (or maximal solution $\alpha\ \mathsf{JF}$) of the equation $\alpha \cong (\beta, \alpha)\ \mathsf{F}$. (The general constructions involve n $(m+n)$-ary RTCs $\mathscr{F}_i$ with type constructors $(\overline{\beta}, \overline{\alpha})\ \mathsf{F}_i$ and yield n m-ary RTCs $\mathscr{IF}_1, \ldots, \mathscr{IF}_n$ (or $\mathscr{JF}_1, \ldots, \mathscr{JF}_n$) with their type constructors of the form $\overline{\beta}\ \mathsf{IF}_i$ (or $\overline{\beta}\ \mathsf{JF}_i$).)

Abstractly, the theories of algebras and of coalgebras are dual, allowing a unified treatment of the basic (co)algebraic concepts. However, since the category of types is not self-dual, concrete constructions are often specific to each.

We fix a type β. A (β-)*algebra* is a pair $\mathscr{A} = (A, s)$ where:

- $A : \alpha\ \mathsf{set}$ is the *carrier set* of $\mathscr{A}$ (and α is the *underlying type* of $\mathscr{A}$),
- $s : (\beta, \alpha)\ \mathsf{F} \to \alpha$ is the *structural function* of $\mathscr{A}$,

such that A is *closed under* s, in that $\forall x \in \mathsf{Fin}\,\beta\,A.\ s\ x \in A$ (and thus we may regard s as a function $s : \mathsf{Fin}\,\beta\,A \to A$). Dually, a ($\beta$-)*coalgebra* is given by a pair $(A : \alpha\ \mathsf{set}, s : \alpha \to (\beta, \alpha)\ \mathsf{F})$ such that $\forall x \in A.\ s\ x \in \mathsf{Fin}\,\beta\,A$. Algebras form a category where morphisms $f : \mathscr{A}_1 = (\alpha_1, A_1, s_1) \to \mathscr{A}_2 = (\alpha_2, A_2, s_2)$ are functions $f : \alpha_1 \to \alpha_2$ such that the diagram on the left of Fig. 5 is commutative, and dually for coalgebras and the diagram on the right.

In the category of algebras, one can form *products* of families of algebras having the same underlying type, the carrier set of the product being the product of the carrier sets of the components. Dually, one can form *sums* of families of coalgebras using sums of sets. An algebra $\mathscr{A}$ is called *initial* if for all algebras $\mathscr{A}'$ there exists a unique morphism $f : \mathscr{A} \to \mathscr{A}'$, and *weakly initial* if we omit the uniqueness requirement. Dually, a coalgebra is *final* if it admits a unique morphism from any other coalgebra, and *weakly final* if uniqueness is dropped.

We are looking for a type constructor $\beta\ \mathsf{IF}$ (dually, $\beta\ \mathsf{JF}$) and function $\mathsf{fld} : (\beta, \beta\ \mathsf{IF}) \to \beta\ \mathsf{IF}$ (dually, $\mathsf{unf} : \beta\ \mathsf{JF} \to (\beta, \beta\ \mathsf{JF})$) such that the algebra $(\beta\ \mathsf{IF}, \mathsf{fld})$ is initial (dually, the coalgebra $(\beta\ \mathsf{JF}, \mathsf{unf})$ is final).

Typically, such a (co)algebra is obtained in two phases:

1. Construction of a weakly initial algebra (weakly final coalgebra) $\mathscr{C}$.
2. Construction of an initial algebra (final coalgebra) as a subalgebra (quotient coalgebra) of $\mathscr{C}$.

In the next two subsections, we discuss the key aspects of these constructions in HOL, both times starting with the simpler phase 2.

E. Initial Algebra

Initial algebra from weakly initial algebra: Given an algebra $\mathscr{A} = (A, s)$, let M_s be the intersection of all sets B such that (B, s) is an algebra, and let $\mathscr{M}(\mathscr{A})$, the *minimal subalgebra* of $\mathscr{A}$, be (M_s, s). It is immediate that there exists at most one morphism from $\mathscr{M}(\mathscr{A})$ to any other algebra. Then, given a weakly initial algebra $\mathscr{C}$, the desired initial β-algebra is its minimal subalgebra, $\mathscr{M}(\mathscr{C})$. Of course, $\mathscr{M}(\mathscr{C})$ depends on β (which was fixed all along). Now $\beta\ \mathsf{IF}$ is introduced by a type definition, carving out the underlying set of $\mathscr{M}(\mathscr{C})$ as a new type, and the folding map fld is defined by copying on $\beta\ \mathsf{IF}$ the structural map of $\mathscr{M}(\mathscr{C})$ (so that in effect $(\beta\ \mathsf{IF}, \mathsf{fld})$ becomes isomorphic to $\mathscr{M}(\mathscr{C})$).

Construction of a weakly initial algebra: This relies on a crucial lemma about the cardinality of minimal subalgebras, whose proof (given in the appendix) employs the RTC cardinality assumptions CBD.

Lemma 2: Let $s : (\beta, \alpha)\ \mathsf{F} \to \alpha$. Then $|M_s| \leq (|\beta| + 2)^{\mathsf{Suc\ Fbd}}$ (where $\mathsf{Suc\ Fbd}$ is the successor cardinal of Fbd).

Let Θ be the set of all algebras $\mathscr{A}$ having as underlying type a type γ of sufficiently large cardinality, $(|\beta| + 2)^{\mathsf{Suc\ Fbd}}$; such a type exists, and in fact can be taken to be the very underlying type of this cardinal. The desired weakly initial algebra $\mathscr{C}$ is the product of all algebras in Θ. Indeed, by Lemma 2, for any algebra $\mathscr{B}$, its minimal subalgebra $\mathscr{M}(\mathscr{B})$ is isomorphic to one in Θ, to which $\mathscr{C}$ has a projection morphism. This gives a morphism from $\mathscr{C}$ to $\mathscr{M}(\mathscr{B})$, hence also one from $\mathscr{C}$ to $\mathscr{B}$. We have thus proved:

Prop. 3: $(\beta\ \mathsf{IF}, \mathsf{fld})$ is the initial β-algebra.

This yields an iterator $\mathsf{iter} : ((\beta, \alpha)\ \mathsf{F} \to \alpha) \to \beta\ \mathsf{IF} \to \alpha$ such that $\mathsf{iter}\ s \circ \mathsf{fld} = s \circ \mathsf{Fmap\ id}\ (\mathsf{iter}\ s)$ (cf. Fig. 1).

Structural induction: The set structure Fset of an RTC not only plays an auxiliary role in the datatype constructions but also provides a simple means to *express induction abstractly, for arbitrary functors*. Since fld is a bijection, for any element $b \in \beta\ \mathsf{IF}$ there is a unique $y \in (\beta, \beta\ \mathsf{IF})\ \mathsf{F}$ such that $b = \mathsf{unf}\ y$—this is an abstract version of case analysis. Then the inductive components of b are precisely the elements of $\mathsf{Fset}_2\ y$, and we have the following induction principle:

Prop. 4: Let $\varphi : \beta\ \mathsf{IF} \to \mathsf{bool}$ and assume $\forall y.\ (\forall b \in \mathsf{Fset}_2\ y.\ \varphi\ b) \Rightarrow \varphi\ (\mathsf{fld}\ y)$. Then $\forall b.\ \varphi\ b$.

For $\mathsf{F} = \mathsf{unit} + \beta \times \alpha$ with $\mathsf{IF} = \mathsf{list}$ (Sect. III-B), the above is equivalent to the familiar induction principle.

RTC structure: It is standard to define a functorial structure for the initial algebra, namely $\mathsf{IFmap}\ f = \mathsf{iter}\ (\mathsf{fld} \circ (\mathsf{Fmap}\ f\ \mathsf{id}))$. As for the set structure, consider $b \in \beta\ \mathsf{IF}$. Intuitively, $\mathsf{IFset}\ b$ should contain all the Fset_1 atoms of b, then the Fset_1 atoms of its inductive components, and so on, iteratively. Moreover, as we have seen, delving into the inductive components is achieved by means of Fset_2. We are led to defining IFset as $\mathsf{iter\ collect}$, i.e., as the unique function making the Fig. 6 diagram commutative, where $\mathsf{collect}\ a = \mathsf{Fset}_1\ a \cup \bigcup \mathsf{Fset}_2\ a$.

$$\begin{tikzcd}
(\beta, \beta\ \mathsf{IF})\ \mathsf{F} \arrow[r, "\mathsf{fld}"] \arrow[d, "\mathsf{Fmap\ id\ IFset}"'] & \beta\ \mathsf{IF} \arrow[d, "\mathsf{IFset}"] \\
(\beta, \beta\ \mathsf{set})\ \mathsf{F} \arrow[r, "\mathsf{collect}"] & \beta\ \mathsf{set}
\end{tikzcd}$$

Fig. 6. Set structure for IF

Prop. 5: $(\mathsf{IF}, \mathsf{IFmap}, \mathsf{IFset}, 2^{\mathsf{Fbd}})$ is an RTC.

As an RTC, IF is also a relator (Sect. C). Importantly for modular reasoning however, we can express IFpred directly in terms of Fpred. Thus, IFpred is uniquely determined by the recursive equations $\mathsf{IFpred}\ \varphi\ (\mathsf{fld}\ x_1)\ (\mathsf{fld}\ x_2) \Leftrightarrow \mathsf{Fpred}\ \varphi\ (\mathsf{IFpred}\ \varphi)\ x_1\ x_2$. For example, for the list functor, the above equation splits in the following, according to the relator structure of the component functors (unit, $+$, and $\times$):

- $\mathsf{list\_pred}\ \varphi\ \mathsf{Nil\ Nil} \Leftrightarrow \mathsf{True}$,
- $\mathsf{list\_pred}\ \varphi\ \mathsf{Nil}\ (\mathsf{Cons}\ b\ bs) \Leftrightarrow \mathsf{False}$,
- $\mathsf{list\_pred}\ \varphi\ (\mathsf{Cons}\ a\ as)\ \mathsf{Nil} \Leftrightarrow \mathsf{False}$,
- $\mathsf{list\_pred}\ \varphi\ (\mathsf{Cons}\ a\ as)\ (\mathsf{Cons}\ b\ bs) \Leftrightarrow$ $\varphi\ a\ b \wedge \mathsf{list\_pred}\ \varphi\ as\ bs$,

revealing list_pred as the componentwise ordering on lists.

F. Final Coalgebra

Final coalgebra from weakly final coalgebra: This follows by the standard coalgebraic theory of bisimulation relations [31]. A bisimulation on a coalgebra $\mathscr{A} = (A, s)$ is a relation $R \subseteq A \times A$ such that $\forall (a, b) \in R.\ \exists z \in \mathsf{Fin}\ \beta\ R.\ \mathsf{Fmap\ id\ fst}\ z = a \wedge \mathsf{Fmap\ id\ fst}\ z = b$, i.e., such that in Fig. 7 (left) there exists a function along the dotted arrow making the two diagrams commutative. This abstract concept covers the natural ad hoc notions of bisimulation for concrete functors [31]. A bisimulation R is in effect an endomorphism on A in the types-and-relations category $\mathscr{R}$ such that $(a, b) \in R$ implies $(s\ a, s\ b) \in \mathsf{Frel\ Id}\ R$—Fig. 7 (right). Hence composition of bisimulations is a bisimulation, and then it follows easily that the largest bisimulation $\mathsf{LB}(\mathscr{A})$ on a coalgebra $\mathscr{A}$ is an equivalence relation, and that the resulting quotient coalgebra $\mathscr{A}/_{\mathsf{LB}(\mathscr{A})}$ has the property that any coalgebra has at most one morphism to it.

Now let $\mathscr{C}$ be a weakly final coalgebra. By the above discussion, via an argument dual to the corresponding one for algebras, we have $\mathscr{C}/_{\mathsf{LB}(\mathscr{C})}$ final and based on it we define the desired type $\beta\ \mathsf{JF}$ and its unfolding bijection unf.

Construction of a weakly final coalgebra: The abstract construction indicated in Rutten [31], as the sum of all coalgebras over a sufficiently large type (roughly dual to our weakly initial algebra construction), is possible in HOL thanks to our cardinality provisos. However, a more concrete

$$
\begin{tikzcd}
A \arrow[d, "s"'] & R \arrow[l, "\mathsf{fst}"'] \arrow[r, "\mathsf{snd}"] \arrow[d, dashed] & A \arrow[d, "s"] \\
\mathsf{Fin}\,\beta\,A & \mathsf{Fin}\,\beta\,R \arrow[l, "\mathsf{Fmap\ id\ fst}"'] \arrow[r, "\mathsf{Fmap\ id\ snd}"] & \mathsf{Fin}\,\beta\,A
\end{tikzcd}
\qquad
\begin{tikzcd}
A \arrow[r, leftrightarrow, "R"] \arrow[d, "s"'] & A \arrow[d, "s"] \\
\mathsf{Fin}\,\beta\,A \arrow[r, leftrightarrow, "\mathsf{Frel\ Id}\ R"] & \mathsf{Fin}\,\beta\,A
\end{tikzcd}
$$

Fig. 7. Bisimulation

construction gives us a better grip on cardinality, allowing us to check the RTC properties for the resulting coalgebra.

To lighten the presentation, we next identify sets with types—for example, we allow ourselves to apply type constructors such as list to sets. Given a prefix-closed subset Kl of Fbd list and $kl \in Kl$, we let $\mathsf{Suc}_{Kl,kl}$, the set of *Kl-successors of kl*, be $\{kl \mathbin{@} [k].\ kl \mathbin{@} [k] \in Kl\}$, where @ denotes list concatenation and $[k]$ the k-singleton list. We define an Fbd-*tree* to be a pair (Kl, tr), where $Kl \subseteq \mathsf{Fbd}$ list is prefix closed and $tr : Kl \to \mathsf{Fin}\,\beta\,\mathsf{Fbd}$ is such that $\forall kl \in Kl.\ \mathsf{Fset}_2\ (tr\ kl) = \mathsf{Suc}_{Kl,kl}$. Thus, Fbd-trees are at most Fbd-branching trees labeled as follows: Every node is labeled with an element of $\mathsf{Fin}\,\beta\,\mathsf{Fbd}$ whose set of second-argument atoms consists of precisely the node's emerging branches. Given a tree (Kl, tr), we define $\mathsf{sub}_{(Kl,tr)} : \{k.\ [k] \in Kl\} \to C$ to send each k to the immediate k-subtree of (Kl, tr), more precisely, $\mathsf{sub}_{(Kl,tr)}\ k = (Kl', tr')$, where $Kl' = \{kl'.\ [k] \mathbin{@} kl' \in Kl\}$ and $tr' : Kl' \to \mathsf{Fin}\,\beta\,\mathsf{Fbd}$ is defined by $tr'\ kl' = tr\ ([k] \mathbin{@} kl')$.

The set C of Fbd-trees can be naturally organized as a coalgebra $\mathscr{C} = (C, s)$ defining $s\ (Kl, tr) = \mathsf{Fmap\ id\ sub}_{(Kl,tr)}$ $(tr\ \mathsf{Nil})$. Thus, $s\ (Kl, tr)$ operates on (Kl, tr)'s root label $tr\ \mathsf{Nil}$, substituting in its shape the immediate subtrees for the contents. Then $\mathscr{C}$ is shown to be a weakly final coalgebra by roughly the following argument. For each element a in an algebra (A, t), one defines its behavior tree by iterating the unfolding of a according to t—first a, then $t\,a$, then $t\,b$ for all $b \in \mathsf{Fset}_2\ (t\ a)$, and so on. Thanks to CBD-a, such trees are at most Fbd-branching, hence representable in C. We have thus proved:

Prop. 6: $(\beta\ \mathsf{JF}, \mathsf{unf})$ is the final β-coalgebra.

This yields a coiterator $\mathsf{coiter} : (\alpha \to (\beta, \alpha)\ \mathsf{F}) \to \alpha \to \beta\ \mathsf{JF}$ such that $\mathsf{unf}\ (\mathsf{coiter}\ s) = \mathsf{Fmap\ id}\ (\mathsf{coiter}\ s) \circ s$ (cf. Fig. 2).

Structural coinduction: Since $\mathsf{LB}(\mathscr{C})$ is the greatest bisimulation on $\mathscr{C}$, it follows that Id is the greatest bisimulation on the quotient coalgebra $\mathscr{C}/_{\mathsf{LB}(\mathscr{C})}$. This gives us the following coinduction principle on $(\beta\ \mathsf{JF}, \mathsf{unf})$ (which is a copy of $\mathscr{C}/_{\mathsf{LB}(\mathscr{C})}$): If R is a bisimulation relation, then $R \subseteq \mathsf{Id}$. Viewing bisimilarities via the relator structure (cf. Fig. 7, left) and using the predicate notation, we can rephrase the coinduction principle as follows:

Prop. 7: Let $\varphi : \beta\ \mathsf{JF} \to \beta\ \mathsf{JF} \to \mathsf{bool}$ and assume $\forall a\ b.\ \varphi\ a\ b \Rightarrow \mathsf{Fpred\ Eq}\ \varphi\ (\mathsf{unf}\ a)\ (\mathsf{unf}\ b)$ (where $\mathsf{Eq} : \beta \to \beta \to \mathsf{bool}$ is the equality predicate). Then $\forall a\ b.\ \varphi\ a\ b \Rightarrow a = b$.

RTC structure: Again, the functorial structure of the final coalgebra is standard, namely, $\mathsf{JFmap}\ f = \mathsf{coiter}$ $((\mathsf{Fmap}\ f\ \mathsf{id}) \circ \mathsf{unf})$. Moreover, JFset can be defined by collecting all the Fset_1 results of repeated unfolding, namely $\mathsf{Fset}_1\ a = \bigcup_{i \in \mathsf{nat}} \mathsf{collect}_{i,a}$, where $\mathsf{collect}_{i,a}$ is defined recursively on i as follows: $\mathsf{collect}_{0,a} = \emptyset$; $\mathsf{collect}_{i+1,a} = \mathsf{Fset}_1$ $(\mathsf{unf}\ a) \cup \bigcup\{\mathsf{collect}_{i,b}.\ b \in \mathsf{Fset}_2\ (\mathsf{unf}\ a)\}$. Similarly to IFpred, the relator JFpred can be described in terms of Fpred, by $\mathsf{JFpred}\ \varphi\ a_1\ a_2 \Leftrightarrow \mathsf{Fpred}\ \varphi\ (\mathsf{JFpred}\ \varphi)\ (\mathsf{unf}\ a_1)$ $(\mathsf{unf}\ a_2)$.

Prop. 8: $(\mathsf{JF}, \mathsf{JFmap}, \mathsf{JFset}, \mathsf{Fbd}^{\mathsf{Fbd}})$ is an RTC.

V. Formalization and Implementation

The results in this paper are formalized in Isabelle/HOL and implemented in ML as a prototypical definitional package, together with a few examples of applications. This development is publicly available [32].

A. Formalized Metatheory

Isabelle/HOL proved well suited for formalizing category theory over types, with relevant concepts, including functor and natural transformation, handled in a lightweight, family-free notation as polymorphic types or constants. The main (co)algebraic constructions of this paper correspond to the theories named LFP and GFP in our formal development.

These constructions require a theory of cardinals in HOL, including cardinal arithmetic and regular cardinals. Simple type theory does not cater for ordinals as a canonical collection of well-orders, a very convenient concept for the standard theory of cardinals. Therefore, we worked with well-orders directly, dispersed polymorphically over types, with cardinals defined as well-orders minimal with respect to initial-segment embeddings. This theory and its challenges are presented separately [29].

B. Definitional Package

Theorem 1 and its formalization form the basis of a new (co)datatype package for Isabelle/HOL. Users define (co)datatypes using an intuitive high-level specification syntax; internally, the package ensures that each specification corresponds to an RTC, defines the (co)datatype, and proves that the result is itself an RTC.

More specifically, each RTC is represented by an ML record consisting of the polymorphic constants and their properties as proved theorems, stored in Isabelle's theory database [36, §4.1]. The basic RTCs for unit, $+$, $\times$, func_α, fset, countable sets, and finite multisets are constructed in user space, as they do not require ML; users can construct and register custom RTCs in the same way.

In the simple (nonmutual) case, the package parses the right-hand side of a (co)datatype specification as a composition $\mathscr{F}$ of already defined RTCs and proves that itself forms an RTC as in Sect. IV-B. Then the package defines the initial algebra or final coalgebra for $\mathscr{F}$ and establishes automatically their characteristic theorems (for (co)recursion, (co)induction, etc.) and RTC structure as in Sect. IV-E or IV-F. All these are performed by specially tailored Isabelle tactics, whose running time is independent of the amount of nesting (unlike for the Melham–Gunter approach).

C. Example

We demonstrate the definitional package on the type of finitely branching trees of possibly infinite depth [32]:

$$\textbf{codatatype}\ \alpha\,\mathsf{tree_I} = \mathsf{Node}\ (\mathsf{lab}\colon \alpha)\ (\mathsf{sub}\colon (\alpha\,\mathsf{tree_I})\,\mathsf{list})$$

The declaration syntax allows named selectors (lab and sub).

The command derives the expected characteristic theorems for $\alpha\,\mathsf{tree_I}$, including the coinduction rule

$$\frac{\varphi\,x\,y \qquad \forall a\,b.\ \varphi\,a\,b \Rightarrow \mathsf{lab}\,a = \mathsf{lab}\,b \wedge \mathsf{list\_pred}\,\varphi\,(\mathsf{sub}\,a)\,(\mathsf{sub}\,b)}{x = y}$$

where $\mathsf{list\_pred}\,\varphi$ is the componentwise extension of φ to lists (Sect. IV-E). Corecursive (coiterative) functions can be defined using a convenient syntax; for example, tree reversal is specified below in terms of map and rev on lists:

corec trev **where**
$\mathsf{lab}\ (\mathsf{trev}\ t) = \mathsf{lab}\ t$
$\mathsf{sub}\ (\mathsf{trev}\ t) = \mathsf{rev}\ (\mathsf{map}\ \mathsf{trev}\ (\mathsf{sub}\ t))$

Using the tree coinduction rule and Isabelle's automation, we can prove the following lemma with a one-line proof:

lemma $\mathsf{trev}\ (\mathsf{trev}\ t) = t$

The (co)datatype package interacts seamlessly with the existing infrastructure for reasoning about (co)inductive predicates (defined via Knaster–Tarski), as illustrated by the following proof of König's lemma for $\alpha\,\mathsf{tree_I}$. We first need a stream type to represent infinite paths in a tree:

$$\textbf{codatatype}\ \alpha\,\mathsf{strm} = \mathsf{SCons}\ (\mathsf{hd}\colon \alpha)\ (\mathsf{tl}\colon \alpha\,\mathsf{strm})$$

Using the existing coinductive package, we can define the notions of an infinite tree and a proper path in a tree as greatest predicates satisfying the equations $\mathsf{infinite}\ t \Leftrightarrow (\exists u \in \mathsf{set}\ (\mathsf{sub}\ t).\ \mathsf{infinite}\ u)$ and $\mathsf{proper\_path}\ p\ t \Leftrightarrow \mathsf{hd}\ p = \mathsf{lab}\ t \wedge (\exists u \in \mathsf{set}\ (\mathsf{sub}\ t).\ \mathsf{proper\_path}\ (\mathsf{tl}\ p)\ u)$. The corecursive function kpath uses Hilbert choice (ε) to return a witness infinite path:

corec kpath **where**
$\mathsf{hd}\ (\mathsf{kpath}\ t) = \mathsf{lab}\ t$
$\mathsf{tl}\ (\mathsf{kpath}\ t) = \mathsf{kpath}\ (\varepsilon u.\ u \in \mathsf{set}\ (\mathsf{sub}\ t) \wedge \mathsf{infinite}\ u)$

We can then prove the desired lemma by coinduction:

lemma $\mathsf{infinite}\ t \Rightarrow \mathsf{proper\_path}\ (\mathsf{kpath}\ t)\ t$

VI. Further Related Work

Interactive theorem provers include various mechanisms for introducing new types, whether primitive (intrinsic), axiomatic, or definitional [6, p. 3]. In the world of HOL, the primitive type definition mechanism (Sect. II-A) and the datatype package (Sect. III-A) are the most widely used, but there are many others. Homeier [18] developed a package to define quotient types in HOL4, now ported to Isabelle [21]. Nominal Isabelle [33] extends HOL with infrastructure for reasoning about datatypes containing name binders; Urban is rebasing it on the quotient package, possibly in unison with our (co)datatype package, exploiting the support for non-free constructors. HOLCF, a HOL library for domain theory, has long included an axiomatic package for defining (co)recursive domains; Huffman [20] recast it into a purely definitional package, based on a large enough universal domain—a simplification that unfortunately is not available for general HOL datatypes. The package combines many of the categorical ideas present in our work, notably the modular mixture of recursion via enriched constructors. Some ideas have yet to be automated in a definitional package: Völker [34] sketches a categorical approach to datatypes that prefigures our work; Vos and Swierstra [35] elaborate an ad hoc construction for recursion through finite sets; and Paulson [27] designed building blocks for codatatypes.

PVS, whose logic is a simple type theory extended with dependent types and subtyping (but without polymorphism), provides monolithic axiomatic packages for datatypes [26] and codatatypes [11]. Hensel and Jacobs [16] illustrate the categorical approach to (co)datatypes in PVS by axiomatic declarations of various flavors of trees (including our $\mathsf{tree_F}$ and $\mathsf{tree_I}$) with associated (co)iterators and proof principles. HOL_ω, which extends HOL4 with higher-rank polymorphism, provides a safe primitive for introducing abstractly specified types [19]. Isabelle/ZF, based on ZFC, reduces (co)datatypes to (co)inductive predicates [28], with no support for mixed (co)recursion; for codatatypes, it relies on a concrete, definitional treatment of non-well-founded objects. In Agda and Coq, (co)datatypes are built into the underlying calculus. Mixed (co)recursion is possible [24] but not the combination with non-free types.

VII. Conclusion

We presented a theoretical framework for defining types in higher-order logic. The framework relies on the abstract notion of a rich type constructor (RTC), consisting of a type constructor plus further categorical structure. RTCs are closed under composition and (co)algebraic fixpoints, providing all the necessary ingredients to define (co)datatypes.

Our solution is foundational: The characteristic (co)datatype theorems are derived from an internal construction, rather than stated as axioms. Unlike the traditional Melham–Gunter approach, our solution is also fully compositional, enabling mutual and nested (co)recursion involving arbitrary combinations of datatypes, codatatypes, and custom RTCs.

There is a large body of previous work on (co)datatypes as (co)algebras in category theory. Our main contribution has been to adapt this work to achieve compatibility with HOL's restrictive type system. Our ideas are implemented in a prototypical definitional package for Isabelle/HOL. The package is expected to be included in the next official release of the theorem prover, making Isabelle the first HOL-based prover with general support for codatatypes and thereby answering a long-standing user request.

After implementing the original datatype package for Isabelle, Berghofer and Wenzel [6] suggested three areas for future work: codatatypes, non-freely generated types, and composition of definitional packages. Thirteen years later, their vision is very close to a full materialization. Although we focused on Isabelle, our approach is equally applicable to the other HOL-based theorem provers, such as HOL4 [9] HOL Light [15], and ProofPower–HOL [3].

Methodologically, we found that category theory helped us develop intuitions about the types of HOL, recasting them as richly structured objects rather than mere collections of elements. As a continuation of this program, we want to rebuke the myth that parametricity is inapplicable to HOL, by extending RTCs with a parametricity predicate and exploiting their relator nature. We also intend to accommodate further category theory insight into the world of theorem provers, such as the (co)induction mixture presented in Jacobs et al. [16], [17].

Acknowledgments. We thank Tobias Nipkow for encouraging this work. The second author thanks Andreas Lochbihler for inspiring discussions.

References

[1] P. Aczel and N. P. Mendler. A final coalgebra theorem. In *CTCS '89*, vol. 389 of *LNCS*, pp. 357–365. Springer, 1989.

[2] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, vol. 27 of *Applied Logic*. Springer, 2nd ed., 2002.

[3] R. D. Arthan. Some mathematical case studies in ProofPower–HOL. In *TPHOLs '04 (Emerging Trends)*, pp. 1–16, 2004.

[4] M. Barr. Terminal coalgebras in well-founded set theory. *Theor. Comput. Sci.*, 114(2):299–315, 1993.

[5] M. Barr. Additions and corrections to "Terminal coalgebras in well-founded set theory." *Theor. Comput. Sci.*, 124:189–192, 1994.

[6] S. Berghofer and M. Wenzel. Inductive datatypes in HOL—Lessons learned in formal-logic engineering. In *TPHOLs '99*, vol. 1690 of *LNCS*, pp. 19–36, 1999.

[7] A. Church. A Formulation of the Simple Theory of Types. *The Journal of Symbolic Logic*, 5(2):56–68, 1940.

[8] T. M. Daniel Hausmann and L. Schröder. Iterative circular coinduction for CoCasl in Isabelle/HOL. In *FASE '05*, LNCS, pp. 341–356, 2005.

[9] M. J. C. Gordon and T. F. Melham, eds. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[10] M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF: A Mechanised Logic of Computation*, vol. 78 of *LNCS*. Springer, 1979.

[11] H. Gottliebsen. Co-inductive proofs for streams in PVS. In *TPHOLs '07 (Emerging Trends)*, pp. 113–127, 2007.

[12] E. L. Gunter. Why we can't have SML-style `datatype` declarations in HOL. In *TPHOLs '92*, vol. A-20 of *IFIP Transactions*, pp. 561–568. North-Holland/Elsevier, 1993.

[13] E. L. Gunter. A broader class of trees for recursive type definitions for HOL. In *HUG '93*, vol. 780 of *LNCS*, pp. 141–154. Springer, 1994.

[14] J. Harrison. Inductive definitions: Automation and application. In *TPHOLs '95*, vol. 971 of *LNCS*, pp. 200–213. Springer, 1995.

[15] J. Harrison. HOL Light: A tutorial introduction. In *FMCAD '96*, vol. 1166 of *LNCS*, pp. 265–269. Springer, 1996.

[16] U. Hensel and B. Jacobs. Proof principles for datatypes with iterated recursion. In *Category Theory and Computer Science*, pp. 220–241, 1997.

[17] C. Hermida and B. Jacobs. Structural induction and coinduction in a fibrational setting. *Inf. Comput.*, 145(2):107–152, 1998.

[18] P. V. Homeier. A design structure for higher order quotients. In *TPHOLs '05*, vol. 3603 of *LNCS*, pp. 130–146. Springer, 2005.

[19] P. V. Homeier. Abstract data types in HOL-Omega. `http://permalink.gmane.org/gmane.comp.mathematics.hol/1168`, 18 Apr. 2011.

[20] B. Huffman. A purely definitional universal domain. In *TPHOLs '09*, vol. 5674 of *LNCS*, pp. 260–275. Springer, 2009.

[21] C. Kaliszyk and C. Urban. Quotients revisited for Isabelle/HOL. In *SAC '11*, pp. 1639–1644. ACM, 2011.

[22] E. G. Manes and M. A. Arbib. *Algebraic Approaches to Program Semantics*. Springer, 1986.

[23] T. F. Melham. Automating recursive type definitions in higher order logic. In *Current Trends in Hardware Verification and Automated Theorem Proving*, pp. 341–386. Springer, 1989.

[24] K. Nakata, T. Uustalu, and M. Bezem. A proof pearl with the fan theorem and bar induction—Walking through infinite trees with mixed induction and coinduction. In *APLAS '11*, vol. 7078 of *LNCS*, pp. 353–368. Springer, 2011.

[25] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, vol. 2283 of *LNCS*. Springer, 2002.

[26] S. Owre and N. Shankar. Abstract datatypes in PVS. Tech. report CSL-93-9R, C.S. Lab., SRI International, 1993.

[27] L. C. Paulson. Mechanizing coinduction and corecursion in higher-order logic. *J. Log. Comput.*, 7(2):175–204, 1997.

[28] L. C. Paulson. A fixedpoint approach to (co)inductive and (co)datatype definitions. In *Proof, Language, and Interaction*, pp. 187–212. MIT Press, 2000.

[29] A. Popescu and D. Traytel. Ordinals and cardinals in HOL. `http://www21.in.tum.de/~traytel/lics12_card.tgz`, 2012.

[30] J. J. M. M. Rutten. Relators and metric bisimulations. *Electr. Notes Theor. Comput. Sci.*, 11:252–258, 1998.

[31] J. J. M. M. Rutten. Universal coalgebra: A theory of systems. *Theor. Comput. Sci.*, 249:3–80, 2000.

[32] D. Traytel, A. Popescu, and J. C. Blanchette. Formal development associated with this paper. `http://www21.in.tum.de/~traytel/lics12_data.tgz`, 2012.

[33] C. Urban. Nominal techniques in Isabelle/HOL. *J. Autom. Reasoning*, 40(4):327–356, 2008.

[34] N. Völker. On the representation of datatypes in Isabelle/HOL. Isabelle Users Workshop, 1995.

[35] T. E. J. Vos and S. D. Swierstra. Inductive data types with negative occurrences in HOL. Thirty Five Years of Automath, 2002.

[36] M. Wenzel and B. Wolff. Building formal method tools in the Isabelle/Isar framework. In *TPHOLs '07*, vol. 4732 of *LNCS*, pp. 352–367. Springer, 2007.

APPENDIX

A. Coinduction up to Equality

Coinduction "up to equality" is a syntactic strengthening of the raw coinduction principle of Prop. 7 that reduces the coinduction proof task to disjunction with equality.

Prop. 9: Let $\varphi : \beta\ \mathsf{JF} \to \beta\ \mathsf{JF} \to \mathsf{bool}$ and assume $\forall a\ b.\ \varphi\ a\ b \Rightarrow \mathsf{Fpred\ Eq}\ (\varphi \mid \mathsf{Eq})\ (\mathsf{unf}\ a)\ (\mathsf{unf}\ b)$, where $\mathsf{Eq} : \beta \to \beta \to \mathsf{bool}$ is equality and '$|$' denotes disjunction of binary predicates. Then $\forall a\ b.\ \varphi\ a\ b \Rightarrow a = b$.

B. (Co)recursion versus (Co)iteration

(Co)recursion is a more powerful definition principle than (co)iteration, allowing, at (co)recursion time, the consideration of not only elements of the target type (i.e., results computed so far), but also the original values of the source type (Figs. 8 and 9). For example, the predecessor function on natural numbers cannot be defined by iteration without introducing auxiliary arguments, but it is definable by a trivial recursion.

C. Proof of Lemma 2

Lemma 2: Let $s : (\beta,\alpha)\ \mathsf{F} \to \alpha$. Then $|M_s| \leq (|\beta|+2)^{\mathsf{Suc\ Fbd}}$.

Proof: The definition of M_s "from above," as an intersection, is not helpful for establishing a cardinal bound. We need an alternative construction of M_s "from below," as a union. For this, we define the family $(K_i)_{i<\mathsf{Suc\ Fbd}}$ by transfinite recursion as follows:

- $K_i = \bigcup_{j<i} K_j$, if i is a limit ordinal (thus, $K_0 = \emptyset$);
- $K_{i+1} = K_i \cup \{s\ x.\ \mathsf{Fset}_2\ x \subseteq K_i\}$.

Let $K_\infty = \bigcup_{i<\mathsf{Suc\ Fbd}} K_i$. We must prove $M_s = K_\infty$. First, $K_\infty \subseteq M_s$ follows easily by induction on i using that M_s is an algebra. For the harder inclusion $K_\infty \subseteq M_s$, it suffices to show that K_∞ is an algebra. Let x be such that $x \in \mathsf{Fin}\,\beta\,K_\infty$, i.e., $\mathsf{Fset}_2\ x \subseteq K_\infty$. Since $\mathsf{Suc\ Fbd}$ is a regular cardinal and, by CBD-a, $|\mathsf{Fset}_2\ x| < \mathsf{Suc\ Fbd}$, we obtain $i < \mathsf{Suc\ Fbd}$ such that $\mathsf{Fset}_2\ x \subseteq K_i$. Hence $s\ x \in K_{i+1} \subseteq K_\infty$. It then suffices to show $|K_\infty| \leq (|\beta|+2)^{\mathsf{Suc\ Fbd}}$. The stronger property $\forall i < \mathsf{Suc\ Fbd}.\ |K_i| \leq (|\beta|+2)^{\mathsf{Suc\ Fbd}}$ follows by induction on i, via CBD-b and cardinal arithmetic. ■

D. Weak Pullbacks

We use a definition of weak pullbacks that restricts the participating functions on given sets. We define the predicate $\mathsf{wpull}\ A\ B_1\ B_2\ f_1\ f_2\ p_1\ p_2$ to hold iff for all $b_1 \in B_1, b_2 \in B_2$ if $f_1\ b_1 = f_2\ b_2$ holds, then there exist an $a \in A$ such that $p_1\ a = b_1 \wedge p_2\ a = b_2$. Thus, the WP property of a binary RTC says that if $\mathsf{wpull}\ A_1\ B_{11}\ B_{21}\ f_{11}\ f_{21}\ p_{11}\ p_{21}$ and $\mathsf{wpull}\ A_2\ B_{12}\ B_{22}\ f_{12}\ f_{22}\ p_{12}\ p_{22}$ hold, then so does $\mathsf{wpull}\ (\mathsf{Fin}\ A_1\ A_2)\ (\mathsf{Fin}\ B_{11}\ B_{12})\ (\mathsf{Fin}\ B_{21}\ B_{21})\ (\mathsf{Fmap}\ f_{11}\ f_{12})\ (\mathsf{Fmap}\ f_{21}\ f_{22})\ (\mathsf{Fmap}\ p_{11}\ p_{12})\ (\mathsf{Fmap}\ p_{21}\ p_{22})$.

Our definition is weaker than the standard notion from literature [31], since it does not require p_1, p_2, f_1, and f_2 to form a commutative diagram.

$$\begin{tikzcd}
(\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \arrow[r, "\mathsf{fld}"] \arrow[d, "\mathsf{Fmap\ id\ (iter\ }s)"'] & \alpha\ \mathsf{IF} \arrow[d, "\mathsf{iter}\ s"] \\
(\alpha, \beta)\ \mathsf{F} \arrow[r, "s"] & \beta
\end{tikzcd}
\qquad
\begin{tikzcd}
(\alpha, \alpha\ \mathsf{IF})\ \mathsf{F} \arrow[r, "\mathsf{fld}"] \arrow[d, "\mathsf{Fmap\ id}\ \langle \mathsf{id}, \mathsf{rec}\ s\rangle"'] & \alpha\ \mathsf{IF} \arrow[d, "\mathsf{rec}\ s"] \\
(\alpha, \alpha\ \mathsf{IF} \times \beta)\ \mathsf{F} \arrow[r, "s"] & \beta
\end{tikzcd}$$

Fig. 8. Iterator (left) and recursor (right) for the initial algebra α IF

$$\begin{tikzcd}
\beta \arrow[r, "s"] \arrow[d, "\mathsf{coiter}\ s"'] & (\alpha, \beta)\ \mathsf{F} \arrow[d, "\mathsf{Fmap\ id\ (coiter\ }s)"] \\
\alpha\ \mathsf{JF} \arrow[r, "\mathsf{unf}"] & (\alpha, \alpha\ \mathsf{JF})\ \mathsf{F}
\end{tikzcd}
\qquad
\begin{tikzcd}
\beta \arrow[r, "s"] \arrow[d, "\mathsf{corec}\ s"'] & (\alpha, \beta + \alpha\ \mathsf{JF})\ \mathsf{F} \arrow[d, "\mathsf{Fmap\ id\ ([corec\ }s\mathsf{, id]})"] \\
\alpha\ \mathsf{JF} \arrow[r, "\mathsf{unf}"] & (\alpha, \alpha\ \mathsf{JF})\ \mathsf{F}
\end{tikzcd}$$

Fig. 9. Coiterator (left) and corecursor (right) for the final coalgebra α JF

E. n-Ary Rich Type Constructors

Definition: An n-ary *rich type constructor* is a tuple (F, Fmap, Fset, Fbd), where

- F is an n-ary type constructor,
- $\mathsf{Fmap} : (\alpha_1 \to \beta_1) \to \cdots \to (\alpha_n \to \beta_n) \to (\alpha_1, \ldots, \alpha_n)\ \mathsf{F} \to (\beta_1, \ldots, \beta_n)\ \mathsf{F}$,
- $\mathsf{Fset}_i : (\alpha_1, \ldots, \alpha_n)\ \mathsf{F} \to \alpha_i\ \mathsf{set}$ for $i \in \{1, \ldots, n\}$,
- Fbd is an infinite cardinal number,

satisfying the following properties for $i \in \{1, \ldots, n\}$:

FUNC (F, Fmap) is a binary functor.

NAT$_i$ For all $\alpha_1, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n$, Fset_i is a natural transformation between $((\alpha_1, \ldots, \alpha_{i-1}, \_, \alpha_{i+1}, \ldots, \alpha_n)\ \mathsf{F}, \mathsf{Fmap})$ and (set, image).

WP (F, Fmap) preserves weak pullbacks.

CONG If $\forall a \in \mathsf{Fset}_i\ x.\ f_i\ a = g_i\ a$ for all $i \in \{1, \ldots, n\}$, then $\mathsf{Fmap}\ f_1\ \ldots\ f_n\ x = \mathsf{Fmap}\ g_1\ \ldots\ g_n\ x$.

CBD The following cardinal-bound conditions hold:

a. $\forall x : (\alpha_1, \ldots, \alpha_n)\ \mathsf{F}.\ \ |\mathsf{Fset}_i\ x| \le \mathsf{Fbd}$ for all $i \in \{1, \ldots, n\}$;

b. $|\mathsf{Fin}\ A_1\ \ldots\ A_n| \le (|A_1| + \ldots + |A_n| + 2)^{\mathsf{Fbd}}$.

Composition: Given m-ary RTCs $\mathscr{F}_i = (\mathsf{F}^i, \mathsf{Fmap}^i, \mathsf{Fset}^i, \mathsf{Fbd}^i)$ with $i \in \{1, \ldots, n\}$ and an n-ary RTC $\mathscr{G} = (\mathsf{G}, \mathsf{Gmap}, \mathsf{Gset}, \mathsf{Gbd})$, their composition is the m-ary RTC $\mathscr{H} = \mathscr{G} \circ (\mathscr{F}_1, \ldots, \mathscr{F}_n)$ defined as follows:

- (H, Hmap) is the functorial composition of (G, Gmap) with $(\mathsf{F}^i, \mathsf{Fmap}^i)$;
- $\mathsf{Hset}_j\ y = \bigcup_{i=1}^{n} \bigcup_{x \in \mathsf{Gset}_i\ y} \mathsf{Fset}^i_j\ x$ for $j \in \{1, \ldots, m\}$;
- $\mathsf{Hbd} = \mathsf{Gbd} * (\mathsf{Fbd}^1 + \ldots + \mathsf{Fbd}^n)$.

In general, the RTCs $\mathscr{F}^i$ may have different arities m_i. This case is reducible to the above definition of composition using *lifting* of RTCs. Any m-ary RTC $\mathscr{F}$ can be lifted to an $(m+1)$-ary RTC by simply setting $\mathsf{Fset}_{m+1} x = \emptyset$.

Furthermore, the variables of the type constructors F^i need not be disjoint. For the composition of $(\gamma_1, \gamma_2)\ \mathsf{G} = \gamma_1 + \gamma_2$ with $\alpha_2\ \mathsf{F}^1 = \alpha_2\ \mathsf{func}_{\alpha_1}$ and $(\alpha_1, \alpha_2)\ \mathsf{F}^2 = \alpha_1 \times \alpha_2$, the result is a unary type constructor parameterized by α_1, i.e., $\alpha_2\ \mathsf{H}_{\alpha_1} = (\alpha_1 \to \alpha_2) + (\alpha_1 \times \alpha_2)$. The type variable α_1 does not appear in the composed type constructor because it is is a parameter and not a type variable of F^1. Therefore, a forgetful RTC operation that turns $(\alpha_1, \alpha_2)\ \mathsf{F}^2$ into $\alpha_2\ \mathsf{F}'$ is needed. We define the *kill* operation on an m-ary RTC $\mathscr{F}$, resulting in an $(m-1)$-ary RTC by forgetting Fset_1 and setting $\mathsf{Fmap}' = \mathsf{Fmap\ id}$. To obtain CBD-b for F', we must also adjust the cardinal bound: $\mathsf{Fbd}' = |\alpha_1| * \mathsf{Fbd}$.