Proceedings Paper:
Blanchette, J.C., Popescu, A. and Traytel, D. (2015) Witnessing (Co)datatypes. In: Vitek, J., (ed.) Programming Languages and Systems: 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings. 24th European Symposium on Programming, ESOP 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, 11-18 Apr 2015, London, UK. Lecture Notes in Computer Science, 9032. Springer Berlin, pp. 359-382. ISBN: 9783662466681. ISSN: 0302-9743. EISSN: 1611-3349.

https://doi.org/10.1007/978-3-662-46669-8_15

Witnessing (Co)datatypes

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

Abstract

Datatypes and codatatypes are useful for specifying and reasoning about (possibly infinite) computational processes. The interactive theorem prover Isabelle/HOL has recently been extended with a definitional package that supports both. Here we describe a complete procedure for deriving nonemptiness witnesses in the general mutually recursive, nested case—nonemptiness being a proviso for introducing new types in higher-order logic. The nonemptiness problem also provides an illuminating case study that shows the package in action, tracing its journey from abstract category theory to hands-on functionality.

General Terms Algorithms, Theory, Verification

Keywords (Co)datatypes, higher-order logic, interactive theorem proving, Isabelle/HOL, category theory

1. Introduction

Interactive theorem provers are becoming increasingly popular as vehicles for formalizing the theory and metatheory of 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 [22]. Codatatypes are also useful to capture lazy data structures, such as Haskell's lists.

Theorem provers based on higher-order logic (HOL), such as HOL4, HOL Light, Isabelle/HOL, and ProofPower–HOL, are traditionally implemented around a trusted inference kernel through which all theorems are generated. Various definitional packages reduce high-level specifications (of types or functions) to primitive inferences; characteristic theorems are derived rather than simply 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. While some theorem provers support codatatypes (e.g., Agda, Coq, and PVS), Isabelle is the first to provide a definitional implementation. Our previous paper [32] developed the underlying constructions, adapted from category theory; in this follow-up, we focus on the more practical aspects of the package.

The main such aspect that concerns us here is the generation of nonemptiness witnesses. Types in HOL are required to be nonempty; the "finite stream" specification

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

must be rejected because it would lead to an empty datatype. For nonempty types, the package discharges nonemptiness proof obligations by exhibiting existential witnesses.

The nonemptiness problem appears deceptively simple: It is well understood for the mutually recursive datatypes traditionally supported by HOL provers, and for codatatypes the final coalgebra is never empty.[1] However, nested recursion via codatatypes and non-free types (e.g., finite sets) complicates the picture, as shown by this attempt at defining infinitely branching trees with finite branches by nested recursion through a codatatype of infinite 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 fails: To obtain a witness for $\alpha\ \mathsf{tree}$, we would need a witness for $(\alpha\ \mathsf{tree})\ \mathsf{stream}$, and vice versa. Replacing infinite streams with finite lists makes the definition acceptable, because the empty list stops the recursion. Even though final coalgebras are never empty, here the datatype provides a better witness (the empty list) than the codatatype (which requires an $\alpha\ \mathsf{tree}$ to build an $(\alpha\ \mathsf{tree})\ \mathsf{stream}$).

Mutual, nested datatype specifications and their nonemptiness witnesses can get complex, as shown by the following commands (aimed at introducing new types with our package):

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

The definitions are legal but the last group becomes illegal if t_2 is replaced by t_3 in the constructor T_{11}.

What makes the problem even more interesting is our open-world assumption: the type constructors handled by our package are not syntactically predetermined, in particular, they are not restricted to polynomial functors—the user is allowed to register new type constructors in the package database after checking a few semantic properties. Thus, assume that, in the above mutual definition of the datatypes t_i, the type constructors list, $\times$ and tree are replaced by others, say, $\alpha\ \mathsf{F}_1$, $(\alpha,\beta)\ \mathsf{F}_2$, and $(\alpha,\beta)\ \mathsf{F}_3$, respectively, about which we may know nonemptiness information, e.g., that $\alpha\ \mathsf{F}_1$ is always nonempty, or $(\alpha,\beta)\ \mathsf{F}_2$ is nonempty whenever α is. What is the weakest property concerning nonemptiness of the F_i's we need to require in order to have the definition go through? While a lightweight (perhaps syntactic) sufficient condition could in principle prove useful, a condition that is also necessary would allow us to proceed in a *complete* fashion, not rejecting any semantically valid construction.

Our solution, presented in Sect. 4, exploits the package's abstract, functorial view of types. Each (co)datatype, and more gener-

[1] We ourselves fell prey to this illusion, being led to think that nonemptiness was a minor implementation matter. This explains why the problem is not mentioned in our earlier paper [32], not even as future work.

ally 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 their components. Each computational step performed by the package is certified in HOL, and the solution is also shown to be complete.

The other main practical aspect on which we focus is the user interface provided by the package. With the implementation in place (witnesses included), we can complete the journey from abstract category theory to hands-on functionality. This is best illustrated through a case study: a variation of context-free grammars acting on finite sets and their associated possibly infinite derivation trees. The example was carefully chosen to display the support for codatatypes and for nested recursion through non-free types; it also supplies, almost clandestinely, precious building blocks to the nonemptiness proofs of Sect. 4, hence its early placement at Sect. 2.

Finally, Sect. 3 recalls the underlying categorical constructions and connects them to the concrete properties exposed to the user. This section plays a double role, simultaneously acting as an epilogue to Sect. 2 and a prologue to Sect. 4. It also describes the process of deriving customized coinduction principles for codatatypes; this engineering aspect is usually ignored by the category theorist, but it is crucial for the mechanized proof developer. Indeed, the raw categorical structure carried over by the package to perform its type constructions is practically unusable, from both a cultural and a technical point of view: the average Isabelle user should not be expected or required to know category theory; moreover, the outputted theorems should be easy to integrate with the other formal developments, typically not centered around notions such as functoriality or natural transformations, main pillars of our constructions. We show how we have addressed these problems by a compositional approach to customization, taking advantage of the overall compositionality and open-endedness of the collection of types targeted by the package.

The formalization and implementation described in this paper are publicly available [8] and are also part of the latest Isabelle/HOL release [19], where the successful and failing (co)datatype definitions exemplified throughout the paper can now be tested.

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 usually prefer the curried form $f : \alpha_1 \to \cdots \to \alpha_n \to \beta$ to the traditional 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; thus, the identity function $\mathsf{id} : \alpha \to \alpha$ is defined for any set α and corresponds to a family $(\mathsf{id}_\alpha)_{\alpha \in \mathscr{S}}$. Another example is function composition, $\circ : (\beta \to \gamma) \to (\alpha \to \beta) \to \alpha \to \gamma$.

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. 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 $\mathsf{Inr}\ (f_2\ a_2)$. 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 $(a_1, \ldots, a_m, b_1, \ldots, b_n)$. Given n m-ary functions $f_1, \ldots, f_n$, $\overline{f}\ \overline{a}$ denotes $(f_1\ \overline{a}, \ldots, f_n\ \overline{a})$, 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 Definitional Package in Action

We introduce the (co)datatype definitional package through a concrete example: derivation trees for a context-free grammar, where we perform the following changes to the usual setting:

- trees are possibly infinite;
- the generated words are not lists, but finite sets.

The formalization of this particular codatatype [8] is the central hub of this paper. First, it is a case study aimed at illustrating some unique characteristics of our package's expressiveness, including the allowance of permutative types such as finite sets inside (co)datatype definitions. Second, it is used to describe the customization process performed by the package. Last but not least, it is employed in the metatheory of arbitrary (co)datatypes needed by the package, namely, in the derivation of nonemptiness witnesses and a proof of soundness and completeness of our algorithm.

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

2.1 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. For both datatypes and codatatypes, the package provides basic properties about constructors and selectors, such as injectivity ($\mathsf{Node}\ n\ as = \mathsf{Node}\ n'\ as' \Leftrightarrow n = n' \wedge as = as'$) and exhaustiveness ($\forall t.\ \exists n\ as.\ t = \mathsf{Node}\ n\ as$).

Coiterator

For dtree, the package also defines the 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 $b \in \beta$,

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

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 specified by r, and its continuation is specified corecursively by c. Formally, $f = \mathsf{unfold}\ r\ c$.

Structural coinduction

Proofs by coinduction are supported by the structural rule

$$\frac{\forall t_1\ t_2.\ \theta\ t_1\ t_2 \Rightarrow \mathsf{root}\ t_1 = \mathsf{root}\ t_2 \wedge \mathsf{fset\_rel}\ (\mathsf{sum\_rel}\ (=)\ \theta)\ (\mathsf{cont}\ t_1)\ (\mathsf{cont}\ t_2)}{\theta\ t_1\ t_2 \Rightarrow t_1 = t_2}$$

The rule is parameterized by a predicate $\theta : \mathsf{dtree} \to \mathsf{dtree} \to \mathsf{bool}$ that is required to be a bisimulation by the antecedent. The predicate $\mathsf{fset\_rel}\ (\mathsf{sum\_rel}\ (=)\ \theta)$ is the componentwise extension of θ to $(\mathsf{T} + \mathsf{dtree})\ \mathsf{fset}$. Unfolding the characteristic theorems for $\mathsf{fset\_rel}$

and sum_rel yields the antecedent

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

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.

The principle effectively states that the equality relation is the largest bisimulation.

2.2 Corecursion and Coinduction on Derivation Trees

We start with a simple example of corecursive definition. Let $+ : \mathsf{N} \times \mathsf{N} \to \mathsf{N}$ be a commutative binary operation. We define a parallel composition of trees $\parallel : \mathsf{dtree} \times \mathsf{dtree} \to \mathsf{dtree}$ (written infix), which combines nonterminal-labeled nodes using $+$ and terminal-labeled leaves using $\cup$, corecursively as $\parallel = \mathsf{unfold}\ r\ c$, where $r : \mathsf{dtree} \times \mathsf{dtree} \to \mathsf{N}$ and $c : \mathsf{dtree} \times \mathsf{dtree} \to (\mathsf{T} + \mathsf{dtree} \times \mathsf{dtree})\ \mathsf{fset}$ are

$$\begin{aligned} r\,(t_1, t_2) &= \mathsf{root}\, t_1 + \mathsf{root}\, t_2 \\ c\,(t_1, t_2) &= \mathsf{Inl} \bullet (\mathsf{Inl}^-(\mathsf{cont}\, t_1) \cup \mathsf{Inl}^-(\mathsf{cont}\, t_2)) \ \cup \\ &\qquad \mathsf{Inr} \bullet (\mathsf{Inr}^-(\mathsf{cont}\, t_1) \times \mathsf{Inr}^-(\mathsf{cont}\, t_2)) \end{aligned}$$

In accordance with the characteristic coiterator equations, $\parallel$ is determined by the corecursive equations

$$\begin{aligned} \mathsf{root}\,(t_1 \parallel t_2) &= r\,(t_1, t_2) \\ \mathsf{cont}\,(t_1 \parallel t_2) &= (\mathsf{id} \oplus \parallel) \bullet c\,(t_1, t_2) \end{aligned}$$

By expanding r and c and massaging the second equation, we obtain these characterizations:

$$\begin{aligned} \mathsf{root}\,(t_1 \parallel t_2) &= \mathsf{root}\, t_1 + \mathsf{root}\, t_2 \\ \mathsf{Inl}^-(\mathsf{cont}\,(t_1 \parallel t_2)) &= \mathsf{Inl}^-(\mathsf{cont}\, t_1) \ \cup\ \mathsf{Inl}^-(\mathsf{cont}\, t_2) \\ \mathsf{Inr}^-(\mathsf{cont}\,(t_1 \parallel t_2)) &= \parallel \bullet\, (\mathsf{Inr}^-(\mathsf{cont}\, t_1) \ \times\ \mathsf{Inr}^-(\mathsf{cont}\, t_2)) \\ &= \{t_1' \parallel t_2' \ \mid\ t_1' \in \mathsf{Inr}^-(\mathsf{cont}\, t_1) \wedge \\ &\qquad\qquad t_2' \in \mathsf{Inr}^-(\mathsf{cont}\, t_2)\} \end{aligned}$$

The equations specify the root, the successor leaves, and the immediate subtrees of the tree $t_1 \parallel t_2$, in that order. To prove that $\parallel$ is commutative, we first formulate the goal in the format expected by the structural coinduction rule, $\theta\, t\, t' \Rightarrow t = t'$, with $\theta\, t\, t' \Longleftrightarrow \exists t_1\, t_2.\ t = t_1 \parallel t_2 \wedge t' = t_2 \parallel t_1$. Structural coinduction yields the following subgoals, which are easy to discharge using the equations for $\parallel$ and commutativity of $+$ and $\cup$:

$$\begin{aligned} &\mathsf{root}\,(t_1 \parallel t_2) = \mathsf{root}\,(t_2 \parallel t_1) \\ &\mathsf{Inl}^-(\mathsf{cont}\,(t_1 \parallel t_2)) = \mathsf{Inl}^-(\mathsf{cont}\,(t_2 \parallel t_1)) \\ &\forall t \in \mathsf{Inr}^-(\mathsf{cont}\,(t_1 \parallel t_2)).\ \exists t' \in \mathsf{Inr}^-(\mathsf{cont}\,(t_2 \parallel t_1)).\ \theta\, t\, t' \\ &\forall t' \in \mathsf{Inr}^-(\mathsf{cont}\,(t_2 \parallel t_1)).\ \exists t \in \mathsf{Inr}^-(\mathsf{cont}\,(t_1 \parallel t_2)).\ \theta\, t\, t' \end{aligned}$$

2.3 A Variation of Context-Free Grammars

We continue with a larger example that illustrates the coinductive infrastructure offered by the package. The example is a variation of the notion of context-free grammar, 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 [26]. 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.

We illustrate fixpoint (co)induction on dtree. *Well-formed* derivation trees for Gr are defined coinductively as the greatest predicate $\mathsf{wf} : \mathsf{dtree} \to \mathsf{bool}$ such that, for all $t \in \mathsf{dtree}$,

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

Intuitively, 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 $\mathsf{subtr}\ t\ t' \Longleftrightarrow t = t' \vee (\exists t''.\ \mathsf{subtr}\ t\ t'' \wedge \mathsf{Inr}\ t'' \in \mathsf{cont}\ t')$ holds for all $t, t' \in \mathsf{dtree}$.[2] A coinductive definition would be unsuitable here: $\mathsf{subtr}\ t\ t'$ would be true for any infinite tree t' by virtue of the second disjunct (the step). We write $\mathsf{Subtr}\ t$ for the set of subtrees of t. The frontier $\mathsf{Fr} : \mathsf{dtree} \to \mathsf{T}\ \mathsf{set}$ and interior $\mathsf{Itr} : \mathsf{dtree} \to \mathsf{N}\ \mathsf{set}$ of a tree are defined similarly. The language generated by the grammar Gr from a nonterminal $n \in \mathsf{N}$ (using possibly infinite derivation trees) is defined as $\mathscr{L}_{\mathsf{Gr}}(n) = \{\mathsf{Fr}\ t \mid \mathsf{wf}\ t \wedge \mathsf{root}\ t = n\}$.

The choice of the permutative structure of finite sets instead of lists to represent words has important consequences for the computation of the generated language, where regular trees suffice. We we elaborate on this next, taking the opportunity to illustrate corecursion.

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 regularly generated 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 $r : \mathsf{Itr}\ t_0 \to \mathsf{T}$ 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

[2] Inductive predicates are often specified by their introduction rules—here, $\mathsf{subtr}\ t\ t$ and $\mathsf{subtr}\ t\ t'' \wedge \mathsf{Inr}\ t'' \in \mathsf{cont}\ t' \Rightarrow \mathsf{subtr}\ t\ t'$. The formulation as an equivalence is convenient for its uniform treatment of the coinductive case.

corecursive equations

$$\mathsf{root}\ (\mathsf{H}\ n) = n$$
$$\mathsf{cont}\ (\mathsf{H}\ n) = (\mathsf{id} \oplus (\mathsf{H} \circ \mathsf{root})) \bullet \mathsf{cont}\ (\mathsf{pick}\ n)$$

It is not hard to prove the following by fixpoint coinduction:

Theorem 1. For all $n \in \mathsf{ltr}\ t_0$, $\mathsf{H}\ n$ 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.

(The proof is given in Appendix D.) Therefore we take $\mathsf{rcut}\ t_0$ to be $\mathsf{H}\ (\mathsf{root}\ t_0)$.

Figure 1 illustrates 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 in the right-hand tree denote infinite trees that are their own subtrees.

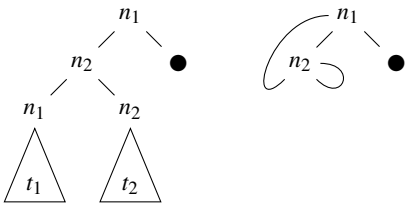

Fig. 1. A derivation tree (left) and its simplest regular cut (right)

3. The Journey from Category Theory

The previous section's definitions rest on characteristic theorems provided by Isabelle's (co)datatype package—they are derived from underlying constructions adapted from category theory. The fundamental concept is that of bounded natural functors, a well-behaved class of functors with additional structure. The theory is an essential preliminary to the nonemptiness witness computation of Sect. 4 and puts the case study of Sect. 2 on more solid ground.

3.1 Functors and Functor Operations

We consider various operators F on sets, such as sums, products, etc., which we call *set constructors*. We are interested in set constructors that are *functors* on the category of sets and functions, meaning that they are equipped with an action on morphisms commuting with identities and composition. This action is a polymorphic constant $\mathsf{Fmap} : (\alpha_1 \to \beta_1) \to \cdots \to (\alpha_n \to \beta_n) \to \overline{\alpha}\ \mathsf{F} \to \overline{\beta}\ \mathsf{F}$ satisfying $\mathsf{Fmap}\ \mathsf{id}^n = \mathsf{id}$ 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 $(\mathsf{F}, \mathsf{Fmap})$. We let Func_n be the collection of n-ary functions. Let us review basic examples.

Identity functor (ID, id)

The identity functor 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 involved operators have already been described in Sect. 1.

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 more complex functors. Composition requires the functors F_j to take the same type arguments $\overline{\alpha}$ in the same order. The auxiliary operations of permutation and lifting, together with the identity and (n,α)-constant functors (Sect. 3.1), make it possible to compose functors freely.

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 codatatype 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* $\mathsf{fold}_j : \left(\prod_{k\in[n]}\ (\overline{\alpha}, \overline{\beta})\ \mathsf{F}_k \to \beta_k\right) \to \overline{\alpha}\ \mathsf{IF}_j \to \beta_j$

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 $f_j : \overline{\alpha}\ \mathsf{IF} \to \beta_j$ 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* $\mathsf{unfold}_j : \left(\prod_{k\in[n]}\ \beta_k \to (\overline{\alpha}, \overline{\beta})\ \mathsf{F}_k\right) \to \beta_j \to \overline{\alpha}\ \mathsf{JF}_j$

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 $f_j : \beta_j \to \overline{\alpha}\ \mathsf{JF}$ 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.

The (co)datatype package is based on a class $\mathscr{B}$ of functors, called *bounded natural functors* (*BNFs*) [32], such that $\mathscr{B}$ contains all the basic functors listed in Sect. 3.1 except for unbounded powerset, and $\mathscr{B}$ supports, and is closed under, the above operations.

For the derivation tree example of Sect. 2.1, the input BNF to the initial algebra operation is $\mathsf{pre\_dtree} = (\times) \circ (\mathsf{C_N}, \mathsf{fset} \circ ((+) \circ (\mathsf{C_T}, \mathsf{ID})))$. In the sequel, we prefer the more readable notation $\alpha\ \mathsf{pre\_dtree} = \mathsf{N} \times (\mathsf{T} + \alpha)\ \mathsf{fset}$.

3.2 Open-Endedness

Unlike the (co)datatype specification mechanisms of most theorem provers (such as the previous one of Isabelle and the current ones of the other HOL-based provers, as well as those of PVS and Coq), here 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 infrastructure (in the form or 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. (The full list of BNF-defining properties can be found in [32].) After this, the new BNF is integrated and can appear nested in future (co)datatype definitions. For instance, Isabelle users have recently introduced the BNFs $\alpha\ \mathsf{bag}$, of bags (i.e., bags with finite-multiplicity elements) over α, and $\alpha\ \mathsf{list}_5$, of lists of at most five elements. (More details are given in Appendix E.)

3.3 Abstract Structural (Co)induction

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? The 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 applied to a list x gives all the elements appearing in x.[3] As usual, if F is unary, we omit the superscript from Fset^1.

Given $j \in [n]$, the elements of $\mathsf{Fset}_j^{m+k}\ x$ (for $k \in [n]$) are the recursive components of $\mathsf{ctor}_j\ x$. (Note that here subscripts select functors F_j in the tuple $\overline{\mathsf{F}}$; then, as previously agreed, superscripts select Fset operators for different arguments of F_j.) Using this insight, 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 gives the abstract principle

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

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 [29]: that our functors preserve weak pullbacks, which allows us to organize them as relators [28]. 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:

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

Structural coinduction can be stated abstractly as

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

for sets $\overline{\alpha}_n$ and binary predicates $\theta_j \in \overline{\alpha}\ \mathsf{JF}_j \to \overline{\alpha}\ \mathsf{JF}_j \to \mathsf{bool}$ [28]. The antecedent ensures that θ is a bisimulation.

3.4 Completing the Journey: From Abstract to Concrete

Given the definition of dtree (Sect. 2.1), the package first composes the intermediate BNF $\alpha\ \mathsf{pre\_dtree} = \mathsf{N} \times (\mathsf{T} + \alpha)\ \mathsf{fset}$ from the constants N and T, identity, sum, product, and finite set. 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 are derived from those of sums and products.

Customizing corecursion and coinduction

The abstract coiteration principle described in Sect. 3.1 relies on a coiterator $\mathsf{unfold} : (\beta \to \beta\ \mathsf{pre\_dtree}) \to \beta \to \mathsf{dtree}$ such that $\mathsf{dtor} \circ \mathsf{unfold}\ s = \mathsf{pre\_dtree\_map}\ (\mathsf{unfold}\ s) \circ s$. 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, we obtain $\mathsf{dtor}\ (\mathsf{unfold}\ \langle r, c\rangle\ b) = \mathsf{pre\_dtree\_map}\ (\mathsf{unfold}\ s)\ (r\,b, c\,b)$. This can be further improved by unfolding the definition of $\mathsf{pre\_dtree\_map}$, 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

[3] Our Fset has similarities with Pierce's notion of support from his account of (co)inductive types [27] and with Abel and Altenkirch's urelement relation from their framework for strong normalization [2]. The main distinguishing feature of our notion is that it additionally takes categorical structure into consideration [32].

$\mathsf{cont}\ (\mathsf{unfold}\ \langle r, c\rangle\ b) = (\mathsf{id} \oplus \mathsf{unfold}\ \langle r, c\rangle) \bullet c\ b$. The coiteration rule of Sect. 3.1 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 $\mathsf{unfold}'\ r\ c = \mathsf{unfold}\ \langle r, c\rangle$.

Customizing the abstract structural coinduction principle from Sect. 3.3 requires more work. From the abstract, functor-agnostic notion of bisimulation, our goal is to derive concrete coinduction rules that avoid any references to the intermediate functor F (e.g., pre_dtree). Category-theoretical approaches typically appeal to ad hoc proofs that the abstract formulation of bisimulation, in terms of the graph of Fmap, is equivalent to the traditional bisimulation principle for various interesting concrete cases (e.g., [29, Ex. 2.1]). However, in a theorem prover, we prefer a uniform, automatic customization that yields the expected concrete instances.

The key is to employ an alternative compositional characterization. To this end, each standard basic BNF provides an ad hoc characterization. For example:

- Product: $\mathsf{prod\_rel}\ \theta_1\ \theta_2\ (a_1, a_2)\ (b_1, b_2) \Leftrightarrow \theta\ a_1\ b_1 \wedge \theta\ a_2\ b_2$
- Bounded k-Powerset: $\mathsf{set\_rel}_k\ \theta\ A\ B \Leftrightarrow (\forall a \in A.\ \exists b \in B.\ \theta\ a\ b) \wedge (\forall b \in B.\ \exists a \in A.\ \theta\ a\ b)$

The same can be achieved for any potential user-defined BNF: The user is prompted to provide an ad hoc characterization of the relator. (If the user provides none, then the general-purpose definition in terms of Fmap is kept.) E.g., for bags, a convenient such characterization proceeds inductively by the following clauses, where $\uplus$ denotes bag union (by summing the elements' multiplicities):

$$\mathsf{bag\_rel}\ \theta\ \emptyset\ \emptyset \qquad \frac{\mathsf{bag\_rel}\ \theta\ A\ B \qquad \theta\ a\ b}{\mathsf{bag\_rel}\ \theta\ (A \uplus \{a\})\ (B \uplus \{b\})}$$

Moreover, each BNF operation derives a characterization of the resulting relator in terms of those of the components.

- Composition: $\mathsf{Hrel}\ \overline{\theta} = \mathsf{Grel}\ (\overline{\mathsf{Frel}}\ \overline{\theta})$
- Initial algebra: $\mathsf{IFrel}_j\ \overline{\theta}\ (\mathsf{ctor}_j\ a)\ (\mathsf{ctor}_j\ b) \Leftrightarrow \mathsf{Frel}_j\ \overline{\theta}\ (\overline{\mathsf{IFrel}}\ \overline{\theta})\ a\ b$
- Final coalgebra: $\mathsf{JFrel}_j\ \overline{\theta}\ a\ b \Leftrightarrow \mathsf{Frel}_j\ \overline{\theta}\ (\overline{\mathsf{JFrel}}\ \overline{\theta})\ (\mathsf{dtor}_j\ a)\ (\mathsf{dtor}_j\ b)$

The last two equivalences hold in a fixpoint-(co)inductive way: For the initial algebra, the IFrel_j's are the least predicates (mutually) satisfying the equivalence; for the final coalgebra, the JFrel_j's are the greatest predicates satisfying the equivalence.

For $\alpha\ \mathsf{pre\_dtree} = \mathsf{N} \times (\mathsf{T} + \alpha)\ \mathsf{fset}$ and its final coalgebra dtree, the relators are characterized by the following equivalences:

$$\mathsf{pre\_dtree\_rel}\ \theta\ a\ b \Leftrightarrow \mathsf{prod\_rel}\ (\mathsf{C_N\_rel}\ \theta)\ (\mathsf{fset\_rel}\ (\mathsf{sum\_rel}\ (\mathsf{C_T\_rel}\ \theta)\ (\mathsf{ID\_rel}\ \theta)))\ a\ b$$

$$\mathsf{dtree\_rel}\ a\ b \Leftrightarrow \mathsf{pre\_dtree\_rel}\ \mathsf{dtree\_rel}\ (\mathsf{dtor}\ a)\ (\mathsf{dtor}\ b)$$

The final formulation, as presented in Sect. 2.1, emerges by substituting $\langle \mathsf{root}, \mathsf{cont}\rangle$ for dtor and unfolding the characteristic equations for the involved basic BNFs.

4. Computing Nonemptiness Witnesses

In the previous two sections, we referred to the codatatype dtree and other collections of elements as *sets*, postponing the discussion of an important aspect of our package. 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 [6, 10]: 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.
- It is not clear how to unfold datatypes nested in codatatypes or vice versa.

Note that, 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 how they look like* (hence, with no option to "delve" into them)? To answer this, we use a generalization of pointed types [17, 21], maintaining witnesses that assert conditional nonemptiness for combinations of arguments. We first present the solution by examples.

4.1 Examples

We start with the very simple 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 the signature of the witness (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 $\mathsf{pre\_list\_wit}_1 = \mathsf{Inl}\ ()$ and $\mathsf{pre\_list\_wit}_2\ a\ b = \mathsf{Inr}\ (a, b)$. 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 $\mathsf{list\_ctor}\ \mathsf{pre\_list\_wit}_1$ (i.e., Nil) for $\alpha\ \mathsf{list}$.

Because they can store infinite objects, codatatypes are never empty. 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:

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

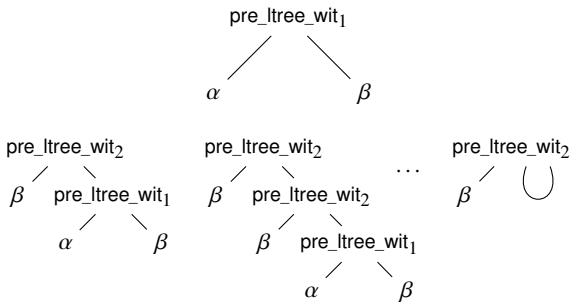

Fig. 2. Derivation trees for ltree witnesses

```
codatatype (α,β) ltree =
  LNode β ((α + (α,β) ltree) 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 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 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 (α,β,γ) pre_ltree $= \beta \times (\alpha + \gamma)$ stream with γ representing the corecursive component. Composition yields two witnesses for pre_ltree:

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

These can serve to build infinitely many witnesses for ltree. Figure 2 enumerates the possible combinations, starting with $\mathsf{pre\_ltree\_wit}_1$. This witness requires only the non-corecursive components α and β being nonempty, and hence immediately yields a witness $\mathsf{ltree\_wit}_1 : \alpha \to \beta \to (\alpha,\beta)$ ltree (by applying the constructor LNode). The second witness $\mathsf{pre\_ltree\_wit}_2$ 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 $\mathsf{pre\_ltree\_wit}_1$ or $\mathsf{pre\_ltree\_wit}_2$, and so on. At the limit, $\mathsf{pre\_ltree\_wit}_2$ is used infinitely often. The corresponding witness $\mathsf{ltree\_wit}_2 : \beta \to (\alpha,\beta)$ ltree can be defined by coiteration as $\lambda b.\ \mu t.\ \mathsf{pre\_ltree\_wit}_2\ b\ t$. 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 thus 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: $\mathscr{I}_{\alpha\ \mathsf{set}_k} = \{\emptyset\}$
- Bag: $\mathscr{I}_{\alpha\ \mathsf{bag}} = \{\emptyset\}$

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 2. 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 $\{\bigcup_{j\in J} I \mid J \in \mathscr{J} \wedge (I_j)_{j\in J} \in \prod_{j\in J} \mathscr{I}_j\}$ is a perfect set for H.

Theorem 3. Let $\mathscr{I} \subseteq [n]$ be a perfect set for F. Then $\mathscr{I}$ is also perfect for $\mathsf{F}^{(i,j)}$ and $\mathsf{F}{\uparrow}$.

Theorems 2 and 3 hold not only for functors but also for plain set constructors (with a further cardinality-monotonicity assumption needed for the completeness part of Theorem 2). The most interesting cases are the genuinely functorial ones of initial algebras and final coalgebras, which we discuss next.

Witnesses for the initial algebra and the final coalgebra 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 Sect. 2.3 turn out to be perfectly suited for recording the combinatorics of these compositions, in such a way that not only soundness, but also 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 construct a (set) grammar $\mathsf{Gr} = (\mathsf{T},\mathsf{N},\mathsf{P})$ as in Sect. 2.3 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:

$$\begin{aligned} \overline{\alpha}\ \mathsf{IF}_1 &\cong (\overline{\alpha}, \overline{\alpha}\ \mathsf{IF}_1, \ldots, \overline{\alpha}\ \mathsf{IF}_n)\ \mathsf{F}_1 \\ &\vdots \\ \overline{\alpha}\ \mathsf{IF}_n &\cong (\overline{\alpha}, \overline{\alpha}\ \mathsf{IF}_1, \ldots, \overline{\alpha}\ \mathsf{IF}_n)\ \mathsf{F}_n \end{aligned}$$

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 possibly infinite derivations is still sound, and also becomes complete.

Theorem 4. Assume that the (mutual) final coalgebra of $\overline{\mathsf{F}}$ exists and consists of n m-ary functors $\overline{\alpha}_m\ \mathsf{JF}_j$ (as in Sect. 3.1). 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 Theorem 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 $j \in \mathsf{ltr}\ t_0$, 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 $j \in \mathsf{ltr}\ t_0$ 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 $j \notin \mathsf{ltr}\ t_0$, s_j is the unique function from $\emptyset$. If $j \in \mathsf{ltr}\ t_0$, 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 $j \in \mathsf{ltr}\ t_0$, $\mathsf{unfold}_j\ \overline{s} : \mathsf{unit} \to \overline{\alpha}\ \mathsf{JF}_j$ 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 $J = \{j.\ \overline{\alpha}\ \mathsf{F}_j \neq \emptyset\}$. 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 $K_j \subseteq I \cup \{m + j.\ j \in J\}$.

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 $\mathsf{cont}\ (g\ j) = (\mathsf{id} \oplus g) \bullet c\ j = \mathsf{Inl} \bullet (K_j \cap I) \cup \mathsf{Inr} \bullet \{g\ j.\ m + j \in K_j\}$. Taking $t_0 = g\ j_0$ and using Theorem 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. □

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 5. Assume that the (mutual) initial algebra of $\overline{\mathsf{F}}$ exists and consists of n m-ary functors $\overline{\alpha}_m\ \mathsf{IF}_j$ (as in Sect. 3.1). Then $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}(j)$ is a perfect set for IF_j for $j \in [n]$.

Let us see how Theorems 2–5 can be combined in establishing or refuting nonemptiness for some of our motivating examples from Sect. 4.1 and the introduction. These (and other) examples have been checked with our package [8]:

- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_list}} = \{\emptyset\}$ by Th. 2; $\mathscr{I}_{\alpha\ \mathsf{list}} = \{\emptyset\}$ by Th. 5
- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_fstream}} = \{\{\alpha,\beta\}\}$;
  $\mathscr{I}_{\alpha\ \mathsf{fstream}} = \emptyset$ by Th. 5 (i.e. α fstream is empty)
- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_stream}} = \{\{\alpha,\beta\}\}$; $\mathscr{I}_{\alpha\ \mathsf{stream}} = \{\{\alpha\}\}$ by Th. 4
- $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_x}} = \{\{\beta\}\}$ and $\mathscr{I}_{(\alpha,\beta)\ \mathsf{pre\_y}} = \{\emptyset\}$;
  $\mathscr{I}_{\mathsf{x}} = \{\emptyset\}$ and $\mathscr{I}_{\mathsf{y}} = \{\emptyset\}$ by Th. 5
- $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_ltree}} = \{\{\alpha,\beta\}, \{\beta,\gamma\}\}$ by Th. 2;
  $\mathscr{I}_{(\alpha,\beta)\ \mathsf{ltree}} = \{\{\beta\}\}$ by Th. 4
- $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_t_1}} = \{\{\beta\}, \{\alpha,\gamma\}\}$, $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_t_2}} = \{\emptyset\}$, and
  $\mathscr{I}_{(\alpha,\beta,\gamma)\ \mathsf{pre\_t_3}} = \{\{\alpha\}, \{\gamma\}\}$ by Th. 2;
  $\mathscr{I}_{\mathsf{t}_i} = \{\emptyset\}$ by Th. 5

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

Theorem 6. 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 2–5.

Consequently, a procedure implementing Theorems 2–5 will preserve enough nonemptiness witnesses to ensure that all datatype specifications describing nonempty types are accepted. The next subsection presents such a procedure.

4.3 Computational Aspects

Theorem 4 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, regular infinite trees are of course excessive—we next show that $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}(n)$ can be computed directly via recursive equations not involving derivation trees.

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

Recall that our derivation trees from Sect. 2 produce not usual words (defined as lists), but finite sets—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

$$L \leq L' \ \text{ iff } \ \forall w \in L.\ \exists w' \in L'.\ w' \subseteq w$$

and equivalence, $\equiv$, by

$$L \equiv L' \ \text{ iff } \ L \leq L' \text{ and } L' \leq L$$

It is easy to see that any set $\equiv$-equivalent to a perfect set is again perfect. Note also that Theorem 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 the available productions whose source nonterminals are in *ns*, removing each time the expanded nonterminal from *ns*. Thus, provided n is in the allowed set *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 set *ns*. Formally:

Theorem 7. For all $ns \subseteq \mathsf{N}$ and $n \in \mathsf{N}$, $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n \equiv$

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

Theorem 7 provides an alternative, recursive definition of $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n$. The definition terminates because the argument *ns* is fi-

[4] This is in effect the Smyth preorder extension [31] of the subword relation.

nite 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 intended application translates to the number of simultaneously introduced codatatypes.

Here is how this 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

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

For datatypes, the computation of $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}$, is achieved analogously to Theorem 7, by defining $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\,ns\,n$ as a generalization of $\mathscr{L}^{\mathrm{rf}}_{\mathsf{Gr}}\,n$.

Theorem 8. The statement of Theorem 7 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/ML

The package maintains nonemptiness information to be prepared for producing nonemptiness proofs upon datatype definitions. The equations from Theorems 7 and 8 involve only executable operations over finite sets of numbers, sums, and products. Since the descriptions of Theorems 2 and 3 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 $\mathsf{Fin}\,\overline{A} = \{x.\ \forall i \in [n].\ \mathsf{Fset}^i\ x \subseteq A_i\}$. 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 [20], 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 has to bear the burden of both *computing* HOL terms and *certifying* the computation, i.e., ensuring that certain terms are theorems. The combinatorial computation of witnessing sets of indices described in Theorems 7 and 8 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 Sect. 2:

```
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 4 and 5 from Sect. 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 witness computation happens outside the logic, but the witnesses are then verified through Isabelle's kernel. After introducing any BNF, its witness set is minimized, in that redundant witnesses are removed.

The whole development pertaining to the production and certification of witnesses amounts to about 500 lines of Standard ML [8].

6. Related Work

Coinductive (or coalgebraic) datatypes have become popular in recent years in the study of infinite behaviors and non-terminating computation. Whereas inductive datatypes are well-studied and widely available in most programming languages and theorem provers, 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 [7, 25], Agda [3], and CIRC [23]. The work of this paper is in line with this research and is part of the larger task of extending Isabelle/HOL with a flexible coinductive infrastructure.

Other definitional packages must also prove nonemptiness of newly defined types, but typically the proofs are easy: Homeier's quotient package for HOL4 [16] exploits the observation that quotients of nonempty sets are nonempty, and Huffman's (co)recursive domain package for Isabelle/HOLCF [18] can rely on a minimal element $\bot$. For the traditional datatype packages introduced by Melham [24], extended by Gunter [11], simplified by Harrison [12], and implemented in Isabelle/HOL by Berghofer and Wenzel [6], proving nonemptiness is nontrivial, but by reducing nested definitions to mutual definitions, they could employ a standard reachability analysis [6, § 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—this is discussed in detail in a previous paper [32], with many pointers to the literature. In category theory settings however, type nonemptiness is often not crucial, either because the developments do not target any particular logic [5, 9, 13, 15, 29] or because the targeted logics cater for (potentially empty) dependent types [1, 4, 14, 30]. Our nonemptiness-witness maintenance has similarities with the preservation of enriched types along various constructions, e.g., initial algebras and final coalgebras of pointed functors are also pointed [17]. On the other hand, none of these categorical frameworks is concerned with notions of completeness of this extra information.

7. Conclusion

We presented a complete solution to the nonemptiness witness problem that arises in the context of Isabelle's new (co)datatype package. The solution exploits the functorial view of types that lies at the heart of the package. Despite its location at the intersection of category theory and higher-order logic, the problem has broad practical motivation in terms of the popularity of HOL provers. Our solution, like the rest of the definitional package, is part of the 2013 edition of Isabelle.

The problem and its solution also enjoy an elegant metatheory, which itself is best expressed in terms of codatatypes. In our case study, the package simultaneously presented itself and parts of its own metatheory.

Although the underlying categorical constructions were introduced in an earlier paper [32], we found it useful to recall them here. We used this opportunity to connect the theoretical constructions with the concrete properties made available to end users and describe the constructions in their full generality—for n-ary functors, mutual initial algebras, and mutual final coalgebras.

Acknowledgment

Tobias Nipkow made this collaboration 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. Andreas Abel and Martin Hofmann pointed out relevant work. Mark Summerfield and several anonymous reviewers commented on drafts of this paper. The first author's research was supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant Ni 491/14-1). The second author's research was supported by the project Security Type Systems and Deduction (grant Ni 491/13-1) as part of the DFG program Reliably Secure Software Systems (RS^3, Priority Program 1496). The third author's research was supported by the DFG doctorate program 1480 (PUMA). The authors are listed in alphabetical order regardless of individual contributions or seniority.

References

[1] M. Abbott, T. Altenkirch, and N. Ghani. Containers: Constructing strictly positive types. *Theor. Comput. Sci.*, 342(1):3–27, 2005.

[2] A. Abel and T. Altenkirch. A predicative strong normalisation proof for a lambda-calculus with interleaving inductive types. In T. Coquand, P. Dybjer, B. Nordström, and J. M. Smith, eds., *TYPES '99*, vol. 1956 of *LNCS*, pp. 21–40. Springer, 2000.

[3] A. Abel, B. Pientka, D. Thibodeau, and A. Setzer. Copatterns: programming infinite structures by observations. In *POPL*, pp. 27–38, 2013.

[4] T. Altenkirch and P. Morris. Indexed containers. In *LICS 2009*, pp. 277–285. IEEE, 2009.

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

[6] S. Berghofer and M. Wenzel. Inductive datatypes in HOL—Lessons learned in formal-logic engineering. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, eds., *TPHOLs '99*, vol. 1690 of *LNCS*, pp. 19–36, 1999.

[7] Y. Bertot. Filters on coinductive streams, an application to Eratosthenes' sieve. In *TLCA '05*, pp. 102–115, 2005.

[8] J. C. Blanchette, A. Popescu, and D. Traytel. Formal development associated with this paper. `http://www21.in.tum.de/~traytel/codata_wit_devel.tar.gz`.

[9] N. Ghani, P. Johann, and C. Fumex. Fibrational induction rules for initial algebras. In *CSL*, pp. 336–350, 2010.

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

[11] E. L. Gunter. A broader class of trees for recursive type definitions for HOL. In J. J. Joyce and C.-J. H. Seger, eds., *HUG '93*, vol. 780 of *LNCS*, pp. 141–154. Springer, 1994.

[12] J. Harrison. Inductive definitions: Automation and application. In E. T. Schubert, P. J. Windley, and J. Alves-Foss, eds., *TPHOLs '95*, vol. 971 of *LNCS*, pp. 200–213. Springer, 1995.

[13] R. Hasegawa. Two applications of analytic functors. *Theor. Comput. Sci.*, 272(1-2):113–175, 2002.

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

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

[16] P. V. Homeier. A design structure for higher order quotients. In J. Hurd and T. F. Melham, eds., *TPHOLs 2005*, vol. 3603 of *LNCS*, pp. 130–146. Springer, 2005.

[17] B. T. Howard. Inductive, coinductive, and pointed types. In *ICFP*, pp. 102–109, 1996.

[18] B. Huffman. A purely definitional universal domain. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, eds., *TPHOLs 2009*, vol. 5674 of *LNCS*, pp. 260–275. Springer, 2009.

[19] The Isabelle theorem prover, 2013. http://isabelle.in.tum.de/.

[20] A. Krauss and T. Nipkow. Proof pearl: Regular expression equivalence and relation algebra. *J. Autom. Reasoning*, 49(1):95–106, 2012.

[21] M. Lenisa, J. Power, and H. Watanabe. Distributivity for endofunctors, pointed and co-pointed endofunctors, monads and comonads. *Electr. Notes Theor. Comput. Sci.*, 33:230–260, 2000.

[22] A. Lochbihler. Java and the Java memory model—A unified, machine-checked formalisation. In H. Seidl, ed., *ESOP 2012*, vol. 7211 of *LNCS*, pp. 497–517. Springer, 2012.

[23] D. Lucanu, E.-I. Goriac, G. Caltais, and G. Roşu. CIRC: A behavioral verification tool based on circular coinduction. In *CALCO'09*, pp. 433–442, 2009.

[24] T. F. Melham. Automating recursive type definitions in higher order logic. In G. Birtwistle and P. A. Subrahmanyam, eds., *Current Trends in Hardware Verification and Automated Theorem Proving*, pp. 341–386. Springer, 1989.

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

[26] L. C. Paulson. A fixedpoint approach to (co)inductive and (co)datatype definitions. In G. D. Plotkin, C. Stirling, and M. Tofte, eds., *Proof, Language, and Interaction—Essays in Honour of Robin Milner*, pp. 187–212. MIT Press, 2000.

[27] B. C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

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

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

[30] C. Schwaab and J. G. Siek. Modular type-safety proofs in Agda. To appear in *PLPV 2013*.

[31] M. B. Smyth. Power domains. *J. Comput. Syst. Sci.*, 16(1):23–36, 1978.

[32] D. Traytel, A. Popescu, and J. C. Blanchette. Foundational, compositional (co)datatypes for higher-order logic—Category theory applied to theorem proving. In *LICS 2012*, pp. 596–605. IEEE, 2012.

APPENDIX

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

A. Embedding of Initial Algebras into Final Coalgebras

It is well-known that, for n $(m+n)$ functors $(\overline{\alpha},\overline{\beta})\,\mathsf{F}_j$, their mutual initial algebra can be embedded in their mutual final coalgebra via $\mathsf{emb}_j : \overline{\alpha}\ \mathsf{IF}_j \to \overline{\alpha}\ \mathsf{JF}_j$ defined

- either recursively, as $\mathsf{emb}_j = \mathsf{fold}_j\ \overline{\mathsf{ctor}'}$, where $\mathsf{ctor}'_j : (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{JF}})\,\mathsf{F}_j \to \overline{\alpha}\ \mathsf{JF}_j$ is the inverse of the final-coalgebra unfolding bijection dtor_j,
- or corecursively, as $\mathsf{emb}_j = \mathsf{unfold}\ \overline{\mathsf{dtor}'}$, where $\mathsf{dtor}'_j : \overline{\alpha}\ \mathsf{IF}_j \to (\overline{\alpha}, \overline{\alpha}\ \overline{\mathsf{IF}})\,\mathsf{F}_j$ is the inverse of the initial-algebra folding bijection ctor_j.

Along this embedding, ctor'_j is the extension of ctor_j and dtor_j is the extension of dtor'_j. Moreover, the image of emb_j consists precisely of the "finite" elements of $\overline{\alpha}\ \mathsf{JF}_j$, i.e., of those characterized by the induction principle of $\overline{\alpha}\ \mathsf{IF}_j$. Therefore, we define $\overline{\mathsf{JFfinite}}$ as the least tuple of predicates satisfying the following specification.

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

We have that, for all j, $\mathsf{emb}_j \bullet (\overline{\alpha}\ \mathsf{IF}_j) = \{a \in \overline{\alpha}\ \mathsf{JF}_j \mid \mathsf{JFfinite}_j\ a\}$.

B. More on the Coinductive Tree Case Study

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

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

$\mathsf{Fr} : \mathsf{dtree} \to \mathsf{N\ set}$ 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 Sect. 4.3), $\mathsf{Itr} : \mathsf{N\ set} \to \mathsf{dtree} \to \mathsf{N\ set}$ and $\mathsf{Fr} : \mathsf{N\ set} \to \mathsf{dtree} \to \mathsf{N\ set}$, are defined inductively modifying the above clauses textually as follows:

- '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:

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

ftree is in effect an instance of the JFfinite predicate introduced in Appendix A.

C. Inductive Trees

The datatype definition

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

(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 Sect. 2.1.

The difference concerns induction and recursion.

Iteration

The general principle described in Sect. 3.1 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

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

that is,

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

The fdtree-defining BNF is the same as the dtree-defining BNF: $\beta\ \mathsf{pre\_fdtree} = \mathsf{N} \times (\mathsf{T} + \beta)\ \mathsf{fset}$ and $\mathsf{pre\_fdtree\_map}\ f = \mathsf{id} \otimes (\mathsf{image}\ (\mathsf{id} \oplus f))$.

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

$$\forall s : \mathsf{N} \times (\mathsf{T} + \beta)\ \mathsf{fset} \to \beta.\ \forall n\ as.\ \mathsf{fold}\ s\ (\mathsf{FNode}\ n\ as) = s\ (\mathsf{pre\_fdtree\_map}\ (\mathsf{fold}\ s)\ (n, as))$$

By unfolding the definition of pre_fdtree_map, we obtain

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

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 Sect. 3.3 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:

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

Embedding of fdtree in dtree

Here is what the standard embedding emb described in Appendix A looks like for the concrete instance $\mathsf{fdtree} \to \mathsf{dtree}$:

- recursively: fold Node, yielding $\mathsf{emb}\ (\mathsf{FNode}\ n\ as) = \mathsf{Node}\ n\ ((\mathsf{id} \oplus \mathsf{emb}) \bullet as)$;
- corecursively: unfold froot fcont, yielding the equations $\mathsf{root}\ (\mathsf{emb}\ t) = \mathsf{froot}\ t$ and $\mathsf{cont}\ (\mathsf{emb}\ t) = (\mathsf{id} \oplus \mathsf{emb}) \bullet \mathsf{fcont}\ t$.

As an instance of the situation from Appendix A, the image of emb is the set of finite dtree's, $\{t \in \mathsf{dtree}.\ \mathsf{ftree}\}$.

D. Proofs

For more details on some of the proofs, we refer the reader to our Isabelle formalization [8], which employs essentially the same notations as this text.

Proof of Theorem 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

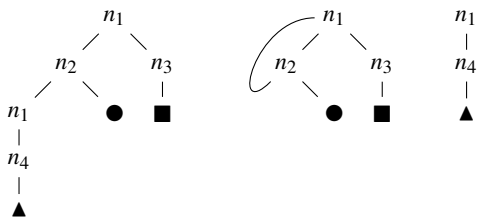

Fig. 3. A finite derivation tree (left), a regular cut of it (middle), and a finite regular cut of it (right)

$\mathsf{Fr}\ (\mathsf{H}\ n) \subseteq \mathsf{Fr}\ t_0$ follows by routine fixpoint induction on the definition of Fr (since at each node $n' \in \mathsf{ltr}\ (\mathsf{H}\ n)$ 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 $n' \in \mathsf{ltr}\ (\mathsf{H}\ n)$ we have the production of $\mathsf{pick}\ n'$). □

Proof of Theorem 2: 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., $\overline{\alpha}\ \mathsf{H} \neq \emptyset$, as desired.

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 $J = \{j \in [n].\ \overline{\beta}\ \mathsf{F}_j \neq \emptyset\}$. 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 $(f_j : \overline{\beta}\ \mathsf{F}_j \to \gamma_j)_{i\in[n]}$. Then, thanks to $\mathsf{Gmap}\ \overline{f} : \overline{\beta}\ \overline{\mathsf{F}}\ \mathsf{G} \to \overline{\gamma}\ \mathsf{G}$, by (1) we obtain that $\overline{\gamma}\ \mathsf{G} \neq \emptyset$, as desired.

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 the 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, as desired. □

Proof of Theorem 3: Immediate from the definitions. □

In some of the following proofs we exploit an embedding of datatypes as finite codatatypes (Appendix A). 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. Figure 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 $\mathsf{fpick} : \mathsf{ltr}\ t_0 \to \mathsf{Subtr}\ t_0$ similarly to pick from Sect. 2.3, 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 1. Assume t_0 is a finite derivation tree. Then:

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

Proof: (1) Similar to the proof of Theorem 1. (2) By routine induction on t_0. □

Proof of Theorem 5: 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 $j \in \mathsf{ltr}\ t_0$, 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 $\exists j \in \mathsf{ltr}\ t_0.\ t = t_j$, 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, $\overline{\alpha}\ \mathsf{JF}_{j_0} \neq \emptyset$, as desired.

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 $J = \{j' \in [n].\ \overline{\alpha}\ \mathsf{IF}_{j'} \neq \emptyset\}$. 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 $K \subseteq I \cup \{m + j'.\ j' \in J\}$. We take t to have j as root, $I \cap K$ as leaves and $(t_{j'})_{j'\in J}$ as immediate subtrees; namely, $t = \mathsf{Node}\ j\ ((\mathsf{Inl} \bullet I) \cup (\mathsf{Inr} \bullet \{t_{j'}.\ j' \in J\}))$.

Let t_0 be a tree as above corresponding to j_0 (since $\overline{\alpha}\ \mathsf{IF}_{j_0} \neq \emptyset$). Then, by Lemma 1, $t_1 = \mathsf{rcut}\ t_0$ 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 $I_1 \subseteq I$, as desired. □

Proof of Theorem 7: $\mathscr{L}^{\mathrm{r}}_{\mathsf{Gr}}\ ns\ n \subseteq \{\emptyset\}$, 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.

To prove the right-to-left direction, let $ss \in (\mathsf{T} + \mathsf{N})\ \mathsf{fset}$ 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 $\{U\ n'.\ n' \in \mathsf{Inr}^- ss\}$, namely, $\mathsf{Node}\ n\ ((\mathsf{id} \oplus U) \bullet ss)$, is the desired regular derivation tree whose frontier is included ts. □

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 by the following rules:

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

Lemma 2. 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. □

Proof of Theorem 8: According to Lemma 2 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. □

E. Registration of a New BNF in Isabelle

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

```
bnf_def
  bag_map :: ('a => 'b) => 'a bag => 'b bag
  [set_of :: 'a bag => 'a set]
  natLeq :: (nat * nat) set
  [{#} :: 'a bag]
  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 definition is not shown here):

- the functorial action, bag_map;
- the "Fset" operation discussed in Sect. 3.3, set_of;
- a cardinal bound, here, that of natural numbers, natLeq (in Isabelle, cardinals are represented as minimal well-order relations);
- a witness term, here, the empty bag $\{\#\}$;
- a customized relator, as discussed in Sect. 3.4, bag_rel.

Then the user is prompted to prove a few facts, including one saying that set_of is a natural transformation:

```
set_of o bag_map f = image f o set_of
```

Upon discharging these goals, the α bag BNF is registered.