Witnessing (Co)datatypes

Jasmin Christian Blanchette[1], Andrei Popescu[2], and Dmitriy Traytel[1]

[1] Fakultät für Informatik, Technische Universität München, Germany
[2] School of Science and Technology, Middlesex University, UK

Abstract. Datatypes and codatatypes are useful for specifying and reasoning about (possibly infinite) computational processes. The Isabelle/HOL proof assistant has recently been extended with a definitional package that supports both. We describe a complete procedure for deriving nonemptiness witnesses in the general mutually recursive, nested case—nonemptiness being a proviso for introducing types in higher-order logic.

1 Introduction

Proof assistants, or interactive theorem provers, are becoming increasingly popular as vehicles for formalizing the metatheory of logical systems and programming languages. Such developments often involve datatypes and codatatypes in various constellations. For example, Lochbihler's formalization of the Java memory model represents possibly infinite executions using a codatatype [28]. Codatatypes are also useful to capture lazy data structures, such as Haskell's lists.

A popular and expanding family of proof assistants, heavily used in software and hardware verification, are those based on higher-order logic (HOL)—examples include HOL4 [39], HOL Light [19], Isabelle/HOL [32], HOL Zero [1] and ProofPower [5]. They are traditionally implemented around a trusted inference kernel through which all theorems are generated. Various definitional packages reduce high-level specifications to primitive inferences; characteristic theorems are derived rather than postulated. This reduces the amount of code that must be trusted. We recently extended Isabelle/HOL with a definitional package for mutually recursive, nested (co)datatypes [10, 14]. While some proof assistants support codatatypes (e.g., Agda, Coq, and PVS), Isabelle is the first to provide a definitional implementation.

In this paper, we focus on a fundamental problem posed by any HOL development that extends the type infrastructure: proofs of, or "witnesses" for, the nonemptiness of newly introduced types. Besides its importance to formal logic engineering, the problem also enjoys theoretical relevance, since it essentially amounts to the decision problem for the nonemptiness of open-ended, mutual, nested (co)datatypes. Furthermore, our modular witness generation algorithm is relevant outside the proof assistant world, in areas such as program synthesis [18].

Our starting point is the nonemptiness requirement on HOL types. This is a well-known design decision connected to the presence of Hilbert choice in HOL [17, 33]—it is embraced by all HOL-based provers. The inductive specification

$$\texttt{datatype}\ \alpha\ \mathsf{fstream} = \mathsf{FSCons}\ \alpha\ (\alpha\ \mathsf{fstream})$$

of "finite streams" must be rejected because it would lead to an empty datatype.

While checking nonemptiness appears to be an easy reachability test, nested recursion complicates the picture, as shown by this attempt to define infinitely branching trees with finite branches by nested recursion via a codatatype of streams:

$$\texttt{codatatype}\ \alpha\ \mathsf{stream} = \mathsf{SCons}\ \alpha\ (\alpha\ \mathsf{stream})$$
$$\texttt{datatype}\ \alpha\ \mathsf{tree} = \mathsf{Node}\ \alpha\ ((\alpha\ \mathsf{tree})\ \mathsf{stream})$$

The second definition should fail: To get a witness for α tree, we would need a witness for (α tree) stream, and vice versa. Replacing streams with finite lists should make the definition acceptable, because the empty list stops the recursion. Even though final coalgebras are never empty (except in trivial cases), here the datatype provides a better witness (the empty list) than the codatatype (which requires an α tree to build an (α tree) stream). Mutual, nested datatype specifications can be arbitrarily complex:

$$\texttt{datatype}\ (\alpha,\beta)\ \mathsf{tree} = \mathsf{Leaf}\ \beta \mid \mathsf{Branch}\ ((\alpha + (\alpha,\beta)\ \mathsf{tree})\ \mathsf{stream})$$
$$\texttt{codatatype}\ (\alpha,\beta)\ \mathsf{ltree} = \mathsf{LNode}\ \beta\ ((\alpha + (\alpha,\beta)\ \mathsf{ltree})\ \mathsf{stream})$$
$$\texttt{datatype}\ \mathsf{t}_1 = \mathsf{T}_{11}\ (((\mathsf{t}_1,\mathsf{t}_2)\ \mathsf{ltree})\ \mathsf{stream}) \mid \mathsf{T}_{12}\ (\mathsf{t}_1 \times (\mathsf{t}_2 + \mathsf{t}_3)\ \mathsf{stream})$$
$$\texttt{and}\ \mathsf{t}_2 = \mathsf{T}_2\ ((\mathsf{t}_1 \times \mathsf{t}_2)\ \mathsf{list})\ \ \texttt{and}\ \mathsf{t}_3 = \mathsf{T}_3\ ((\mathsf{t}_1, (\mathsf{t}_3,\mathsf{t}_3)\ \mathsf{tree})\ \mathsf{tree})$$

The definitions are legitimate, but the last group should be rejected if t_2 is replaced by t_3 in the constructor T_{11}.

What makes the problem interesting is our open-endedness assumption: The type constructors handled by our package are not syntactically predetermined. In particular, they are not restricted to polynomial functors—the user can register new type constructors in the package database after establishing a few semantic properties.

Our solution exploits the package's abstract, functorial view of types. Each (co)datatype, and more generally each functor (type constructor) that participates in a definition, carries its own witnesses together with soundness proofs. Operations such as functorial composition, initial algebra, and final coalgebra derive their witnesses from those of the operands. Each computational step performed by the package is certified in HOL. The solution is complete: Given precise information about the functors participating in a definition, all nonempty datatypes are identified as such.

We start by recalling the package's abstract layer, which is based on category theory (Section 2). Then we look at a concrete instance: a variation of context-free grammars acting on finite sets and their associated possibly infinite derivation trees (Section 3). The example supplies precious building blocks to the nonemptiness proofs (Section 4). It also displays some unique characteristics of our package, such as support for nested recursion through non-free types. Many other features and user conveniences are described elsewhere [10, 13]. The Isabelle formalization covering the results presented here is publicly available [12]. The implementation is part of the latest Isabelle release [25] (Section 5).

Conventions. We work informally in a mathematical universe $\mathscr{S}$ of sets but adopt many conventions from higher-order logic and functional programming. Function application is normally written in prefix form without parentheses (e.g., $f\ x\ y$). Sets are ranged over by capital Roman letters ($A, B, \ldots$) and Greek letters ($\alpha, \beta, \ldots$). For n-ary functions, we often prefer the curried form $f : \alpha_1 \to \cdots \to \alpha_n \to \beta$ to the tuple form $f : \alpha_1 \times \cdots \times \alpha_n \to \beta$ but occasionally pass tuples to curried functions. Polymorphic operators are regarded as families of higher-order constants indexed by sets.

Operators on sets are normally written in postfix form: $\alpha\ \mathsf{set}$ is the powerset of α, consisting of sets of elements of α; $\alpha\ \mathsf{fset}$ is the set of finite sets over α. Given $f : \alpha \to \beta$, $A \subseteq \alpha$, and $B \subseteq \beta$, $\mathsf{image}\ f\ A$, or $f \bullet A$, is the image of A through f, and $f^{-}\ B$ is the inverse image of B through f. The set unit contains a single element $()$, and $[n] = \{1, \ldots, n\}$. Prefix and postfix operators bind more tightly than infixes, so that $\alpha \times \beta\ \mathsf{set}$ is read as $\alpha \times (\beta\ \mathsf{set})$ and $f \bullet g\ x$ as $f \bullet (g\ x)$.

The notation $\overline{a}_n$, or simply $\overline{a}$, denotes $(a_1, \ldots, a_n)$. Given $\overline{a}_m$ and $\overline{b}_n$, $(\overline{a}, \overline{b})$ denotes the flat tuple $(a_1, \ldots, a_m, b_1, \ldots, b_n)$. Given n m-ary functions $f_1, \ldots, f_n$, the notation $\overline{f}\ \overline{a}$ stands for $(f_1\ \overline{a},\ \ldots,\ f_n\ \overline{a})$, and similarly $\overline{\alpha}\ \overline{\mathsf{F}} = (\overline{\alpha}\ \mathsf{F}_1, \ldots, \overline{\alpha}\ \mathsf{F}_n)$. Depending on the context, $\overline{\alpha}_n\ \mathsf{F}$ either denotes the application of F to $\overline{\alpha}$ or merely indicates that F is an n-ary set operator.

2 The Category Theory behind the Package

User-specified (co)datatypes and their characteristic theorems are derived from underlying constructions adapted from category theory. The central concept is that of bounded natural functors, a well-behaved class of functors with additional structure.

2.1 Functors and Functor Operations

We consider operators F on sets, which we call *set constructors*. We are interested in set constructors that are *functors* on the category of sets and functions, i.e., that are equipped with an action on morphisms commuting with identities and composition. This action is a polymorphic constant $\mathsf{Fmap} : (\alpha_1 \to \beta_1) \to \cdots \to (\alpha_n \to \beta_n) \to \overline{\alpha}\ \mathsf{F} \to \overline{\beta}\ \mathsf{F}$ that satisfies $\mathsf{Fmap}\ \mathsf{id}^n = \mathsf{id}$ and $\mathsf{Fmap}\ (g_1 \circ f_1)\ \ldots\ (g_n \circ f_n) = \mathsf{Fmap}\ \overline{g} \circ \mathsf{Fmap}\ \overline{f}$. Formally, functors are pairs $(\mathsf{F}, \mathsf{Fmap})$. Basic examples are presented below.

Identity functor $(\mathsf{ID}, \mathsf{id})$. The identity maps any set and any function to itself.

(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 set constructor $\overline{\beta}\ \mathsf{C}_{n,\alpha} = \alpha$ and the action $\mathsf{Cmap}_{n,\alpha}\ f_1\ \ldots\ f_n = \mathsf{id}$. We write C_α for $\mathsf{C}_{1,\alpha}$.

Sum functor $(+, \oplus)$. The sum $\alpha_1 + \alpha_2$ 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 $f_1 : \alpha_1 \to \beta_1$ and $f_2 : \alpha_2 \to \beta_2$, let $f_1 \oplus f_2 : \alpha_1 + \alpha_2 \to \beta_1 + \beta_2$ be the function sending $\mathsf{Inl}\ a_1$ to $\mathsf{Inl}\ (f_1\ a_1)$ and $\mathsf{Inr}\ a_2$ to $\mathsf{Inr}\ (f_2\ a_2)$.

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 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 $\lambda a.\ (f_1\ a, f_2\ a)$. 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 (func_α, comp_α). Given a set α, let $\beta\ \mathsf{func}_\alpha = \alpha \to \beta$. For all $g : \beta \to \gamma$, let $\mathsf{comp}_\alpha\ g : \beta\ \mathsf{func}_\alpha \to \gamma\ \mathsf{func}_\alpha$ be $\mathsf{comp}_\alpha\ g\ f = g \circ f$.

Powerset functor (set, image). For all $f : \alpha \to \beta$, the function $\mathsf{image}\ f : \alpha\ \mathsf{set} \to \beta\ \mathsf{set}$ sends each subset A of α to the image of A through the function $f : \alpha \to \beta$.

Bounded k-powerset functor (set_k, image). Given a cardinal k, for all sets α, the set $\alpha\ \mathsf{set}_k$ carves out from $\alpha\ \mathsf{set}$ only those sets of cardinality less than k. The finite powerset functor fset corresponds to $\mathsf{set}_{\aleph_0}$.

Functors can be composed to form complex functors. Composition requires the functors F_j to take the same type arguments $\overline{\alpha}$ in the same order. The operations of permutation and lifting, together with the identity and (n,α)-constant functors, make it possible to compose functors freely. Let Func_n be the collection of n-ary functions.

Composition. Given $\overline{\alpha}\ \mathsf{F}_j$ for $j \in [n]$ and $\overline{\beta}_n\ \mathsf{G}$, the *functor composition* $\mathsf{G} \circ \overline{\mathsf{F}}$ is defined on objects as $(\overline{\alpha}\ \overline{\mathsf{F}})\ \mathsf{G}$ and similarly on morphisms.

Permutation. Given $\mathsf{F} \in \mathsf{Func}_n$ and $i, j \in [n]$ with $i < j$, the *(i,j)-permutation* of F, written $\mathsf{F}^{(i,j)} \in \mathsf{Func}_n$, is defined on objects as $\overline{\alpha}\ \mathsf{F}^{(i,j)} = (\alpha_1, \ldots, \alpha_{i-1}, \alpha_j, \alpha_{i+1}, \ldots, \alpha_{j-1}, \alpha_i, \alpha_{j+1}, \ldots, \alpha_n)\ \mathsf{F}$ and similarly on morphisms.

Lifting. Given $\mathsf{F} \in \mathsf{Func}_n$, the *lifting* of F, written $\mathsf{F}{\uparrow} \in \mathsf{Func}_{n+1}$, is defined on objects as $(\overline{\alpha}_n, \alpha_{n+1})\ \mathsf{F}{\uparrow} = \overline{\alpha}_n\ \mathsf{F}$ and similarly on morphisms. In other words, $\mathsf{F}{\uparrow}$ is obtained from F by adding a superfluous argument.

Datatypes are defined by taking the initial algebra of a set of functors and codatatypes by taking the final coalgebra. Both operations are partial.

Initial algebra. Given n $(m+n)$-ary functors $(\overline{\alpha}_m, \overline{\beta}_n)\ \mathsf{F}_j$, their (*mutual*) *initial algebra* consists of n m-ary functors $\overline{\alpha}\ \mathsf{IF}_j$ that satisfy the isomorphism $\overline{\alpha}\ \mathsf{IF}_j \cong (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{IF}})\ \mathsf{F}_j$ minimally. (The variables $\overline{\alpha}$ are the passive parameters, and $\overline{\beta}$ are the fixpoint variables.) The functors IF_j are characterized by

- n polymorphic *folding bijections* (constructors) $\mathsf{ctor}_j : (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{IF}})\ \mathsf{F}_j \to \overline{\alpha}\ \mathsf{IF}_j$ and
- n polymorphic *iterators* $\mathsf{fold}_j : \left(\prod_{k\in[n]} (\overline{\alpha}, \overline{\beta})\ \mathsf{F}_k \to \beta_k\right) \to \overline{\alpha}\ \mathsf{IF}_j \to \beta_j$

and subject to the following properties (for all $j \in [n]$):

- Iteration equations: $\mathsf{fold}_j\ \overline{s} \circ \mathsf{ctor}_j = s_j \circ \mathsf{Fmap}\ \mathsf{id}^m\ (\overline{\mathsf{fold}\ \overline{s}})$.
- Unique characterization of iterators: Given $\overline{\beta}$ and $\overline{s}$, the only functions $f_j : \overline{\alpha}\ \mathsf{IF}_j \to \beta_j$ satisfying $f_j \circ \mathsf{ctor}_j = s_j \circ \mathsf{Fmap}\ \mathsf{id}^m\ \overline{f}$ are $\mathsf{fold}_j\ \overline{s}$.

The functorial actions IFmap_j for IF_j are defined by iteration in a standard way.

Final coalgebra. The final coalgebra operation is categorically dual to initial algebra. Given n $(m+n)$-ary functors $(\overline{\alpha}_m, \overline{\beta}_n)\ \mathsf{F}_j$, their (*mutual*) *final coalgebra* consists of n m-ary functors $\overline{\alpha}\ \mathsf{JF}_j$ that satisfy the isomorphism $\overline{\alpha}\ \mathsf{JF}_j \cong (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{JF}})\ \mathsf{F}_j$ maximally. The functors JF_j are characterized by

- n polymorphic *unfolding bijections* (destructors) $\mathsf{dtor}_j : \overline{\alpha}\ \mathsf{JF}_j \to (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{JF}})\ \mathsf{F}_j$ and
- n polymorphic *coiterators* $\mathsf{unfold}_j : \left(\prod_{k\in[n]} \beta_k \to (\overline{\alpha}, \overline{\beta})\ \mathsf{F}_k\right) \to \beta_j \to \overline{\alpha}\ \mathsf{JF}_j$

and subject to the following properties:

- Coiteration equations: $\mathsf{dtor}_j \circ \mathsf{unfold}_j\ \overline{s} = \mathsf{Fmap}\ \mathsf{id}^m\ (\overline{\mathsf{unfold}\ \overline{s}}) \circ s_j$.
- Unique characterization of coiterators: Given $\overline{\beta}$ and $\overline{s}$, the only functions $f_j : \beta_j \to \overline{\alpha}\ \mathsf{JF}_j$ satisfying $\mathsf{dtor}_j \circ f_j = \mathsf{Fmap}\ \mathsf{id}^m\ \overline{f} \circ s_j$ are $\mathsf{unfold}_j\ \overline{s}$.

The functorial actions JFmap_j for JF_j are defined by coiteration in the standard way.

2.2 Bounded Natural Functors

The (co)datatype package is based on a class $\mathscr{B}$ of functors, called *bounded natural functors* (*BNFs*). The particular axioms defining $\mathscr{B}$ are described in [14]. The class $\mathscr{B}$ contains all the basic functors (except for unbounded powerset) and is closed under the operations described in Section 2.1.

Unlike the (co)datatype specification mechanisms of most state-of-the-art proof assistants (including those based on higher order logic or type theory [9, 15, 39, etc.]), in our package the involved types are not syntactically predetermined by a fixed grammar. $\mathscr{B}$ does include the class of polynomial functors, but is open-ended in that users may register further functors as members of $\mathscr{B}$.

The registration process takes place as follows. The user provides a type constructor F and its associated BNF structure (in the form of polymorphic HOL constants), including the Fmap functorial action on objects. Then the user verifies the BNF properties, e.g., that $(\mathsf{F}, \mathsf{Fmap})$ is indeed a functor. After this, the new BNF is integrated and can appear nested in future (co)datatype definitions. For example, Isabelle users have recently introduced the BNFs α bag, of bags (i.e., bags with finite-multiplicity elements) over α, and α list_5, of lists of at most five elements. Other nonstandard BNFs can be produced using the nonfree datatype package [38].

Besides closure under functor operations, another important question for theorem proving is how to state induction and coinduction abstractly, irrespective of the shape of the functor. We know how to state induction on lists, or trees, but how about on initial algebras of arbitrary functors?

Our answer is based on enriching the structure of functors $\overline{\alpha}_n\ \mathsf{F}$ with additional data: For each $i \in [n]$, BNFs must provide a natural transformation $\mathsf{Fset}^i : \overline{\alpha}\ \mathsf{F} \to \alpha_i\ \mathsf{set}$ that gives, for $x \in \overline{\alpha}\ \mathsf{F}$, the set of α_i-atoms that take part in x. For example, if $(\alpha_1, \alpha_2)\ \mathsf{F} = \alpha_1 \times \alpha_2$, then $\mathsf{Fset}^1\ (a_1, a_2) = \{a_1\}$ and $\mathsf{Fset}^2\ (a_1, a_2) = \{a_2\}$; if $\alpha\ \mathsf{F} = \alpha\ \mathsf{list}$ (the list functor, obtained as minimal solution to $\beta \cong \mathsf{unit} + \alpha \times \beta$), then Fset ($= \mathsf{Fset}^1$) applied to a list x gives all the elements appearing in x.[3]

Given $j \in [n]$, the elements of $\mathsf{Fset}_j^{m+k}\ x$ (for $k \in [n]$) are the recursive components of $\mathsf{ctor}_j\ x$. (Notice that subscripts select functors F_j in the tuple $\overline{\mathsf{F}}$, whereas superscripts select Fset operators for different arguments of F_j.) The explicit modeling of the recursive components makes it possible to state induction and coinduction abstractly for arbitrary BNFs— [14] and Appendix A give more details.

[3] This Fset has similarities with Pierce's notion of support from his account of (co)inductive types [35] and with Abel and Altenkirch's urelement relation from their framework for strong normalization [3]. A distinguishing feature of our notion is the additional consideration of categorical structure [14].

3 Coinductive Derivation Trees

We next study a concrete codatatype definable with our package. It consists of derivation trees for a context-free grammar, where we perform the following changes to the usual setting: Trees are possibly infinite and the generated words are not lists, but finite sets. The formalization of this example [12] lays at the heart of our results presented in the next section. Indeed, this particular codatatype will provide the infrastructure for tracking nonemptiness of arbitrary (co)datatypes.

We take a few liberties with Isabelle notations to lighten the presentation; in particular, until Section 4, we ignore the distinction between types and sets.

Definition of Derivation Trees. We fix a set T of *terminals* and a set N of *nonterminals*. The command

$$\texttt{codatatype}\ \mathsf{dtree} = \mathsf{Node}\ (\mathsf{root}\colon \mathsf{N})\ (\mathsf{cont}\colon (\mathsf{T} + \mathsf{dtree})\ \mathsf{fset})$$

introduces a constructor $\mathsf{Node} : \mathsf{N} \to (\mathsf{T} + \mathsf{dtree})\ \mathsf{fset} \to \mathsf{dtree}$ and two selectors $\mathsf{root} : \mathsf{dtree} \to \mathsf{N}$, $\mathsf{cont} : \mathsf{dtree} \to (\mathsf{T} + \mathsf{dtree})\ \mathsf{fset}$. A tree has the form $\mathsf{Node}\ n\ as$, where n is a nonterminal (the tree's *root*) and *as* is a finite set of terminals and trees (its *continuation*). The `codatatype` keyword indicates that this tree formation rule may be applied an infinite number of times.

Given the above definition of dtree, the package first composes the input BNF to the final coalgebra operation $\mathsf{pre\_dtree} = (\times) \circ (\mathsf{C}_{\mathsf{N}}, \mathsf{fset} \circ ((+) \circ (\mathsf{C}_{\mathsf{T}}, \mathsf{ID})))$ from the constants N and T, identity, sum, product, and finite set. In the sequel, we prefer the more readable notation $\alpha\ \mathsf{pre\_dtree} = \mathsf{N} \times (\mathsf{T} + \alpha)\ \mathsf{fset}$. Then it constructs the final coalgebra dtree ($= \mathsf{JF}$) from $\mathsf{pre\_dtree}$ ($= \mathsf{F}$).

The unfolding bijection $\mathsf{dtor} : \mathsf{dtree} \to \mathsf{dtree}\ \mathsf{pre\_dtree}$ is decomposed in two selectors: $\mathsf{root} = \mathsf{fst} \circ \mathsf{dtor}$ and $\mathsf{cont} = \mathsf{snd} \circ \mathsf{dtor}$. The constructor Node is defined as the inverse of the unfolding bijection. The basic properties of constructors and selectors (e.g., injectiveness, distinctness) are derived from those of sums and products.

After some preprocessing that involves splitting according to the indicated destructors, the abstract coiterator from Section 2 instantiates to the dtree coiterator $\mathsf{unfold} : (\beta \to \mathsf{N}) \to (\beta \to (\mathsf{T} + \beta)\ \mathsf{fset}) \to \beta \to \mathsf{dtree}$ characterized as follows: For all sets β, functions $r : \beta \to \mathsf{N}$, $c : \beta \to (\mathsf{T} + \beta)\ \mathsf{fset}$, and elements $b \in \beta$,

$$\mathsf{root}\ (\mathsf{unfold}\ r\ c\ b) = r\ b \qquad \mathsf{cont}\ (\mathsf{unfold}\ r\ c\ b) = (\mathsf{id} \oplus \mathsf{unfold}\ r\ c) \bullet c\ b$$

Intuitively, the coiteration contract reads as follows: Given a set β, to define a function $f : \beta \to \mathsf{dtree}$ we must indicate how to build a tree for each $b \in \beta$. The root is given by r, and its continuation is given corecursively by c. Formally, $f = \mathsf{unfold}\ r\ c$.

A Variation of Context-Free Grammars. We consider a variation of context-free grammars, acting on finite sets instead of sequences. We assume that the previously fixed sets T and N, of terminals and nonterminals, are finite and that we are given a set of *productions* $\mathsf{P} \subseteq \mathsf{N} \times (\mathsf{T} + \mathsf{N})\ \mathsf{fset}$. The triple $\mathsf{Gr} = (\mathsf{T}, \mathsf{N}, \mathsf{P})$ forms a (*set*) *grammar*, which is fixed for the rest of this section. Both finite and infinite derivation trees are of interest. The codatatype dtree provides the right universe for defining well-formed trees as a coinductive predicate.

Fixpoint (or Knaster–Tarski) (co)induction is provided in Isabelle/HOL by a separate package [34]. Fixpoint induction relies on the minimality of a predicate (the least fixpoint); dually, fixpoint coinduction relies on maximality (the greatest fixpoint). It is well-known that datatypes interact well with definitions by fixpoint induction. For codatatypes, both fixpoint induction and coinduction play an important role—the former to express safety properties, the latter to express liveness.

Well-formed derivation trees for Gr are defined coinductively as the greatest predicate $\mathsf{wf} : \mathsf{dtree} \to \mathsf{bool}$ such that, for all $t \in \mathsf{dtree}$,

$$\begin{aligned}\mathsf{wf}\, t \iff\ & (\mathsf{root}\, t,\, (\mathsf{id} \oplus \mathsf{root}) \bullet \mathsf{cont}\, t) \in \mathsf{P} \ \wedge\ \mathsf{root} \text{ is injective on } \mathsf{Inr}^{-}(\mathsf{cont}\, t) \ \wedge \\ & \forall t' \in \mathsf{Inr}^{-}(\mathsf{cont}\, t).\ \mathsf{wf}\, t'\end{aligned}$$

Each nonterminal node of a well-formed derivation tree t represents a production. This is achieved formally by three conditions: (1) the root of t forms a production together with the terminals constituting its successor leaves and the roots of its immediate subtrees; (2) no two immediate subtrees of t have the same root; (3) properties 1 and 2 also hold for the immediate subtrees of t. The definition's coinductive nature ensures that these properties hold for arbitrarily deep subtrees of t, even if t has infinite depth.

In contrast to wellformedness, the notions of subtree, frontier (the set of terminals appearing in a tree), and interior (the set of nonterminals appearing in a tree) require inductive definitions. The *subtree* relation is defined as the least predicate $\mathsf{subtr} : \mathsf{dtree} \to \mathsf{dtree} \to \mathsf{bool}$ such that $\mathsf{subtr}\, t\, t' \iff t = t' \vee (\exists t''.\ \mathsf{subtr}\, t\, t'' \wedge \mathsf{Inr}\, t'' \in \mathsf{cont}\, t')$ holds for all $t, t' \in \mathsf{dtree}$. We write $\mathsf{Subtr}\, t$ for the set of subtrees of t. Frontier $\mathsf{Fr} : \mathsf{dtree} \to \mathsf{T}\ \mathsf{set}$ and interior $\mathsf{Itr} : \mathsf{dtree} \to \mathsf{N}\ \mathsf{set}$ are defined similarly.

The language generated by the grammar Gr from a nonterminal $n \in \mathsf{N}$ (via possibly infinite derivation trees) is defined as $\mathscr{L}_{\mathsf{Gr}}(n) = \{\mathsf{Fr}\, t \mid \mathsf{wf}\, t \wedge \mathsf{root}\, t = n\}$.

Regular Derivation Trees. A derivation tree is *regular* if each subtree is uniquely determined by its root. Formally, we define $\mathsf{regular}\, t$ as the existence of a function $f : \mathsf{N} \to \mathsf{Subtr}\, t$ such that $\forall t' \in \mathsf{Subtr}\, t.\ f\ (\mathsf{root}\, t') = t'$. The regular language of a nonterminal is defined as $\mathscr{L}^{\mathsf{r}}_{\mathsf{Gr}}(n) = \{\mathsf{Fr}\, t \mid \mathsf{wf}\, t \wedge \mathsf{root}\, t = n \wedge \mathsf{regular}\, t\}$.

Given a possibly nonregular derivation tree t_0, a *regular cut* of t_0 is a regular tree $\mathsf{rcut}\, t_0$ such that $\mathsf{Fr}\ (\mathsf{rcut}\, t_0) \subseteq \mathsf{Fr}\, t_0$. Here is one way to perform the cut:

1. Choose a subtree of t_0 for each interior node $n \in \mathsf{Itr}\, t_0$ via a function $\mathsf{pick} : \mathsf{Itr}\, t_0 \to \mathsf{Subtr}\, t_0$ with $\forall n \in \mathsf{Itr}\, t_0.\ \mathsf{root}\ (\mathsf{pick}\, n) = n$.
2. Traverse t_0 and replace each subtree with root n with $\mathsf{pick}\, n$. The replacement should be performed hereditarily, i.e., also in the emerging subtree $\mathsf{pick}\, n$.

This replacement task is elegantly achieved by the corecursive function $\mathsf{H} : \mathsf{Itr}\, t_0 \to \mathsf{dtree}$ defined as $\mathsf{unfold}\ r\ c$, where $r : \mathsf{Itr}\, t_0 \to \mathsf{N}$ and $c : \mathsf{Itr}\, t_0 \to (\mathsf{T} + \mathsf{Itr}\, t_0)\ \mathsf{fset}$ are specified as follows: $r\, n = n$ and $c\, n = (\mathsf{id} \oplus \mathsf{root}) \bullet \mathsf{cont}\ (\mathsf{pick}\, n)$. H is therefore characterized by the corecursive equations $\mathsf{root}\ (\mathsf{H}\, n) = n$ and $\mathsf{cont}\ (\mathsf{H}\, n) = (\mathsf{id} \oplus (\mathsf{H} \circ \mathsf{root})) \bullet \mathsf{cont}\ (\mathsf{pick}\, n)$. It is not hard to prove the following by fixpoint coinduction:

Lemma 1. For all $n \in \mathsf{Itr}\, t_0$, $\mathsf{H}\, n$ is regular and $\mathsf{Fr}\ (\mathsf{H}\, n) \subseteq \mathsf{Fr}\, t_0$. Moreover, $\mathsf{H}\, n$ is well-formed provided t_0 is well-formed.

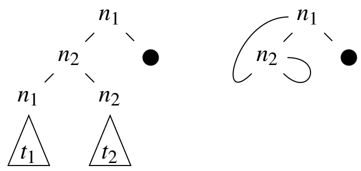

Fig. 1. A derivation tree (left) and its simplest regular cut (right)

We define $\mathsf{rcut}\ t_0$ to be $\mathsf{H}\ (\mathsf{root}\ t_0)$. Fig. 1 shows a derivation tree and its simplest regular cut. The bullet denotes a terminal, and t_1 and t_2 are arbitrary trees with roots n_1 and n_2. The loops denote infinite trees that are their own subtrees.

4 Computing Nonemptiness Witnesses

In the previous two sections, we referred to the codatatype dtree and other collections of elements as *sets*, ignoring an important aspect of HOL. While for most purposes sets and types can be identified in an abstract treatment of HOL, types have the additional restriction that they may not be empty. The main primitive way to define custom types in HOL is to specify from an existing type α a nonempty subset $A : \alpha\ \mathsf{set}$ that is isomorphic to the desired type. Hence, to register a collection of elements as a HOL type it is necessary to prove it nonempty.

Mutual datatype definitions are a particular case of the above situation, with the additional requirement that the nonemptiness proof should be performed automatically by the package. In the context of our package, we need to produce the relevant nonemptiness proofs taking into consideration arbitrary combinations of datatypes, codatatypes and user-defined BNFs.

A first temptation to tackle the problem is to follow the traditional approach of HOL datatype packages [7]: Try to unfold all the definitions of the involved nested datatypes, inlining them as additional components of the mutual definition, until only sums of products remain, and then perform a reachability analysis. At a closer inspection, this approach turns out problematic in our framework for several reasons. Due to open-endedness, there is no fixed set of basic types. Delving into nested types requires reproving nonemptiness facts, which is extremely inefficient. Moreover, it is not clear how to unfold datatypes nested in codatatypes or vice versa.

Counting on everything being eventually reducible to the fixed situation of sums of products, the traditional approach worries about nonemptiness only at datatype-definition time. Here, we look for a prophylactic solution instead, trying to prepare the BNFs in advance for future nonemptiness checks involving them. To this end, we ask the following: Given a mutual datatype definition involving several n-ary BNFs, what is the relevant information we need to know about their nonemptiness *without knowing what they look like* (hence, with no option to “delve” into them)? To answer this, we use a generalization of pointed types [23, 27], maintaining witnesses that assert conditional nonemptiness for combinations of arguments. We first present the solution by examples.

4.1 Examples

We start with the easy cases of products and sums. For $\alpha \times \beta$, the proof is as follows: Assuming $\alpha \neq \emptyset$ and $\beta \neq \emptyset$, we construct the witness $(a, b) \in \alpha \times \beta$ for some $a \in \alpha$ and $b \in \beta$. For $\alpha + \beta$, two proofs are possible: Assuming $\alpha \neq \emptyset$, we can construct $\mathsf{Inl}\ a$ for some $a \in \alpha$; alternatively, assuming $\beta \neq \emptyset$, we can construct $\mathsf{Inr}\ b$ for some $b \in \beta$.

To each BNF $\overline{\alpha}\ \mathsf{F}$, we associate a set of witnesses, each of the form $\mathsf{Fwit} : \alpha_{i_1} \to \cdots \to \alpha_{i_k} \to \overline{\alpha}\ \mathsf{F}$ for a subset $\{i_1, \ldots, i_k\} \subseteq [n]$. From a witness, we can construct a set-theoretic proof by following its signature (in the spirit of the Curry–Howard correspondence). Accordingly, $\mathsf{Inr} : \beta \to \alpha + \beta$ can be read as the following contract: Given a proof that β is nonempty, Inr yields a proof that $\alpha + \beta$ is nonempty.

When BNFs are composed, so are their witnesses. Thus, the two possible witnesses for the list-defining functor $(\alpha,\beta)\ \mathsf{pre\_list} = \mathsf{unit} + \alpha \times \beta$ are $\mathsf{wit\_pre\_list}_1 = \mathsf{Inl}\ ()$ and $\mathsf{wit\_pre\_list}_2\ a\ b = \mathsf{Inr}\ (a, b)$. The first witness subsumes the second one, because it unconditionally shows the collection nonempty, regardless of the potential emptiness of α and β. From this witness, we obtain the unconditional witness $\mathsf{list\_ctor}\ \mathsf{wit\_pre\_list}_1$ (i.e., Nil) for $\alpha\ \mathsf{list}$.

Because they can store infinite objects, codatatype set constructors are never empty (provided their arguments are nonempty). Compare the following:

$\texttt{datatype}\ \alpha\ \mathsf{fstream} = \mathsf{FSCons}\ \alpha\ (\alpha\ \mathsf{fstream})$
$\texttt{codatatype}\ \alpha\ \mathsf{stream} = \mathsf{SCons}\ \alpha\ (\alpha\ \mathsf{stream})$

The datatype definition fails because the best witness has a circular signature: $\alpha \to \alpha\ \mathsf{fstream} \to \alpha\ \mathsf{fstream}$. In contrast, the codatatype definition succeeds and produces the witness $(\lambda a.\ \mu s.\ \mathsf{SCons}\ a\ s) : \alpha \to \alpha\ \mathsf{stream}$, namely the (unique) stream s such that $s = \mathsf{SCons}\ a\ s$ for a given $a \in \alpha$. This stream is easy to define by coiteration.

Let us look at a pair of examples involving nesting:

$\texttt{datatype}\ (\alpha,\beta)\ \mathsf{tree} = \mathsf{Leaf}\ \beta \mid \mathsf{Branch}\ ((\alpha + (\alpha,\beta)\ \mathsf{tree})\ \mathsf{stream})$
$\texttt{codatatype}\ (\alpha,\beta)\ \mathsf{ltree} = \mathsf{LNode}\ \beta\ ((\alpha + (\alpha,\beta)\ \mathsf{ltree})\ \mathsf{stream})$

In the tree definition, the two constructors hide a sum BNF, giving us some flexibility. For the Leaf constructor, all we need is a witness $b \in \beta$, from which we construct $\mathsf{Leaf}\ b$. For Branch, we can choose the left-hand side of the nested $+$, completely dodging the recursive right-hand side: From a witness $a \in \alpha$, we construct $\mathsf{Branch}\ (\mu s.\ \mathsf{SCons}\ (\mathsf{Inl}\ a)\ s)$.

For the ltree functor, the two arguments to LNode are hiding a product, so the ltree-defining functor is $(\alpha,\beta,\gamma)\ \mathsf{pre\_ltree} = \beta \times (\alpha + \gamma)\ \mathsf{stream}$ with γ representing the corecursive component. Composition yields two witnesses for $\mathsf{pre\_ltree}$:

$\mathsf{wit\_pre\_ltree}_1\ a\ b = (b,\ \mu s.\ \mathsf{SCons}\ (\mathsf{Inl}\ a)\ s)$
$\mathsf{wit\_pre\_ltree}_2\ b\ c = (b,\ \mu s.\ \mathsf{SCons}\ (\mathsf{Inr}\ c)\ s)$

These can serve to build infinitely many witnesses for ltree. Fig. 2 enumerates the possible combinations, starting with $\mathsf{wit\_pre\_ltree}_1$. This witness requires only the non-corecursive components α and β being nonempty, and hence immediately yields

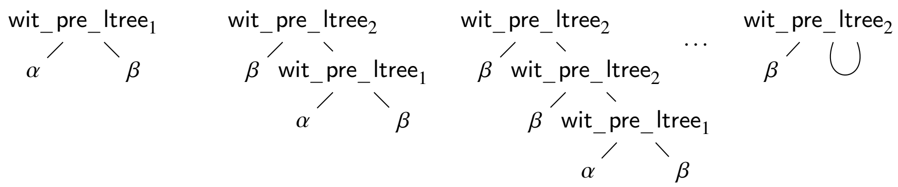

Fig. 2. Derivation trees for ltree witnesses

a witness $\mathsf{ltree\_wit}_1 : \alpha \to \beta \to (\alpha,\beta)$ ltree (by applying the constructor LNode). The second witness $\mathsf{wit\_pre\_ltree}_2$ requires both β and the corecursive component γ to be nonempty; it effectively "consumes" another ltree witness through γ. The consumed witness can again be either $\mathsf{wit\_pre\_ltree}_1$ or $\mathsf{wit\_pre\_ltree}_2$, and so on. At the limit, $\mathsf{wit\_pre\_ltree}_2$ is used infinitely often. The corresponding witness $\mathsf{ltree\_wit}_2 : \beta \to (\alpha,\beta)$ ltree can be defined by coiteration as $\lambda b.\ \mu t.\ \mathsf{wit\_pre\_ltree}_2\ b\ t$. It subsumes $\mathsf{ltree\_wit}_1$ and all the other finite witnesses. Were ltree to be defined as a datatype instead of a codatatype, $\mathsf{ltree\_wit}_1$ would be its best witness.

4.2 A General Solution

The nonemptiness problem for an n-ary set constructor F and a set of indices $I \subseteq [n]$ can be stated as follows: Does it hold that, for all sets $\overline{\alpha}_n$, $\overline{\alpha}\ \mathsf{F} \neq \emptyset$ whenever $\forall i \in I.\ \alpha_i \neq \emptyset$? We call F *I-witnessed* if the above question has a positive answer. E.g., set sum $(+)$ is $\{1\}$-, $\{2\}$-, and $\{1,2\}$-witnessed; set product $(\times)$ is $\{1,2\}$-witnessed; and α list is $\emptyset$-witnessed.

We are led to the following notion of soundness. Given an n-ary functor F, a set $\mathscr{I} \subseteq [n]$ set is (*witness-*)*sound* for F if F is I-witnessed for all $I \in \mathscr{I}$.

Now, when is such a set $\mathscr{I}$ also complete, in that it covers all witnesses? To answer this, first note that, if $I_1 \subseteq I_2$, then I_1-witnesshood implies I_2-witnesshood. Therefore, we are interested in retaining the witnesses completely only up to inclusion of sets of indices. We call a set $\mathscr{I} \subseteq [n]$ set (*witness-*)*complete* for F if for all $J \subseteq [n]$ such that F is J-witnessed, there exists $I \in \mathscr{I}$ such that $I \subseteq J$; (*witness-*)*perfect* for F if it is both sound and complete.

Here are perfect sets $\mathscr{I}_{\mathsf{F}}$ for some basic BNFs:

- Identity: $\mathscr{I}_{\alpha\ \mathsf{ID}} = \{\{\alpha\}\}$
- Constant: $\mathscr{I}_{\mathsf{C}_{n,\alpha}} = \{\emptyset\}$ $(\alpha \neq \emptyset)$
- Sum: $\mathscr{I}_{\alpha+\beta} = \{\{\alpha\},\{\beta\}\}$
- Product: $\mathscr{I}_{\alpha\times\beta} = \{\{\alpha,\beta\}\}$
- Function space: $\mathscr{I}_{\beta\ \mathsf{func}_\alpha} = \{\{\beta\}\}$ $(\alpha \neq \emptyset)$
- Bounded k-powerset: $\mathscr{I}_{\alpha\ \mathsf{set}_k} = \{\emptyset\}$

Parameters α_j are identified with their indices j to improve readability.

We need to maintain perfect sets across BNF operations. Let us start with composition, permutation, and lifting.

Theorem 1. Let $\mathsf{H} = \mathsf{G} \circ \overline{\mathsf{F}}_n$, where $\mathsf{G} \in \mathsf{Func}_n$ has a perfect set $\mathscr{J}$ and each $\mathsf{F}_j \in \mathsf{Func}_m$ has a perfect set $\mathscr{I}_j$. Then $\{\bigcup_{j\in J} I_j \mid J \in \mathscr{J} \wedge (I_j)_{j\in J} \in \prod_{j\in J} \mathscr{I}_j\}$ is a perfect set for H.

Proof sketch: Let $\mathscr{K} = \{\bigcup_{j\in J} I_j \mid J \in \mathscr{J} \wedge (I_j)_j \in \prod_{j\in J} \mathscr{I}_j\}$. We first prove that $\mathscr{K}$ is sound for H. Let $K \in \mathscr{K}$ and $\overline{\alpha}_m$ such that $\forall i \in K.\ \alpha_i \neq \emptyset$. By the definition of $\mathscr{K}$, we obtain $J \in \mathscr{J}$ and $(I_j)_{j\in J}$ such that (1) $K = \bigcup_{j\in J} I_j$ and (2) $\forall j \in J.\ I_j \in \mathscr{I}_j$. Using (1), we have $\forall j \in J.\ \forall i \in I_j.\ \alpha_i \neq \emptyset$. Hence, since each $\mathscr{I}_j$ is sound for F_j, $\forall j \in J.\ \overline{\alpha}\ \mathsf{F}_j \neq \emptyset$. Finally, since $\mathscr{J}$ is sound for G, we obtain $\overline{\alpha}\ \overline{\mathsf{F}}\ \mathsf{G} \neq \emptyset$, i.e., $\overline{\alpha}\ \mathsf{H} \neq \emptyset$.

We now prove that $\mathscr{K}$ is complete for H. Let $K \subseteq [m]$ be a H-witnessed set of indices. Let $\overline{\beta}_n$ be defined as $\beta_j = \mathsf{unit}$ if $j \in K$ and $= \emptyset$ otherwise and let $J = \{j \in [n] \mid \overline{\beta}\ \mathsf{F}_j \neq \emptyset\}$. Since K is H-witnessed, we obtain that $\overline{\beta}\ \mathsf{H} \neq \emptyset$, i.e., (1) $\overline{\beta}\ \overline{\mathsf{F}}\ \mathsf{G} \neq \emptyset$.

We show that (3) G is J-witnessed. Let $\overline{\gamma}_n$ such that $\forall j \in J.\ \gamma_j \neq \emptyset$. Thanks to the definition of J, we have $\forall j \in [n].\ \mathsf{F}_j \neq \emptyset \Rightarrow \gamma_j \neq \emptyset$, and therefore we obtain the functions $(f_j : \overline{\beta}\ \mathsf{F}_j \to \gamma_j)_{i\in[n]}$. With $\mathsf{Gmap}\ \overline{f} : \overline{\beta}\ \overline{\mathsf{F}}\ \mathsf{G} \to \overline{\gamma}\ \mathsf{G}$, by (1) we obtain $\overline{\gamma}\ \mathsf{G} \neq \emptyset$.

From (3), since $\mathscr{J}$ is complete for J, we obtain $J_1 \in \mathscr{J}$ such that $J_1 \subseteq J$. Let $j \in J_1$. By the definition of J, we have $\overline{\beta}\ \mathsf{F}_j \neq \emptyset$, making $\overline{\beta}\ \mathsf{F}_j$ K-witnessed (by definition of $\overline{\beta}$); hence, since $\mathscr{I}_j$ is F_j-complete, we obtain $I_j \in \mathscr{I}_j$ such that $I_j \subseteq K$. Then $K_1 = \bigcup_{j\in J_1} I_j$ belongs to $\mathscr{K}$ and is included in K. $\square$

Theorem 2. Let $\mathscr{I} \subseteq [n]$ set be a perfect set for F. Then $\mathscr{I}$ and $\mathscr{I}^{(i,j)}$ are perfect sets for $\mathsf{F}\!\uparrow$ and $\mathsf{F}^{(i,j)}$, respectively (where $\mathscr{I}^{(i,j)}$ is $\mathscr{I}$ with i and j exchanged in each of its elements).

Theorems 1 and 2 hold not only for functors but also for plain set constructors (with a further cardinality-monotonicity assumption needed for the completeness part of Theorem 1). The most interesting cases are the genuinely functorial ones of initial algebras and final coalgebras, which we discuss next.

Witnesses for initial algebras and final coalgebras will be essentially obtained by repeated compositions of the witnesses of the involved BNFs and the folding bijections, inductively in one case and coinductively in the other. The derivation trees from Section 3 turn out to be perfectly suited for recording the combinatorics of these compositions, so that both soundness and completeness yield easily.

For the rest of this subsection, we fix n $(m+n)$-ary functors $\overline{\beta}\ \mathsf{F}_j$ and assume each F_j has a perfect set $\mathscr{K}_j$. We start by constructing a (set) grammar $\mathsf{Gr} = (\mathsf{T}, \mathsf{N}, \mathsf{P})$ with $\mathsf{T} = [m]$, $\mathsf{N} = [n]$, and $\mathsf{P} = \{(j, \mathsf{cp}(K)) \mid K \in \mathscr{K}_j\}$, where, for each $K \subseteq [m+n]$, $\mathsf{cp}(K)$ is its copy to $[m]+[n]$ defined as $\mathsf{Inl} \bullet ([m] \cap K) \,\cup\, \mathsf{Inr} \bullet \{k \in [n] \mid m+k \in K\}$.

Here is the idea behind this construction. A mutual datatype definition as above introduces n isomorphisms:

$$\overline{\alpha}\ \mathsf{IF}_1 \,\cong\, (\overline{\alpha}, \overline{\alpha}\ \mathsf{IF}_1, \ldots, \overline{\alpha}\ \mathsf{IF}_n)\ \mathsf{F}_1 \qquad \cdots \qquad \overline{\alpha}\ \mathsf{IF}_n \,\cong\, (\overline{\alpha}, \overline{\alpha}\ \mathsf{IF}_1, \ldots, \overline{\alpha}\ \mathsf{IF}_n)\ \mathsf{F}_n$$

We are searching for conditions guaranteeing nonemptiness of the IF_j's. To this end, we walk these isomorphisms from left to right, reducing nonemptiness of $\overline{\alpha}\ \mathsf{IF}_j$ to that of $(\overline{\alpha}, \overline{\alpha}\ \mathsf{IF}_1, \ldots, \overline{\alpha}\ \mathsf{IF}_n)\ \mathsf{F}_j$. Moreover, nonemptiness of the latter can be reduced to nonemptiness of some $\alpha_{i_1}, \ldots, \alpha_{i_p}$ and some $\overline{\alpha}\ \mathsf{IF}_{j_1}, \ldots, \overline{\alpha}\ \mathsf{IF}_{j_q}$, along a witness for F_j of the form $\{i_1, \ldots, i_p\} \,\cup\, \{m+j_1, \ldots, m+j_q\}$. This yields a grammar production $j \to \{\mathsf{Inl}\ i_1, \ldots, \mathsf{Inl}\ i_p\} \,\cup\, \{\mathsf{Inr}\ j_1, \ldots, \mathsf{Inr}\ j_q\}$, where the i_k's are terminals and the j_l's are, like j, nonterminals. The ultimate goal is to eventually obtain reductions of the nonemptiness of $\overline{\alpha}\ \mathsf{IF}_j$ to that of components of $\overline{\alpha}$ alone, i.e., to terminals—this precisely corresponds to derivations in the grammar of terminal sets. It should be intuitively clear that by considering finite derivations we obtain sound witnesses for IF_j. We shall actually prove more: For initial algebras, finite derivations are also witness-complete; for final coalgebras (replacing $\overline{\mathsf{IF}}$ with $\overline{\mathsf{JF}}$), accepting infinite derivations is still sound, and becomes complete.

Theorem 3. Assume that the final coalgebra of $\overline{\mathsf{F}}$ exists and consists of n m-ary functors $\overline{\alpha}_m\ \mathsf{JF}_j$ (as in Section 2.1). Then $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(j)$ is a perfect set for JF_j for $j \in [n]$.

To prove soundness, we define a nonemptiness witness to $\overline{\alpha}\ \mathsf{JF}_j$ corecursively (by abstract $\overline{\mathsf{JF}}$-corecursion). More interestingly: To prove completeness, we define a function to dtree corecursively (by concrete tree corecursion), obtaining a derivation tree, from which we then cut a regular derivation tree via Lemma 1.

Proof sketch: Let $j_0 \in [n]$. We first show that $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(j_0)$ is sound. Let t_0 be a well-formed regular derivation tree with root j_0. We need to prove that F_{j_0} is $\mathsf{Fr}\ t_0$-witnessed. For this, we fix $\overline{\alpha}_m$ such that $\forall i \in \mathsf{Fr}\ t_0.\ \alpha_i \neq \emptyset$, and aim to show that $\overline{\alpha}\ \mathsf{JF}_{j_0} \neq \emptyset$.

For each $j \in \mathsf{Itr}\ t_0$, let t_j be the corresponding subtree of t_0. (It is well-defined, since t_0 is regular.) Note that $t_0 = t_{j_0}$. For each K such that $(j, \mathsf{cp}(K)) \in \mathsf{P}$, since $K \in \mathscr{K}_j$ and $\mathscr{K}_j$ is sound for F_j, we obtain a K-witness for F_j, i.e., a function $w_{j,K} : (\gamma_k)_{k\in K} \to \overline{\gamma}\ \mathsf{F}_j$ (polymorphic in $\overline{\gamma}$).

Let $\overline{\beta}_n$ be defined as $\beta_j = \mathsf{unit}$ if $j \in \mathsf{Itr}\ t_0$ and $= \emptyset$ otherwise. We build a coalgebra structure on $\overline{\beta}$, $(s_j : \beta_j \to (\overline{\alpha},\overline{\beta})\ \mathsf{F}_j)_{j\in[n]}$, as follows: If $j \notin \mathsf{Itr}\ t_0$, s_j is the unique function from $\emptyset$. If $j \in \mathsf{Itr}\ t_0$, then $s_j\ () = w_{j,K}\ (a_i)_{i\in K\cap[m]}\ ()^{|K\cap[m+1,m+n]|}$, where $\mathsf{cp}(K)$ is the right-hand side of the top production of t_j, i.e., $(\mathsf{id} \oplus \mathsf{root}) \bullet \mathsf{cont}\ t_j$. Now, for each $j \in \mathsf{Itr}\ t_0$, $\mathsf{unfold}_j\ \overline{s} : \mathsf{unit} \to \overline{\alpha}\ \mathsf{JF}_j$ ensures the nonemptiness of $\overline{\alpha}\ \mathsf{JF}_j$. In particular, $\overline{\alpha}\ \mathsf{JF}_{j_0} \neq \emptyset$.

We now show that $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(j_0)$ is complete. Let $I \subseteq [m]$ such that JF_{j_0} is I-witnessed. We need to find $I_1 \in \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(j_0)$ such that $I_1 \subseteq I$. Let $\overline{\alpha}_m$ be defined as $\alpha_i = \mathsf{unit}$ if $i \in I$ and $= \emptyset$ otherwise. Let $J = \{j \mid \overline{\alpha}\ \mathsf{F}_j \neq \emptyset\}$. We define $c : J \to ([m] + J)\ \mathsf{fset}$ by $c\ j = \mathsf{cp}(K_j)$, where K_j is such that $(j, \mathsf{cp}(K_j)) \in \mathsf{P}$ and $K_j \subseteq I \cup \{m + j \mid j \in J\}$.

Let now $g : J \to \mathsf{dtree}$ be $\mathsf{unfold}\ \mathsf{id}\ c$. Thus, for all $j \in J$, $\mathsf{root}\ (g\ j) = j$ and $\mathsf{cont}\ (g\ j) = (\mathsf{id} \oplus g) \bullet c\ j = \mathsf{Inl} \bullet (K_j \cap I)\ \cup\ \mathsf{Inr} \bullet \{g\ j \mid m + j \in K_j\}$. Taking $t_0 = g\ j_0$ and using Lemma 1, we obtain the regular well-formed tree t_1 such that $\mathsf{Fr}\ t_1 \subseteq \mathsf{Fr}\ t_0 \subseteq I$. Hence $\mathsf{Fr}\ t_1$ is the desired I_1. $\square$

The above completeness proof provides an example of self-application of codatatypes: A specific codatatype, of infinite derivation trees, figures in the metatheory of general codatatypes. And this may well be unavoidable: While for soundness the regular trees are replaceable by some equivalent (finite) inductive items, it is not clear how completeness could be proved without first appealing to arbitrary infinite derivation trees and then cutting them down to regular trees.

An analogous result holds for initial algebras. For each $i \in \mathsf{N}$, let $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(i)$ be the language generated by i by means of regular finite Gr derivation trees. Since N is finite, these can be described more directly as trees for which every nonterminal path has no repetitions.

Theorem 4. Assume that the initial algebra of $\overline{\mathsf{F}}$ exists and consists of n m-ary functors $\overline{\alpha}_m\ \mathsf{IF}_j$ (as in Section 2.1). Then $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(j)$ is a perfect set for IF_j for $j \in [n]$.

Let us see how Theorems 1–4 can be combined in establishing or refuting non-emptiness for some of our motivating examples from Sections. 1 and 4.1.

- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_list}} = \{\emptyset\}$ by Th. 1; $\mathscr{I}_{\alpha\ \mathsf{list}} = \{\emptyset\}$ by Th. 4
- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_fstream}} = \{\{\alpha,\beta\}\}$; $\mathscr{I}_{\alpha\ \mathsf{fstream}} = \emptyset$ by Th. 4 (i.e. α fstream is empty)
- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_stream}} = \{\{\alpha,\beta\}\}$; $\mathscr{I}_{\alpha\ \mathsf{stream}} = \{\{\alpha\}\}$ by Th. 3
- $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_ltree}} = \{\{\alpha,\beta\},\{\beta,\gamma\}\}$ by Th. 1; $\mathscr{I}_{(\alpha,\beta)\ \mathsf{ltree}} = \{\{\beta\}\}$ by Th. 3
- $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_t_1}} = \{\{\beta\},\{\alpha,\gamma\}\}$, $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_t_2}} = \{\emptyset\}$, and $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_t_3}} = \{\{\alpha\},\{\gamma\}\}$ by Th. 1; $\mathscr{I}_{\mathsf{t}_i} = \{\emptyset\}$ by Th. 4

Since we have maintained perfect sets throughout all the BNF operations, we obtain the following central result:

Theorem 5. Any BNF built from BNFs endowed with perfect sets of witnesses (in particular all basic BNFs discussed in this paper) by repeated applications of the composition, initial algebra, and final coalgebra operations has a perfect set defined as indicated in Theorems 1–4.

Corollary 1. The nonemptiness problem is decidable for arbitrarily nested, mutual (co)datatypes.

Consequently, a procedure implementing Theorems 1–4 will preserve enough nonemptiness witnesses to ensure that all specifications describing nonempty datatypes are accepted. The next subsection presents such a procedure.

4.3 Computational Aspects

Theorem 3 reduces the computation of perfect sets for final coalgebras to that of $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(n)$. Our use of infinite regular trees in the definition of $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(n)$ had the advantage of allowing a simple proof of soundness, and the only natural proof of completeness we could think of, relating the coinductive nature of arbitrary mutual codatatypes with that of infinite trees. However, from the computational point of view, the explicit use of infinite trees is of course excessive.

In fact, already $\mathscr{L}_{\mathsf{Gr}}(n)$ and $\mathscr{L}^{\mathrm{f}}_{\mathsf{Gr}}(n)$, the non-regular versions of the generated languages, are computable by fixpoint iteration on finite sets. Indeed, it is not hard to prove that $\mathscr{L}_{\mathsf{Gr}}$ and $\mathscr{L}^{\mathrm{f}}_{\mathsf{Gr}}$ are the greatest and least solutions of the following fixpoint equation, involving the variable $X : \mathsf{N} \to ((\mathsf{T}+\mathsf{N})\ \mathsf{set})\ \mathsf{set}$, where the order is componentwise inclusion:

$$X\,n \;=\; \Big\{\mathsf{Inl}^{-}ss \,\cup \bigcup_{n' \in \mathsf{Inr}^{-}ss} K_{n'} \;\Big|\; (n,ss) \in \mathsf{P} \,\wedge\, K \in \prod_{n' \in \mathsf{Inr}^{-}ss} X\,n'\Big\}$$

The equation simply states, in our notations, the expected closure under the grammar productions, familiar from classic formal language theory. In our case though, since the "words" are not lists, but finite sets (i.e., elements of $(\mathsf{T}+\mathsf{N})$ set), we have finite convergence of the fixpoint iteration. By analyzing the height of the lattice, we can actually determine that, both for the least and the greatest solutions, the fixpoint iteration stops after at most $\mathsf{card}(\mathsf{N})$ steps.

However, it is easier to settle this computational aspect by working with the regular versions $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(n)$ and $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(n)$, whose structure nicely exhibits boundedness. Namely, we prove for these languages a bounded version of the above fixpoint equation, featuring a decumulator that witnesses the finite convergence of the computation.

First, we relativize the notion of frontier to that of "frontier through *ns*," $\mathsf{Fr}\ ns\ t$, containing the leaves of t accessible by paths of nonterminals from $ns \subseteq \mathsf{N}$. We also define the corresponding *ns*-restricted regularly generated language $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n$.

In what follows, by "word" we mean "finite set of terminals." We can think of a generated word as being more precise than another provided the former is a subword (subset) of the latter. This leads us to defining, for languages (sets of words), the notions of word-inclusion subsumption,[4] $\leq$, by $L \leq L'$ iff $\forall w \in L.\ \exists w' \in L'.\ w' \subseteq w$, and equivalence, $\equiv$, by $L \equiv L'$ iff $L \leq L'$ and $L' \leq L$. It is easy to see that any set $\equiv$-equivalent to a perfect set is again perfect. Note also that Lemma 1 implies $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(n) \equiv \mathscr{L}_{\mathsf{Gr}}(n)$, which qualifies regular trees as a generated-language optimization of arbitrary trees.

We compute $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n$ up to word-inclusion equivalence $\equiv$ by recursively applying available productions whose source nonterminals are in *ns*, removing each time from *ns* the expanded nonterminal. Thus, if n is in *ns*, $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n$ calls $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns'\ n'$ recursively with $ns' = ns \setminus \{n'\}$ for each nonterminal n' in the chosen production from n, and so on, until the current node is no longer in the decumulator *ns*:

Theorem 6. For all $ns \subseteq \mathsf{N}$ and $n \in \mathsf{N}$, $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n \equiv$

$$\begin{cases} \{\emptyset\} & \text{if } n \notin ns \\ \{\mathsf{Inl}^{-}ss \cup \bigcup_{n' \in \mathsf{Inr}^{-}ss} K_{n'} \mid (n, ss) \in \mathsf{P} \wedge K \in \prod_{n' \in \mathsf{Inr}^{-}ss} \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ (ns \setminus \{n\})\ n'\} & \text{otherwise} \end{cases}$$

Proof sketch: $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n \subseteq \{\emptyset\}$, since $\mathsf{Fr}\ ns\ t = \emptyset$ for all t such that $\mathsf{root}\ t = n$. It remains to show $\emptyset \in \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ t$, i.e., to find a derivation tree with root n. In fact, using the assumption that there are no unused nonterminals, we can build a "default derivation tree" $\mathsf{deftr}\ n$ for each n as follows. We pick, for each n, a set $S\ n \in (\mathsf{T} + \mathsf{N})\ \mathsf{fset}$ such that $(n, S\ n) \in \mathsf{P}$. Then we define $\mathsf{deftr} : \mathsf{N} \to \mathsf{dtree}$ corecursively as $\mathsf{deftr} = \mathsf{unfold}\ \mathsf{id}\ S$, i.e., such that $\mathsf{root}\ (\mathsf{deftr}\ n) = n$ and $\mathsf{cont}\ (\mathsf{deftr}\ n) = (\mathsf{id}\ \oplus\ \mathsf{deftr}) \bullet S\ n$. It is easy to prove by KT-coinduction that $\mathsf{deftr}\ n$ is a derivation tree for each n.

Now assume $n \notin ns$, and let $ns' = ns \setminus \{n\}$. For the left-to-right direction, we prove more than $\leq$, namely, actual inclusion between $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n$ and the righthand side. Assume t is a well-formed regular derivation tree of root n. We need to find $ss \in (\mathsf{T} + \mathsf{N})\ \mathsf{fset}$ and $U : \mathsf{Inr}^{-}ss \to \mathsf{dtree}$ such that, for all $n' \in \mathsf{Inr}^{-}ss$, $U\ n'$ is a well-formed regular derivation tree of root n' and $\mathsf{Fr}\ ns\ t = \mathsf{Inl}^{-}ss\ \cup\ \bigcup_{n' \in \mathsf{Inr}^{-}ss}\ \mathsf{Fr}\ ns'\ (U\ n')$. Clearly ss should be the right-hand side of the top production of t. As for U, of course the immediate subtrees of t provide intuitive candidates; however, these do not work, since our goal is to have $\mathsf{Fr}\ ns\ t$ covered by ($\mathsf{Inl}^{-}ss$ in conjunction with) $\mathsf{Fr}\ ns'\ (U\ n')$, while the immediate subtrees only guarantee this property with respect to $\mathsf{Fr}\ ns\ (U\ n')$, i.e., allowing paths to go through n as well. A correct solution is again offered by a corecursive definition: We build the tree t_0 from t by substituting hereditarily each subtree with root n by t. Formally, we take $t_0 = \mathsf{unfold}\ r\ c$, where $r\ t' = \mathsf{root}\ t'$ and $c\ t' = \mathsf{cont}\ t$ if $\mathsf{root}\ t' = n$ and $c\ t' = \mathsf{cont}\ t'$ otherwise. It is easy to prove that t_0, like t, is a regular derivation tree. Thus, we can define U to give, for any n', the corresponding immediate subtree of t_0.

[4] This is in effect the Smyth preorder extension [40] of the subword relation.

To prove the right-to-left direction, let $ss \in (\mathsf{T}+\mathsf{N})$ fset and $K \in \prod_{n' \in \mathsf{Inr}^- ss} \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\, ns'\, n'$ such that $ts = \mathsf{Inl}^- ss \ \cup\ \bigcup_{n' \in \mathsf{Inr}^- ss} K_{n'}$. Unfolding the definition of $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}$, we obtain $U : \mathsf{Inr}^-\, ss \to \mathsf{dtree}$ such that, for all $n' \in \mathsf{Inr}^-\, ss$, $U\, n'$ is a regular derivation tree of root n' such that $K_{n'} \in \mathsf{Fr}\, ns'\, (U\, n')$. Then the tree of immediate leafs $\mathsf{Inl}^- ss$ and immediate subtrees $\{U\, n' \mid n' \in \mathsf{Inr}^-\, ss\}$, namely, $\mathsf{Node}\, n\, ((\mathsf{id} \oplus U) \bullet ss)$, is the desired regular derivation tree whose frontier is included ts. ☐

Theorem 6 provides an alternative, recursive definition of $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\, ns\, n$. The definition terminates because the argument ns is finite and decreases strictly in the recursive case—in fact, this shows that the height of the recursive call stack is bounded by the number of nonterminals, which for our application translates to the number of simultaneously introduced codatatypes.

Here is how the above recursion operates on the ltree example. We have $\mathsf{T} = \{\alpha, \beta\}$, $\mathsf{N} = \{\gamma\}$, and $\mathsf{P} = \{p_1, p_2\}$, where $p_1 = (\gamma, \{\mathsf{Inl}\,\alpha, \mathsf{Inl}\,\beta\})$ and $p_2 = (\gamma, \{\mathsf{Inl}\,\beta, \mathsf{Inr}\,\gamma\})$. Note that

- $\mathsf{Inl}^- ss = \{\alpha, \beta\}$ and $\mathsf{Inr}^- ss = \emptyset$ for $(n, ss) = p_1$
- $\mathsf{Inl}^- ss = \{\beta\}$ and $\mathsf{Inr}^- ss = \{\gamma\}$ for $(n, ss) = p_2$

The computation has one single recursive call, yielding

$$
\begin{aligned}
\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\, \gamma = \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\, \{\gamma\}\, \gamma \\
&\equiv \{\{\alpha,\beta\} \cup \emptyset\} \cup \{\{\beta\} \cup \textstyle\bigcup_{n' \in \{\gamma\}} K_{n'} \mid K \in \prod_{n' \in \{\gamma\}} \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\, \emptyset\, n'\} \\
&= \{\{\alpha,\beta\}\} \cup \{\{\beta\} \cup K_\gamma \mid K_\gamma \in \mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\, \emptyset\, \gamma\} \\
&= \{\{\alpha,\beta\}\} \cup \{\{\beta\} \cup \emptyset\} \\
&= \{\{\alpha,\beta\}, \{\beta\}\} \\
&\equiv \{\{\beta\}\}
\end{aligned}
$$

For datatypes, the computation of $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}$ is achieved analogously to Theorem 6, defining $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\, ns\, n$ as a generalization of $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\, n$.

Theorem 7. The statement of Theorem 6 still holds if we substitute $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}$ for $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}$ and $\emptyset$ for $\{\emptyset\}$.

5 Implementation in Isabelle

The package maintains nonemptiness information to be prepared for producing nonemptiness proofs upon datatype definitions. The equations from Theorems 6 and 7 involve only executable operations over finite sets of numbers, sums, and products. Since the descriptions of Theorems 1 and 2 are also executable, the implementation task emerges clearly: Store a perfect set along each basic BNF and have each BNF operation compute witnesses from those of its operands.

However, as it stands, I-witnesshood cannot be expressed in HOL because types are always nonempty: How can we state that $(\alpha, \beta)\,\mathsf{tree} \neq \emptyset$ conditionally on $\alpha \neq \emptyset$ or $\beta \neq \emptyset$, in the context of α and β being assumed nonempty in the first place? The solution is to work not with operators $\overline{\alpha}\,\mathsf{F}$ on HOL types directly but rather with their *internalization* to sets, expressed as a polymorphic function $\mathsf{Fin} : \alpha_1\ \mathsf{set} \to \cdots \to \alpha_n\ \mathsf{set} \to (\overline{\alpha}\ \mathsf{F})\ \mathsf{set}$ defined as $\mathsf{Fin}\ \overline{A} = \{x \mid \forall i \in [n].\ \mathsf{Fset}^i\ x \subseteq A_i\}$. I-witnesshood becomes $(\forall i \in I.\ A_i \neq \emptyset) \Rightarrow \mathsf{Fin}\ \overline{A} \neq \emptyset$.

For each n-ary BNF F, the package stores a set of sets $\mathscr{I}$ of numbers in $[n]$ (the perfect set) and, for each set $I \in \mathscr{I}$, a polymorphic constant $w_I : (\alpha_i)_{i\in I} \to \overline{\alpha}\ \mathsf{F}$ and an equivalent formulation of I-witnesshood: $\forall i \in I.\ \mathsf{Fset}^i\ (w_I\ (a_j)_{j\in I}) \neq \emptyset$.

Due to the logic's restricted expressiveness, we cannot prove the theorems presented in this paper in their most general form for arbitrary functors and have the package instantiate them for specific functors. Instead, the package proves the theorems dynamically for the specific functors involved in the datatype definitions. Only the soundness part of the theorems is needed. To paraphrase Krauss and Nipkow [26], completeness belongs to the realm of metatheory and is not required to obtain actual nonemptiness proofs—it does let you sleep better though, by ensuring that the employed criterion is as precise as it can be.

A HOL definitional package bears the burden of *computing* terms and *certifying* the computation, i.e., ensuring that certain terms are theorems. The combinatorial computation of witnessing sets of indices described in Theorems 6 and 7 would be expensive if performed through Isabelle, that is, by executing the equations stated in these theorems as term rewriting in the logic. Instead, we perform the computation outside the logic, employing an ML datatype aimed at representing efficiently the finite and the regular derivation trees dwelling the Isabelle type dtree from Section 3:

```
datatype wit_tree = Wit_Leaf of int
                  | Wit_Node of (int * int * int list) * wit_tree list
```

Here, Wit_Node $((i, j, is), ts)$ stores the root nonterminal i, a numeric identifier of the used production j, and the continuation consisting of the terminals is and the further non-terminal expanded trees ts. Moreover, Wit_Leaf i stores, for the case of regular infinite trees, the nonterminal where a regularity loop occurs, i.e., such that it has a previous occurrence on the path to the root.

From the ML trees, we produce witnesses represented as Isabelle constants of appropriate types (the w_I's described above), by essentially mimicking the (co)recursive definitions employed in the proofs of the soundness parts of Theorems 3 and 4 from Section 4.2. Then, we certify the witnesses by producing the relevant Isabelle proof goals and discharging them by mirroring the corresponding (co)inductive arguments from the aforementioned proofs. In summary: The witnesses are computed outside the logic, but they are verified by Isabelle's kernel. After introducing any BNF, the redundant witnesses are removed.

The development devoted to the production and certification of witnesses amounts to about 1000 lines of Standard ML [12].

6 Related Work

Coinductive (or coalgebraic) datatypes have become popular in recent years in the study of infinite behaviors and nonterminating computation. Whereas inductive datatypes are well studied and widely available in most programming languages and proof assistants, coinductive types are still not mainstream and pose great challenges to be integrated into current systems.

Much research has appeared in the last years, in the context of theorem proving, on how to add coinductive types or improve support of coinductive proofs, including developments in Coq [8, 31], Agda [4], and CIRC [29]. The work of this paper is in line with this research. Our results are applicable to all proof assistants from the HOL family, which together cover a large subset of the proof assistant community. We therefore aimed at an abstract presentation in terms of higher-order logic—only Section 5 is specific to Isabelle/HOL.

Other definitional packages must also prove nonemptiness of newly defined types, but typically the proofs are easy. For example, Homeier's quotient package for HOL4 [22] exploits the observation that quotients of nonempty sets are nonempty, and Huffman's (co)recursive domain package for Isabelle/HOLCF [24] can rely on a minimal element $\perp$. For the traditional datatype packages introduced by Melham [30], and implemented in Isabelle/HOL by Berghofer and Wenzel [7], proving nonemptiness is nontrivial, but by reducing nested definitions to mutual definitions, they could employ a standard reachability analysis [7, § 4.1]. To our knowledge, the completeness of the analysis has not been proved (or even formulated) for previous datatype packages.

Obviously, our overall approach to (co)datatypes is heavily inspired by category-theory developments [6, 16, 20, 21, 37]—this is discussed in detail in a previous paper [14], which puts forward a program (continued here, as well as in [10, 11, 13]) for integrating insight from category theory in proof assistants based on higher-order logic, in order to achieve better structure and functionality. A similar program (of a somewhat larger scale) is pursed in the context of homotopy type theory [2], targeting proof assistants based on type theory, notably Coq and Agda. Our nonemptiness-witness maintenance is similar to the preservation of enriched types along various constructions, e.g., initial algebras and final coalgebras of pointed functors are also pointed [23]; however, existing analysis techniques are only concerned with soundness (not completeness) results.

7 Conclusion

We presented a complete solution to the nonemptiness problem for open-ended, mutual, nested codatatypes. This problem arose in the context of Isabelle's new (co)datatype package and has broad practical motivation in terms of the popularity of HOL-based provers. The problem and its solution also enjoy an elegant metatheory, which itself is best expressed in terms of codatatypes. Our solution, like the rest of the definitional package, is part of the latest edition of Isabelle.

Acknowledgment. Tobias Nipkow made this work possible. Andreas Lochbihler suggested many improvements, notably concerning the format of concrete coinduction principles. Brian Huffman suggested major conceptual simplifications to the package. Florian Haftmann, Christian Urban, and Makarius Wenzel guided us in the jungle of package writing. Stefan Milius and Lutz Schröder found an elegant proof to eliminate one of the BNF cardinality assumptions. Andreas Abel and Martin Hofmann pointed out relevant work. Mark Summerfield and several anonymous reviewers commented on earlier versions of this paper.

Blanchette is supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant Ni 491/14-1). Popescu is supported by the project Security Type Systems and Deduction (grant Ni 491/13-2) as part of the DFG program Reliably Secure Software Systems (RS^3, Priority Program 1496). Traytel is supported by the DFG program Program and Model Analysis (PUMA, doctorate program 1480). The authors are listed in alphabetical order, regardless of individual contributions or seniority.

References

[1] The HOL Zero Theorem prover (2014), `http://proof-technologies.com/holzero`
[2] The Homotopy Type Theory homepage (2014), `http://homotopytypetheory.org`
[3] Abel, A., Altenkirch, T.: A predicative strong normalisation proof for a lambda-calculus with interleaving inductive types. In: Coquand, T., Dybjer, P., Nordström, B., Smith, J.M. (eds.) TYPES '99. LNCS, vol. 1956, pp. 21–40. Springer (2000)
[4] Abel, A., Pientka, B., Thibodeau, D., Setzer, A.: Copatterns: programming infinite structures by observations. In: POPL. pp. 27–38 (2013)
[5] Arthan, R.D.: Some mathematical case studies in ProofPower–HOL. In: Slind, K. (ed.) TPHOLs 2004 (Emerging Trends). pp. 1–16 (2004)
[6] Barr, M.: Terminal coalgebras in well-founded set theory. Theor. Comput. Sci. 114(2), 299–315 (1993)
[7] Berghofer, S., Wenzel, M.: Inductive datatypes in HOL—Lessons learned in formal-logic engineering. In: Bertot, Y., Dowek, G., Hirschowitz, A., Paulin, C., Théry, L. (eds.) TPHOLs '99. LNCS, vol. 1690, pp. 19–36 (1999)
[8] Bertot, Y.: Filters on coinductive streams, an application to Eratosthenes' sieve. In: TLCA '05. pp. 102–115 (2005)
[9] Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development—Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)
[10] Blanchette, J.C., Hölzl, J., Lochbihler, A., Panny, L., Popescu, A., Traytel, D.: Truly modular (co)datatypes for Isabelle/HOL. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 93–110. Springer (2014)
[11] Blanchette, J.C., Popescu, A., Traytel, D.: Foundational extensible corecursion: A proof assistant perspective. Submitted to ESOP. Draft available at `http://www.eis.mdx.ac.uk/staffpages/andreipopescu/pdf/fouco.pdf`
[12] Blanchette, J.C., Popescu, A., Traytel, D.: Supplementary material associated with this paper. `https://github.com/dtraytel/Witnessing-Codatatypes`
[13] Blanchette, J.C., Popescu, A., Traytel, D.: Unified classical logic completeness - A coinductive pearl. In: Demri, S., Kapur, D., Weidenbach, C. (eds.) IJCAR 2014. vol. 8562, pp. 46–60 (2014)
[14] Blanchette, J.C., Traytel, D., Popescu, A.: Foundational, compositional (co)datatypes for higher-order logic—Category theory applied to theorem proving. In: LICS 2012, pp. 596–605. IEEE (2012)
[15] Bove, A., Dybjer, P.: Dependent types at work. In: LerNet ALFA Summer School. pp. 57–99 (2008)
[16] Ghani, N., Johann, P., Fumex, C.: Generic fibrational induction. Logical Methods in Computer Science 8(2) (2012)

[17] Gordon, M.J.C., Melham, T.F. (eds.): Introduction to HOL: A Theorem Proving Environment for Higher Order Logic. Cambridge University Press (1993)
[18] Gvero, T., Kuncak, V., Piskac, R.: Interactive synthesis of code snippets. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 418–423. Springer (2011)
[19] Harrison, J.: HOL Light: A tutorial introduction. In: FMCAD '96. LNCS, vol. 1166, pp. 265–269. Springer (1996)
[20] Hasegawa, R.: Two applications of analytic functors. Theor. Comput. Sci. 272(1-2), 113–175 (2002)
[21] Hermida, C., Jacobs, B.: Structural induction and coinduction in a fibrational setting. Inf. Comput. 145(2), 107–152 (1998)
[22] Homeier, P.V.: A design structure for higher order quotients. In: Hurd, J., Melham, T.F. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 130–146. Springer (2005)
[23] Howard, B.T.: Inductive, coinductive, and pointed types. In: ICFP. pp. 102–109 (1996)
[24] Huffman, B.: A purely definitional universal domain. In: Berghofer, S., Nipkow, T., Urban, C., Wenzel, M. (eds.) TPHOLs 2009. LNCS, vol. 5674, pp. 260–275. Springer (2009)
[25] The Isabelle theorem prover (2014), http://isabelle.in.tum.de/
[26] Krauss, A., Nipkow, T.: Proof pearl: Regular expression equivalence and relation algebra. J. Autom. Reasoning 49(1), 95–106 (2012)
[27] Lenisa, M., Power, J., Watanabe, H.: Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. Electr. Notes Theor. Comput. Sci. 33, 230–260 (2000)
[28] Lochbihler, A.: Java and the Java memory model—A unified, machine-checked formalisation. In: Seidl, H. (ed.) ESOP 2012. LNCS, vol. 7211, pp. 497–517. Springer (2012)
[29] Lucanu, D., Goriac, E.I., Caltais, G., Roşu, G.: CIRC: A behavioral verification tool based on circular coinduction. In: CALCO'09. pp. 433–442 (2009)
[30] Melham, T.F.: Automating recursive type definitions in higher order logic. In: Birtwistle, G., Subrahmanyam, P.A. (eds.) Current Trends in Hardware Verification and Automated Theorem Proving, pp. 341–386. Springer (1989)
[31] Nakata, K., Uustalu, T., Bezem, M.: A proof pearl with the fan theorem and bar induction—Walking through infinite trees with mixed induction and coinduction. In: Yang, H. (ed.) APLAS '11. LNCS, vol. 7078, pp. 353–368. Springer (2011)
[32] Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: A Proof Assistant for Higher-Order Logic, LNCS, vol. 2283. Springer (2002)
[33] Paulson, L.C.: A formulation of the simple theory of types (for Isabelle). In: Conference on Computer Logic. pp. 246–274 (1988)
[34] Paulson, L.C.: A fixedpoint approach to (co)inductive and (co)datatype definitions. In: Plotkin, G.D., Stirling, C., Tofte, M. (eds.) Proof, Language, and Interaction—Essays in Honour of Robin Milner, pp. 187–212. MIT Press (2000)
[35] Pierce, B.C.: Types and Programming Languages. MIT Press (2002)
[36] Rutten, J.J.M.M.: Relators and metric bisimulations. Electr. Notes Theor. Comput. Sci. 11, 252–258 (1998)
[37] Rutten, J.J.M.M.: Universal coalgebra: A theory of systems. Theor. Comput. Sci. 249, 3–80 (2000)
[38] Schropp, A., Popescu, A.: Nonfree datatypes in Isabelle/HOL: Animating a many-sorted metatheory. In: CPP 2013. pp. 114–130 (2013)
[39] Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) TPHOLs 2008. LNCS, vol. 5170, pp. 28–32. Springer (2008)
[40] Smyth, M.B.: Power domains. J. Comput. Syst. Sci. 16(1), 23–36 (1978)

APPENDIX

Here we give more details concerning the concepts discussed in the paper, including proofs.

A Structural (Co)induction

Using the atomic infrastructure described in Section 2.2, the induction principle can be expressed abstractly for the mutual initial algebra $\overline{\mathsf{IF}}$ of functors $\overline{\mathsf{F}}$ as follows for sets $\overline{\alpha}$ and predicates $\varphi_j : \overline{\alpha}\ \mathsf{IF}_j \to \mathsf{bool}$:

$$\frac{\bigwedge_{j=1}^{n} \forall x \in (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{IF}})\ \mathsf{F}_j.\ (\bigwedge_{k=1}^{n} \forall b \in \mathsf{Fset}_j^{m+k} x.\ \varphi_k\, b) \Rightarrow \varphi_j\, (\mathsf{ctor}_j\, x)}{\bigwedge_{j=1}^{n} \forall b \in \overline{\alpha}\ \mathsf{IF}_j.\ \varphi_j\, b}$$

For lists, this instantiates to

$$\frac{\forall x \in \mathsf{unit} + \alpha \times \alpha\ \mathsf{list}.\ (\forall b \in \mathsf{Fset}^2 x.\ \varphi\, b) \Rightarrow \varphi\, (\mathsf{ctor}\ x)}{\forall b \in \alpha\ \mathsf{list}.\ \varphi\, b}$$

which, by taking $\mathsf{Nil} = \mathsf{ctor}\,(\mathsf{Inl}\ ())$ and $\mathsf{Cons}\ a\ b = \mathsf{ctor}\,(\mathsf{Inr}\ (a,b))$, can be recast into the familiar rule

$$\frac{\varphi\ \mathsf{Nil} \qquad \forall a \in \alpha.\ \forall b \in \alpha\ \mathsf{list}.\ \varphi\, b \Rightarrow \varphi\, (\mathsf{Cons}\ a\ b)}{\forall b \in \alpha\ \mathsf{list}.\ \varphi\, b}$$

Moving to coinduction, we need a further well-known assumption: that our functors preserve weak pullbacks, or, equivalently, that they induce relators [36]. For a functor $\overline{\alpha}_n\,\mathsf{F}$, we lift its action $\mathsf{Fmap} : (\alpha_1 \to \beta_1) \to \cdots \to (\alpha_n \to \beta_n) \to \overline{\alpha}\ \mathsf{F} \to \overline{\beta}\ \mathsf{F}$ on functions to an action $\mathsf{Frel} : (\alpha_1 \to \beta_1 \to \mathsf{bool}) \to \cdots \to (\alpha_n \to \beta_n \to \mathsf{bool}) \to (\overline{\alpha}\ \mathsf{F} \to \overline{\beta}\ \mathsf{F} \to \mathsf{bool})$, the *relator*, defined as follows:

$$\begin{gathered}\mathsf{Frel}\ \overline{\varphi}\ x\ y \iff \exists z.\ \mathsf{Fmap}\ \overline{\mathsf{fst}}\ z = x \wedge \mathsf{Fmap}\ \overline{\mathsf{snd}}\ z = y \wedge \\ \bigwedge_{i=1}^{n} \forall (a,b) \in \mathsf{Fset}^i\ z.\ \varphi_i\ a\ b\end{gathered}$$

With these preparations, structural coinduction can also be expressed abstractly, for the mutual final coalgebra $\overline{\mathsf{JF}}$ of functors $\overline{\mathsf{F}}$:

$$\frac{\bigwedge_{j=1}^{n} \forall a\, b \in (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{JF}})\ \mathsf{F}_j.\ \theta_j\, a\, b \Rightarrow \mathsf{Frel}_j\, (=)^m\, \overline{\theta}\, (\mathsf{dtor}_j\, a)\, (\mathsf{dtor}_j\, b)}{\bigwedge_{j=1}^{n} \forall a\, b.\ \theta_j\ a\ b \Rightarrow a = b}$$

for sets $\overline{\alpha}_n$ and binary predicates $\theta_j \in \overline{\alpha}\ \mathsf{JF}_j \to \overline{\alpha}\ \mathsf{JF}_j \to \mathsf{bool}$. The rule is parameterized by predicates $\theta_j : \overline{\alpha}\ \mathsf{JF}_j \to \overline{\alpha}\ \mathsf{JF}_j \to \mathsf{bool}$ required by the antecedent to form an $\overline{\mathsf{F}}$-bisimulation. The principle effectively states that the equality forms the largest $\overline{\mathsf{F}}$-bisimulation [37].

B More Details on the Customization Process: From Abstract to Concrete

Customizing Coiteration. The abstract coiteration principle described in Section 2.1 relies on a coiterator $\mathsf{unfold} : (\beta \to \beta\ \mathsf{pre\_dtree}) \to \beta \to \mathsf{dtree}$ such that $\mathsf{dtor} \circ \mathsf{unfold}\ s = \mathsf{map\_pre\_dtree}\ (\mathsf{unfold}\ s) \circ s$. Writing s as $\langle r, c\rangle$ for $r : \beta \to \mathsf{N}$ and $c : \beta \to (\mathsf{T} + \alpha)\ \mathsf{fset}$ and recasting the equation in pointful form yields $\mathsf{dtor}\ (\mathsf{unfold}\ \langle r, c\rangle\ b) = \mathsf{map\_pre\_dtree}\ (\mathsf{unfold}\ s)\ (r\, b, c\, b)$ This can be further improved by unfolding the definition of $\mathsf{map\_pre\_dtree}$, expressing dtor as $\langle \mathsf{root}, \mathsf{cont}\rangle$, and splitting the result into a pair of equations: $\mathsf{root}\ (\mathsf{unfold}\ \langle r, c\rangle\ b) = r\ b$ and $\mathsf{cont}\ (\mathsf{unfold}\ \langle r, c\rangle\ b) = (\mathsf{id} \oplus \mathsf{unfold}\ \langle r, c\rangle) \bullet c\ b$. The coiteration rule of Section 2.1 emerges by replacing unfold with the curried $\mathsf{unfold}' : (\beta \to \mathsf{N}) \to (\beta \to (\mathsf{T} + \beta)\ \mathsf{fset}) \to \beta \to \mathsf{dtree}$ defined as $\mathsf{unfold}'\ r\ c = \mathsf{unfold}\ \langle r, c\rangle$.

Custoimizing Coinduction. The abstract coinduction principle of Appendix A is customized into the following concrete coinduction for dtree:

$$\frac{\begin{array}{l}\forall t_1\, t_2.\ \theta\, t_1\, t_2 \Rightarrow \mathsf{root}\ t_1 = \mathsf{root}\ t_2\ \wedge \\ \qquad\qquad\qquad \mathsf{fset\_rel}\ (\mathsf{sum\_rel}\ (=)\ \theta)\ (\mathsf{cont}\ t_1)\ (\mathsf{cont}\ t_2)\end{array}}{\theta\, t_1\, t_2 \Rightarrow t_1 = t_2}$$

where the predicate $\mathsf{fset\_rel}\ (\mathsf{sum\_rel}\ (=)\ \theta)$ is an instance of the abstract Frel: it gives the componentwise extension of θ to $(\mathsf{T} + \mathsf{dtree})\ \mathsf{fset}$. Unfolding the characteristic theorems for fset_rel and sum_rel yields the antecedent

$$\begin{array}{rl}\forall t_1\, t_2.\ \theta\, t_1\, t_2 \Rightarrow & \mathsf{root}\ t_1 = \mathsf{root}\ t_2\ \wedge \\ & \mathsf{Inl}^-(\mathsf{cont}\ t_1) = \mathsf{Inl}^-(\mathsf{cont}\ t_2)\ \wedge \\ & \forall t_1' \in \mathsf{Inr}^-(\mathsf{cont}\ t_1).\ \exists t_2' \in \mathsf{Inr}^-(\mathsf{cont}\ t_2).\ \theta\, t_1'\, t_2'\ \wedge \\ & \forall t_2' \in \mathsf{Inr}^-(\mathsf{cont}\ t_2).\ \exists t_1' \in \mathsf{Inr}^-(\mathsf{cont}\ t_1).\ \theta\, t_1'\, t_2'\end{array}$$

where $\mathsf{Inl}^-(\mathsf{cont}\ t)$ is the set of t's successor leaves and $\mathsf{Inr}^-(\mathsf{cont}\ t)$ is the set of its immediate subtrees. Informally:

> If two trees are in relation θ, then they have the same root and the same successor leaves and for each immediate subtree of one, there exists an immediate subtree of the other in relation θ with it.

C More on the Coinductive Tree Case Study

Next we define the notions of interior and frontier of a tree. $\mathsf{Itr} : \mathsf{dtree} \to \mathsf{N}\ \mathsf{set}$ is defined inductively as follows:

$\mathsf{root}\ t \in \mathsf{Itr}\ t$
$\mathsf{Inr}\ t_1 \in \mathsf{cont}\ t \wedge n \in \mathsf{Itr}\ t_1\ \Rightarrow\ n \in \mathsf{Itr}\ t$

$\mathsf{Fr} : \mathsf{dtree} \to \mathsf{N}\ \mathsf{set}$ is defined inductively as follows:

$\mathsf{Inl}\ t \in \mathsf{cont}\ t \Rightarrow t \in \mathsf{Fr}\ t$
$\mathsf{Inr}\ t_1 \in \mathsf{cont}\ t \wedge t \in \mathsf{Fr}\ t_1 \Rightarrow t \in \mathsf{Fr}\ t$

The parameterized versions of interior and frontier (as required in Section 4.3), $\mathsf{Itr} : \mathsf{N}\ \mathsf{set} \to \mathsf{dtree} \to \mathsf{N}\ \mathsf{set}$ and $\mathsf{Fr} : \mathsf{N}\ \mathsf{set} \to \mathsf{dtree} \to \mathsf{N}\ \mathsf{set}$, are defined inductively modifying the above clauses textually:

- 'Itr' is replaced by '$\mathsf{Itr}\ ns$' and 'Fr' is replaced by '$\mathsf{Fr}\ ns$';
- the hypothesis $\mathsf{root}\ t \in ns$ is added.

The notion of a finite tree, $\mathsf{ftree} : \mathsf{dtree} \to \mathsf{bool}$, is also defined inductively:

$\mathsf{Inr} \bullet \mathsf{cont}\ t = \emptyset \Rightarrow \mathsf{ftree}\ t$
$(\forall t_1 \in \mathsf{Inr}^{-}\ t.\ \mathsf{ftree}\ t_1) \Rightarrow \mathsf{ftree}\ t$

D Inductive Trees

The datatype definition

$\texttt{datatype}\ \ \mathsf{fdtree} = \mathsf{FNode}\ (\mathsf{froot} : \mathsf{N})\ (\mathsf{fcont} : (\mathsf{T} + \mathsf{dtree})\ \mathsf{fset})$

(introducing finite trees) produces the operations FNode, froot, and fcont having constructor and selector properties corresponding precisely to the ones of Node, root and cont from the codatatype dtree in Section 3. The difference concerns induction and recursion.

Iteration. The general principle described in Section 2.1 employs in this unary case a iterator fold of (polymorphic) type $(\beta\ \mathsf{pre\_fdtree} \to \beta) \to \mathsf{fdtree} \to \beta$, for which it yields

$$\forall s : \beta\ \mathsf{pre\_fdtree} \to \beta.\ (\mathsf{fold}\ s) \circ \mathsf{ctor} = s \circ \mathsf{map\_pre\_fdtree}\ (\mathsf{fold}\ s)$$

that is,

$$\forall s : \beta\ \mathsf{pre\_fdtree} \to \beta.\ \forall k.\ \mathsf{fold}\ s\ (\mathsf{ctor}\ k) = s\ (\mathsf{map\_pre\_fdtree}\ (\mathsf{fold}\ s)\ k)$$

The fdtree-defining BNF coincides with the dtree-defining BNF: $\beta\ \mathsf{pre\_fdtree} = \mathsf{N} \times (\mathsf{T} + \beta)\ \mathsf{fset}$ and $\mathsf{map\_pre\_fdtree}\ f = \mathsf{id} \otimes (\mathsf{image}\ (\mathsf{id} \oplus f))$.

As in the codatatype case, the above characterization needs some customization. Using the FNode instead of ctor and unfolding the definition of map_pre_fdtree, we obtain

$$\begin{aligned}&\forall s : \mathsf{N} \times (\mathsf{T} + \beta)\ \mathsf{fset} \to \beta.\ \forall n\ as.\\ &\quad \mathsf{fold}\ s\ (\mathsf{FNode}\ n\ as) = s\ (\mathsf{map\_pre\_fdtree}\ (\mathsf{fold}\ s)\ (n, as))\end{aligned}$$

By unfolding the definition of map_pre_fdtree, we obtain

$$\begin{aligned}&\forall s : \mathsf{N} \times (\mathsf{T} + \beta)\ \mathsf{fset} \to \beta.\ \forall n\ as.\\ &\quad \mathsf{fold}\ s\ (\mathsf{FNode}\ n\ as) = s\ (n, (\mathsf{id} \oplus \mathsf{fold}\ s) \bullet as)\end{aligned}$$

Finally, replacing fold with its more convenient curried version $\mathsf{fold}' : (\mathsf{N} \to (\mathsf{T} + \beta)\ \mathsf{fset} \to \beta) \to \mathsf{fdtree} \to \beta$ defined as $\mathsf{fold}'\ s = \mathsf{fold}\ (\lambda(n, as).\ s\ n\ as)$, we obtain the following customized iteration principle, where we write fold instead of fold$'$: For all sets β, functions $s : \mathsf{N} \to (\mathsf{T} + \beta)\ \mathsf{fset} \to \beta$ and elements $n \in \mathsf{N}$ and $as \in (\mathsf{T} + \mathsf{fdtree})\ \mathsf{fset}$, it holds that $\mathsf{fold}\ s\ (\mathsf{FNode}\ n\ as) = s\ n\ ((\mathsf{id} \oplus \mathsf{fold}\ s) \bullet as)$.

Induction. The induction principle from Section A yields for $\varphi : \alpha\ \mathsf{fdtree} \to \mathsf{bool}$

$$\frac{\forall k \in \alpha\ \mathsf{pre\_fdtree}.\ (\forall t \in \mathsf{Fset}\ k.\ \varphi\ t) \Rightarrow \varphi\ (\mathsf{ctor}\ k)}{\forall t \in \alpha\ \mathsf{fdtree}.\ \varphi\ t}$$

i.e., using the curried variation FNode of dtor,

$$\frac{\forall n\ as.\ (\forall t \in \mathsf{Fset}\ (n,\ as).\ \varphi\ t) \Rightarrow \varphi\ (\mathsf{FNode}\ n\ as)}{\forall t \in \alpha\ \mathsf{fdtree}.\ \varphi\ t}$$

Unfolding the definition of Fset, namely, $\mathsf{Fset}\ (n, as) = \mathsf{Inr}^{-}\ as$, we obtain the end-product customized induction for finite trees:

$$\frac{\forall n\ as.\ (\forall t \in \mathsf{Inr}^{-}\ as.\ \varphi\ t) \Rightarrow \varphi\ (\mathsf{FNode}\ n\ as)}{\forall t \in \alpha\ \mathsf{fdtree}.\ \varphi\ t}$$

E Proofs

For more details on some of the proofs, we refer the reader to our Isabelle formalization [12], which employs essentially the same notations as this text.

Proof of Lemma 1: $\mathsf{H}\ n$ is regular by construction: if a subtree of it has root n', then it is equal to $\mathsf{H}\ n'$. The frontier inclusion $\mathsf{Fr}\ (\mathsf{H}\ n) \subseteq \mathsf{Fr}\ t_0$ follows by routine fixpoint induction on the definition of Fr (since at each node $n' \in \mathsf{Itr}\ (\mathsf{H}\ n)$ we only have the immediate leaves of $\mathsf{pick}\ n'$, which is a subtree of $\mathsf{Fr}\ t_0$). Finally, assume that t_0 is well-formed. Then the fact that $\mathsf{H}\ n$ is well-formed follows by routine fixpoint coinduction on the definition of wf (since, again, at each $n' \in \mathsf{Itr}\ (\mathsf{H}\ n)$ we have the production of $\mathsf{pick}\ n'$). ☐

In some of the following proofs we exploit an embedding of datatypes as finite codatatypes. Using this embedding, we can transfer the recursive definition and structural induction principles from $\overline{\mathsf{IF}}$ to finite elements of $\overline{\mathsf{JF}}$, and in particular from fdtree to finite trees in dtree.

The regular cut of a tree works well with respect to the codatatype metatheory, but for datatypes it has the disadvantage that it may produce infinite trees out of finite ones (cf. Fig. 3, left and middle). We need a slightly different concept for datatypes: the finite regular cut. Let t_0 be a finite derivation tree. We choose the function $\mathsf{fpick} : \mathsf{Itr}\ t_0 \to \mathsf{Subtr}\ t_0$ similarly to pick from Section 3, but making sure that in addition the choice of the subtrees $\mathsf{fpick}\ n$ is minimal, in that $\mathsf{fpick}\ n$ does not have n in the interior of a proper subtree (and hence does not have any proper subtree of root n)—such a choice is possible thanks to the finiteness of t_0. We define the finite regular cut of t_0, $\mathsf{rfcut}\ t_0$, just like $\mathsf{rcut}\ t_0$ but using fpick instead of pick. Now we can prove:

Lemma 2. Assume t_0 is a finite derivation tree. Then:

(1) The statement of Lemma 1 holds if we replace rcut by rfcut.
(2) $\mathsf{rfcut}\ t_0$ is finite.

Proof: (1) Similar to the proof of Lemma 1. (2) By routine induction on t_0. $\square$

More detailed proof of Theorem 4: Let $j_0 \in [n]$. We first show that $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(j_0)$ is sound. Let t_0 be a well-formed finite regular derivation tree with root j_0. We need to prove that F_{j_0} is $\mathsf{Fr}\ t_0$-witnessed. For this, we fix $\overline{\alpha}_m$ such that $\forall i \in \mathsf{Fr}\ t_0.\ \alpha_i \neq \emptyset$, and aim to show that $\overline{\alpha}\ \mathsf{IF}_{j_0} \neq \emptyset$.

For each $j \in \mathsf{Itr}\ t_0$, let t_j be the corresponding subtree of t_0. (It is well-defined, since t_0 is regular.) Note that $t_0 = t_{j_0}$. For each K such that $(j, \mathsf{cp}(K)) \in \mathsf{P}$, since $K \in \mathscr{K}_j$ and $\mathscr{K}_j$ is sound for F_j, we obtain a K-witness for F_j, i.e., a function $w_{j,K} : (\alpha_k)_{k \in K} \to \overline{\alpha}\ \mathsf{F}_j$.

We verify the following fact by induction on the finite derivation tree t: If $\exists j \in \mathsf{Itr}\ t_0.\ t = t_j$, then $\overline{\alpha}\ \mathsf{IF}_j \neq \emptyset$. The induction step goes as follows: Assume $t = t_j$ has the form $\mathsf{Node}\ j\ as$, and let J be the set of all roots of the immediate subtrees of t, namely, $\mathsf{root} \bullet (\mathsf{Inr}^- (\mathsf{cont}\ t))$. By the induction hypothesis, $\overline{\alpha}\ \mathsf{IF}_{j'} \neq \emptyset$ (say, $b_{j'} \in \overline{\alpha}\ \mathsf{IF}_{j'}$) for all $j' \in J$. Then $w_{j,K}\ (a_i)_{i \in \mathsf{Inl}^- t}\ (b_{j'})_{j' \in J} \in \overline{\alpha}\ \mathsf{IF}_j$, making $\overline{\alpha}\ \mathsf{IF}_j$ nonempty. In particular, $\overline{\alpha}\ \mathsf{JF}_{j_0} \neq \emptyset$.

We now show that $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(j_0)$ is complete. Let $I \subseteq [m]$ such that IF_{j_0} is I-witnessed. We need to find $I_1 \in \mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(j_0)$ such that $I_1 \subseteq I$.

Let $\overline{\alpha}_m$ be defined as $\alpha_i = \mathsf{unit}$ if $i \in I$ and $= \emptyset$ otherwise. We verify, by structural $\overline{\mathsf{IF}}$-induction on b, that for all $j \in [n]$ and $b \in \overline{\alpha}\ \mathsf{IF}_j$, there exists a finite well-formed derivation tree t such that $\mathsf{root}\ t = j$ and $\mathsf{Fr}\ t \subseteq I$. For the inductive step, assume $\mathsf{ctor}_j\ x \in \overline{\alpha}\ \mathsf{IF}_j$, where $x \in (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{IF}})\ \mathsf{F}_j$. By the induction hypotheses, we obtain the finite well-formed derivation trees $\overline{t}_n$ such that $\mathsf{root}\ t_j = j$ and $\mathsf{Fr}\ t_j \subseteq I$ for all $j \in [n]$. Let $J = \{j' \in [n] \mid \overline{\alpha}\ \mathsf{IF}_{j'} \neq \emptyset\}$. Then F_j is $(I \cup J)$-witnessed, hence by the F_j-completeness of $\mathscr{K}_j$ we obtain $K \in \mathscr{K}_j$ such that $K \subseteq I \cup \{m + j' \mid j' \in J\}$. We take t to have j as root, $I \cap K$ as leaves and $(t_{j'})_{j' \in J}$ as immediate subtrees; namely, $t = \mathsf{Node}\ j\ ((\mathsf{Inl} \bullet I) \cup (\mathsf{Inr} \bullet \{t_{j'} \mid j' \in J\}))$.

Let t_0 be a tree as above corresponding to j_0 (since $\overline{\alpha}\ \mathsf{IF}_{j_0} \neq \emptyset$). Then, by Lemma 2, $t_1 = \mathsf{rcut}\ t_0$ is a well-formed finite derivation tree such that $\mathsf{Fr}\ t_1 \subseteq \mathsf{Fr}\ t_0 \subseteq I$. Thus, taking $I_1 = \mathsf{Fr}\ t_1$, we obtain $I_1 \in \mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(j_0)$ and $I_1 \subseteq I$. $\square$

In what follows, *nl* ranges over lists of nonterminals and '$\cdot$' denotes list concatenation. If n is a nonterminal, n also denotes the n-singleton list. The predicate $\mathsf{path}\ nl\ t$, stating that *nl* is a path in t (starting from the root), is defined inductively as follows:

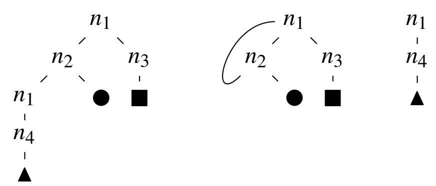

Fig. 3. A finite derivation tree (left), a regular cut of it (middle), and a finite regular cut of it (right)

$\mathsf{path}\ (\mathsf{root}\ t)\ t$
$\mathsf{Inr}\ t' \in \mathsf{cont}\ t \ \wedge\ \mathsf{path}\ nl\ t' \ \Rightarrow\ \mathsf{path}\ ((\mathsf{root}\ t) \cdot nl)\ t'$

Lemma 3. Let t be a finite regular derivation tree. Then t has no paths that contain repetitions.

Proof: Assume, by absurdity, that a path nl in t contains repetitions, i.e., has the form $nl_1 \cdot n \cdot nl_2 \cdot n$, and let t_1 and t_2 be the subtrees corresponding to the paths $nl_1 \cdot n$ and nl, respectively. Then t_2 is a proper subtree of t_1; on the other hand, by the regularity of t, we have $t_1 = t_2$, which is impossible since t_1 and t_2 are finite. $\square$

Proof of Theorem 7: According to Lemma 3 and the properties of regular cuts, we have that (1) $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\ ns'\ n \equiv \mathscr{L}^{\mathrm{pf}}_{\mathsf{Gr}}\ ns'\ n$, where $\mathscr{L}^{\mathrm{pf}}_{\mathsf{Gr}}\ ns'\ n$ is the language defined just like $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\ ns'\ n$, but replacing “regular” with “having no paths that contain repetitions.” Moreover, it is easy to see that (2) the desired facts hold if we replace $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\ ns'\ n$ with $\mathscr{L}^{\mathrm{pf}}_{\mathsf{Gr}}\ ns'\ n$ and $\equiv$ with equality. From (1) and (2) the result follows. $\square$

F Registration of a New BNF in Isabelle

The type constructor α bag of bags (multisets) is registered through the following command:

```
bnf 'a bag
  map: map_bag :: ('a => 'b) => 'a bag => 'b bag
  sets: set_of :: 'a bag => 'a set
  bd: natLeq :: (nat * nat) set
  wits: {#} :: 'a bag
  rel: bag_rel :: ('a => 'b => bool) => 'a bag => 'b bag => bool
```

The command provides the necessary infrastructure that makes α bag a BNF, consisting of various previously introduced constants (whose definitions are not shown here):

- the functorial action, map_bag;
- the “Fset” operation discussed in Section 2.2, set_of;
- a cardinal bound, here, that of natural numbers, natLeq (cardinals are represented as minimal well-order relations);
- a witness term, here, the empty bag {#};
- a customized relator, as discussed in Section B, rel_bag.

Then, the user is asked to prove a few facts, including the nonemptiness witness property `set_of {#} = {}`. Upon discharging these goals, the α bag BNF is registered.