Text S1

This text provides additional definitions and explanations to help clarify the ideas presented in the main text, for readers unfamiliar with a category theory approach. We use standard category theory constructions and methods that can be found in general introductions to category theory (see, e.g., [1,2]), and introductions tailored for a categorical treatment of recursion (see [3–5]). The value added by this account of the material lies in the fact that it is self-contained, and contains only what is relevant to our applications of these concepts. Proofs of some well-known basic category theory results are included, as they help convey an understanding of our category-theoretic explanation of systematicity. For convenience, definitions that appear in the main text have been reproduced here.

Contents

Background ................................................................. 2
  Categories ............................................................... 2
  Duals ................................................................... 4
  Isomorphisms ............................................................. 5
  (Co)Products ............................................................... 6
  Functors .................................................................. 8
  Natural transformations ............................................. 11

F-algebras, initial algebras, and catamorphisms ........................................... 11
  Numbers .................................................................. 14
  Lists ..................................................................... 17
  Trees ..................................................................... 24
  Relation between numbers, lists, and trees ................................. 27
    Numbers from lists .................................................. 27
    Lists from trees ..................................................... 29
    Numbers from trees ............................................... 31
    No natural equivalences .......................................... 31
Background

In this section, we provide the background category theory concepts on which the concept of an \( F \)-algebra and universal construction are based.

Categories

Definition (Category, object, morphism, domain, codomain, composition). A category \( C \) consists of a class of objects \( |C| = (A, B, \ldots) \); and for each pair of object \( A, B \) in \( C \), a set \( C(A, B) \) of morphisms (also called arrows, or maps) from \( A \) to \( B \) where each morphism \( f : A \to B \) has \( A \) as its domain and \( B \) as its codomain, including the identity morphism \( 1_A : A \to A \) for each object \( A \); and a composition operation, denoted “\( \circ \)”, of morphisms \( f : A \to B \) and \( g : B \to C \), written \( g \circ f : A \to C \) that satisfies the laws of:

- identity, where \( f \circ 1_A = f = 1_B \circ f \), for all \( f : A \to B \); and
- associativity, where \( h \circ (g \circ f) = (h \circ g) \circ f \), for all \( f : A \to B \), \( g : B \to C \) and \( h : C \to D \).

We briefly describe several relevant categories, then provide the definition of the category we use as the basis for our explanation of systematicity. A common first example of a category is \( \textbf{Set} \) whose objects are sets, and morphisms are total functions. Another, set-like, category is \( \textbf{Pos} \), the category of partially ordered sets \( (P, \leq) \), called posets, and monotone functions. A poset is a set \( P \) together with a binary relation \( \leq \) defined on \( P \) that is reflexive \( (a \leq a) \), transitive \( (x \leq y \text{ and } y \leq z \text{ implies } x \leq z) \), and antisymmetric \( (x \leq y \text{ and } y \leq x \text{ implies } x = y) \). A monotone function \( f : A \to B \) is order-preserving, i.e., \( x \leq y \text{ implies } f(x) \leq f(y) \). For our purposes, we work in the category \( \textbf{CPO} \) of complete pointed partial
ordered sets and continuous functions (see [6] for an introduction, and [7] in the context of functional programming).

**Definition (CPO).** Let \((P, \leq)\) be a poset and \(S\) any subset of \(P\). An upper bound on \(S\) is an element \(x \in P\) such that for all \(s \in S\) \(s \leq x\). The set of upper bounds for \(S\) is denoted \(UB(S)\), and \(UB(\emptyset) = P\). The least upper bound, if it exists, is the least element of \(UB(S)\), denoted \(LUB(S)\). A poset \((P, \leq)\) is called complete if for every subset \(S \subseteq P\) there exists a least upper bound \(LUB(S)\). A poset \((P, \leq)\) is called pointed if there exists a least element, denoted \(\bot \in P\), such that \(\bot \leq p\) for all \(p \in P\). An \(\omega\)-sequence in a poset \((P, \leq)\) is a subset \(\{p_i\mid 0 \leq i, p_i \leq p_{i+1}\}\). A partially ordered set \((P, \leq)\) in which every \(\omega\)-sequence has a least upper bound is called \(\omega\)-complete, and if \((P, \leq)\) also has a least element, then \((P, \leq)\) is called a \(\omega\)-complete pointed partially ordered set. An \(\omega\)-complete pointed partially ordered set is called a CPO for short. A function \(f : P \to Q\) between CPOs \(P\) and \(Q\) is monotonic whenever \(p_i \leq p_j\) implies \(f(p_i) \leq f(p_j)\) for all \(p_i, p_j \in P\); and \(f\) is continuous whenever each sequence \(\{p_i\mid 0 \leq i\} \subseteq P\) is a sequence \(\{f(p_i)\mid 0 \leq i\} \subseteq Q\). Hence, continuous functions preserve LUBs and least elements. The category CPO has CPOs for objects and continuous functions for morphisms such that the set of functions \(\{f : A \to B\}\) for each pair of objects \(A\) and \(B\) in CPO is a complete partially ordered set, and composition of functions preserves ordering: functions are ordered pointwise, i.e., \(f_i \leq f_j\) whenever \(f_i(a) \leq f_j(a)\) for all \(a \in A\).

The importance of CPO is with regard to “fixed-points” and structural induction, which we do not get from Set. We just provide the intuition here (see [3,7] for formal details). A function \(f : X \to X\) has a fixed point when \(f(x) = x\) for some \(x \in X\). Analagously, a functor \(f : C \to C\) has a “fixed point” when \(f(A) \cong A\). The presence of LUBs means that under certain general conditions recursive applications are guaranteed to converge. The presence of CPOs means that there is a continuous function from the natural numbers to each of its sequences starting from the least element. Hence, structural induction is based on induction over the natural numbers. The proofs that the various (un)folks are (anamorphisms) catamorphisms requires structural induction, and the presence of (final) initial \(F\)-(co)algebras depends of the functors \(F\) having fixed points. Our explanation for systematic recursive capacities in terms of these forms of universal constructions depends on CPO.

The partial order underlying our various applications of CPO is called approximation ordering, and is denoted \(\sqsubseteq\) (see [7]). For sets (e.g., numbers, characters, etc.), approximation ordering is defined by:

\[x \sqsubseteq y \iff (x = \bot) \lor (x = y)\]
For pairs, it is defined by:

\[ \bot \sqsubseteq (x, y) \]

and

\[ (x, y) \sqsubseteq (x', y') \iff (x \sqsubseteq x') \land (y \sqsubseteq y') \]

For lists, it is defined by:

\[ \bot \sqsubseteq xs \]

and

\[ \emptyset \sqsubseteq xs \iff xs = \emptyset \]

\[ (x \cdot xs) \sqsubseteq (y \cdot ys) \iff (x \sqsubseteq y) \land (xs \sqsubseteq ys) \]

The bottom element \( \bot \) is interpreted as the “undefined”, or “unknown” value. Since morphisms in CPO preserve points, partially defined functions are effectively total functions in this category. The element \( \bot \) is also used for partially known information: e.g., a partially completed guest list is constructed as: \( \text{John} \cdot \text{Mary} \cdot \text{Sue} \cdot \bot \). Moreover, by the definition of \( \sqsubseteq \), we have, e.g., \( \text{John} \cdot \text{Mary} \cdot \bot \sqsubseteq \text{John} \cdot \text{Mary} \cdot [\text{Sue}] \), since \( \bot \sqsubseteq xs \) for all \( xs \) in the set of lists, which accords with our intuition that the second list provides more information about the coming guests than the first. Structural induction over lists, for example, proceeds by proving one or more base cases, typically involving \( \bot \) and \( \emptyset \), assuming an arbitrary list \( xs \), and then proving the induction step, \( x \cdot xs \). So, the sequence \( \bot \leq \emptyset \leq \ldots xs \leq x \cdot xs \) corresponds to \( 0 \leq 1 \leq \ldots n \leq n + 1 \), assuming an approximation ordering via list reversal: i.e., \( sx \sqsubseteq sy \iff xs \sqsubseteq ys \), where \( sx \) (\( sy \)) is \( xs \) (\( ys \)) in reverse order. Structural induction over trees proceeds analogously, noting that tree order is related by pair order. Since \( \bot \) is preserved we omit proofs involving this case.

**Duals**

Category theory constructs often come in pairs by reversing the direction of the relevant arrows.

**Definition (Opposite category).** Given a category \( \mathbf{C} \), the *opposite category* \( \mathbf{C}^{\text{op}} \) is the category whose objects are all the objects of \( \mathbf{C} \), and whose morphisms are obtained by reversing the directions of the arrows in \( \mathbf{C} \).

**Definition (Dual).** Given a definition, proposition, or proof in a category \( \mathbf{C} \), its dual is the corresponding definition, proposition, or proof in the category \( \mathbf{C}^{\text{op}} \).
The following examples are pairs of dual constructs: (initial object, terminal object), (coproduct, product), and (couniversal morphism, universal morphism). An isomorphism is self-dual. $F$-algebra, initial algebra and catamorphism also have duals called $F$-coalgebra, final coalgebra and anamorphism (respectively), which we detail later.

**Definition (Initial object).** An initial object in a category $C$ is an object, denoted $0$, such that for every object $A$ in $C$ there exists a unique morphism $u : 0 \rightarrow A$ in $C$.

**Example (Initial object).** In $\text{Set}$ and $\text{CPO}$, the empty set $\emptyset$ is the initial object, and the unique morphism from $\emptyset$ to $A$ is the empty function.

**Definition (Terminal object).** A terminal object in a category $C$ is an object, denoted $1$, such that for every object $A$ in $C$ there exists a unique morphism $u : A \rightarrow 1$ in $C$.

**Example (Terminal object).** In $\text{Set}$, any singleton set $\{x\}$ is a terminal object, and the unique morphism from $A$ to $\{x\}$ is the function that maps every element $a \in A$ to $x$. In this case, there are many terminal objects. In $\text{CPO}$, the terminal object is the set $\{\bot\}$, for which $\bot$ is also the LUB.

**Isomorphisms**

Universal constructions are said to be unique up to a unique isomorphism. Here we explain what this expression means.

**Definition (Isomorphism).** A morphism $f : A \rightarrow B$ is an isomorphism if and only if there exists a morphism $g : B \rightarrow A$, such that $g \circ f = 1_A$ and $f \circ g = 1_B$. If $g$ exists, then it is said to be the inverse of $f$ (also denoted $f^{-1}$). If $f : A \rightarrow B$ is an isomorphism, then $A$ is said to be isomorphic to $B$, written $A \cong B$.

In category theory, where the focus is on the relationships between objects (i.e., the morphisms), rather than the specifics of the objects themselves, one often only requires objects to be identified up to an isomorphism. That is, isomorphic objects are regarded as essentially the “same” object. For example, in a category $C$ with terminal objects, all such objects are isomorphic (as proved below), and so are regarded as the same object, denoted by 1. In $\text{Set}$, where every singleton set is a terminal object, “the” terminal object is often denoted as $\{\ast\}$, where $\ast$ is an unnamed object, when we don’t need its identity. Obviously, the isomorphism from singleton set $\{a\}$ to singleton set $\{b\}$ is the unique map $f : a \leftrightarrow b$, which has the inverse $g : b \leftrightarrow a$. In general, any two sets with the same number of objects are isomorphic. However, for $n > 1$, the isomorphism is not unique. For example, for sets $\{a_1, a_2\}$ and $\{b_1, b_2\}$, the
map \( f_1 : a_1 \mapsto b_1, a_2 \mapsto b_2 \) is an isomorphism, and so too is the map \( f_2 : a_1 \mapsto b_2, a_2 \mapsto b_1 \), though \( f_1 \neq f_2 \). Hence, we have the distinction between unique up to an isomorphism, and unique up to a unique isomorphism.

**Proposition** Terminal objects are unique up to a unique isomorphism.

**Proof** Suppose \( T \) and \( T' \) are two terminal objects. By definition (of terminal objects) there must exist unique morphisms \( u : T' \to T \) and \( v : T \to T' \). Consider the composition \( v' \circ u : T \to T' \). Since every object must have an identity map, this composition can only be the identity, i.e., \( v' \circ u = 1_T \). Exchanging the roles of \( u \) and \( v \) yields \( u \circ v = 1_{T'} \) by the same argument. Therefore \( u \) is an isomorphism. Since \( u \) is unique, it is a unique isomorphism, and likewise for \( v \).

**Proposition** Initial objects are unique up to a unique isomorphism.

**Proof** Duality!—replace the word terminal by initial, and turn all the arrows around, in the proof above, and you have a proof that initial objects are unique up to a unique isomorphism.

(\textbf{C}o)\textbf{P}roducts

**Definition (Product of objects).** A product of objects \( A \) and \( B \) in category \( C \) is, up to unique isomorphism, an object \( P \) (also denoted \( A \times B \)) together with two morphisms (sometimes called projections) \( p_1 : P \to A \) and \( p_2 : P \to B \), jointly expressed as \( (P,p_1,p_2) \), such that for every object \( Z \in |C| \) and pair of morphisms \( f : Z \to A \) and \( g : Z \to B \) there exists a unique morphism \( u : Z \to P \), also denoted \( (f,g) \), such that the following diagram commutes:

![Diagram](1)

By a straightforward extension, the finite product of \( n \) objects \( A_1, \ldots, A_n \) is \( (A_1 \times \cdots \times A_n, p_1, \ldots, p_n ) \).

**Example (Product).** The familiar Cartesian product of sets has this property in \( \text{Set} \). In \( \text{CPO} \), the product of two \( \text{CPOs} \) \( (A, \leq) \) and \( (B, \leq) \) is the Cartesian product \( (A \times B) \) of their underlying sets, where \( (\bot, \bot) \) is the bottom element, together with the partial order defined for pairs: i.e., \( (a, b) \subseteq (a', b') \) if \( (a \subseteq a') \land (b \subseteq b') \) for all \( a, a' \in A \) and \( b, b' \in B \). Not all categories have (all) products.

**Definition (Coproduct of objects).** A coproduct of objects \( A \) and \( B \) in category \( C \) is, up to unique
Categorial compositionality III

isomorphism, an object \( Q \) (also denoted \( A + B \)) together with two morphisms \( q_1: A \to Q \) and \( q_2: B \to Q \), jointly expressed as \( (Q, q_1, q_2) \), such that for every object \( Z \in |C| \) and pair of morphisms \( f: A \to Z \) and \( g: B \to Z \) there exists a unique morphism \( u: Z \to P \), also denoted \( [f, g] \), such that the following diagram commutes:

\[
\begin{array}{ccc}
A & \xrightarrow{q_1} & Q \\
\downarrow{f} & & \downarrow{[f, g]} \phantom{00} & B \\
\phantom{f} & & \phantom{[f, g]} \\
\end{array}
\]

By a straightforward extension, the finite coproduct of \( n \) objects \( A_1, \ldots, A_n \) is \((A_1 + \cdots + A_n, q_1, \ldots, q_n)\).

Example (Coproduct). In \( \text{Set} \), the coproduct (up to a unique isomorphism) is disjoint union. The disjoint union of two sets \( A \) and \( B \), denoted \( A \uplus B \), is the set \( \{(1, a) | a \in A\} \cup \{(2, b) | b \in B\} \). In \( \text{CPO} \), the coproduct is the disjoint union, defined analogously as the disjoint union of the underlying sets (with their respective bottom elements removed), plus the bottom element, i.e., \( \{\bot\} \cup (A^+ \uplus B^+) \), where \( A^+ \) and \( B^+ \) are sets \( A \) and \( B \) depleted of their bottom elements. The associated partial order is: \( \bot \subseteq (1, a) \), \( \bot \subseteq (2, b) \), and \( a \subseteq a' \Rightarrow (1, a) \subseteq (1, a') \) and \( b \subseteq b' \Rightarrow (2, b) \subseteq (2, b') \) for all \( a, a' \in A^+ \) and \( b, b' \in B^+ \).

Not all categories have (all) coproducts.

Definition (Product of morphisms). A product of morphisms \( f_1: A_1 \to B_1 \) and \( f_2: A_2 \to B_2 \) is the unique morphism \( f_1 \times f_2: A_1 \times A_2 \to B_1 \times B_2 \) such that the following diagram commutes:

\[
\begin{array}{ccc}
A_1 & \xrightarrow{p_1} & A_1 \times A_2 \\
\downarrow{f_1} & & \downarrow{f_1 \times f_2} \phantom{00} & A_2 \\
B_1 & \xleftarrow{p'_1} & B_1 \times B_2 \\
\end{array}
\]

Such a unique morphism is guaranteed to exist by the definition of products (if the products exist). Finite products of \( n \) morphisms, denoted \( f_1 \times \cdots \times f_n: A_1 \times \cdots \times A_n \to B_1 \times \cdots \times B_n \), are also defined by a straight-forward extension.

Definition (Coproduct of morphisms). A coproduct of morphisms \( f_1: A_1 \to B_1 \) and \( f_2: A_2 \to B_2 \)
is the unique morphism $f_1 + f_2 : A_1 + A_2 \to B_1 + B_2$ such that the following diagram commutes:

\[\begin{array}{ccc}
A_1 & \xrightarrow{q_1} & A_1 + A_2 & \xleftarrow{q_2} & A_2 \\
\downarrow{f_1} & & \downarrow{f_1 + f_2} & & \downarrow{f_2} \\
B_1 & \xrightarrow{q_1'} & B_1 + B_2 & \xleftarrow{q_2'} & B_2
\end{array}\] (4)

Such a unique morphism is guaranteed to exist by the definition of coproducts (if the coproducts exist). Finite coproducts of $n$ morphisms, denoted $f_1 + \cdots + f_n : A_1 + \cdots + A_n \to B_1 + \cdots + B_n$, are also defined by a straight-forward extension.

**Functors**

**Definition (Functor).** A functor $F : C \to D$ is a map from a category $C$ to a category $D$ that associates each object $A$ in $C$ with an object $F(A)$ in $D$; and each morphism $f : A \to B$ in $C$ with a morphism $F(f) : F(A) \to F(B)$ in $D$, and is structure-preserving in that $F(1_A) = 1_{F(A)}$ for each object $A$ in $C$; and $F(g \circ f) = F(g) \circ F(f)$ for all morphisms $f : A \to B$ and $g : B \to C$.

**Example (Functor).** A common example of a functor is a “forgetful” functor, such as the functor $U : \text{Vec} \to \text{Set}$, which maps a vector space to its underlying set, and a vector space morphism (that is, a linear mapping) to the underlying function. The following definitions give further examples of functors.

**Definition (Isofunctor).** A functor $F : C \to D$ is an isofunctor if and only if there exists a functor $G : D \to C$ such that $G \circ F = 1_C$ and $F \circ G = 1_D$, where $1_C$ and $1_D$ are identity functors.

**Definition (Constant functor).** A constant functor $K_D : C \to D; C \mapsto D; f \mapsto 1_D$ maps all objects $C$ and morphisms $f$ in $C$ to the same object $D$ and identity morphism $1_D$ in $D$.

**Definition (Identity functor).** An identity functor $I : C \to C; C \mapsto C; f \mapsto f$ maps every object $C$ and morphism $f$ in $C$ to itself.

A (co)product of functors is defined, because of its close relationship to polynomial functors, which form the basis of the $F$-algebras we use in our examples of recursive domains.

**Definition (Product of functors—pointwise).** A pointwise (finite) product of functors $F_i : C \to D$,
\[ i \in \{1, \ldots, n\} \] is the functor \( F_1 \times \cdots \times F_n \) indicated in the following diagram:

\[ \begin{array}{c}
C \xrightarrow{F_1 \times \cdots \times F_n} D \\
\downarrow \quad \downarrow \\
C_1 \xrightarrow{F_1(C_1) \times \cdots \times F_n(C_1)} D \\
\downarrow f \quad \downarrow F_1(f) \times \cdots \times F_n(f) \\
C_2 \xrightarrow{F_1(C_2) \times \cdots \times F_n(C_2)} D \\
\end{array} \] (5)

where \( F_i(C_i) \) has a set-like structure, and the product in \( D \) coincides with the Cartesian product of sets, which is the case for the categories that we use.

**Definition (Coproduct of functors—pointwise).** A pointwise (finite) coproduct of functors \( F_i : C \to D, i \in \{1, \ldots, n\} \) is the functor \( F_1 + \cdots + F_n \) indicated in the following diagram:

\[ \begin{array}{c}
C \xrightarrow{F_1 + \cdots + F_n} D \\
\downarrow \quad \downarrow \\
C_1 \xrightarrow{F_1(C_1) + \cdots + F_n(C_1)} D \\
\downarrow f \quad \downarrow F_1(f) + \cdots + F_n(f) \\
C_2 \xrightarrow{F_1(C_2) + \cdots + F_n(C_2)} D \\
\end{array} \] (6)

where \( F_i(C_i) \) has a set-like structure, and the coproduct in \( D \) coincides with the disjoint union of sets, which is the case for the categories that we use.

**Definition (Composition of functors).** Given functors \( F : C \to D \) and \( G : D \to E \), the composition of functors \( G \circ F : C \to E \) is given by \( G \circ F : C \mapsto G(F(C)), f \mapsto G(F(f)) \).

**Definition (Cat).** The category \( \text{Cat} \) has small categories for objects and functors for morphisms. (The technical distinction between small categories and otherwise need not concern us here, see [1].) Hence, composition is functor composition.

**Definition (Endofunctor).** An endofunctor \( F : C \to C \) is a functor whose domain and codomain are the same category \( C \).
Definition (Polynomial functor). A polynomial functor is a constant functor, an identity functor, or a (co)product or composition of polynomial functors. Any polynomial functor is equivalent to one of the form \( \sum_{i=0}^{n} K_{C_i} \times I^i \cong K_{C_0} + K_{C_1} \times I + K_{C_2} \times I^2 + \cdots + K_{C_n} \times I^n \), where \((+)\times\) is the (co)product of functors operator, and \(I^2\) is equivalent to \(I \times I\), etc. (Note that \(K_{C_0} \times I^0 \cong K_{C_0}\).)

A product of categories is also defined, because it is needed for \(F\)-algebras.

Definition (Product of categories). Given categories \(C\) and \(D\), the product of categories \(C \times D\) has pairs \((C; D)\) as objects and pairs \((f; g) : (C_1, D_1) \to (C_2, D_2)\) as morphisms, where \(C\) and \(f : C_1 \to C_2\) are in \(C\), and \(D\) and \(g : D_1 \to D_2\) are in \(D\). Identity morphisms and composition are component-wise.

It is routine to check that \(C \times D\) is a category, and that \((C \times D, F_1, F_2)\) is a product in \(\text{Cat}\), where functors (projections) \(F_1\) and \(F_2\) return the first and second objects (morphisms) of each pair of objects (morphisms) in \(C \times D\).

Definition (Bifunctor). A bifunctor \(F(-, -) : C \times D \to E\) is a functor with two arguments, mapping pairs of objects and pairs of morphisms from a product of categories \(C \times D\) to objects and morphisms in a category \(E\). A bifunctor fixed at first-argument object \(A\) in \(C\) is a functor of the form \(F_A = F(A, -) : D \to E; D \mapsto F(A, D), g \mapsto F(1_A, g)\), where \(D\) is an object and \(g : D_1 \to D_2\) is a morphism in \(D\).

Definition (Parameterized functor). Given a bifunctor \(F : C \times D \to E\) and an object \(A\) in \(C\), a parameterized functor is the functor \(F_A : D \to E\) of the form \(F(A, -)\) defined above.

Example (Parameterized functor). An example of a functor \(F : \text{Set} \times \text{Set} \to \text{Set}\) parameterized by \(A\) is the polynomial functor \(F_A = K_1 + K_A \times I : \text{Set} \to \text{Set}, S \mapsto 1 + A \times S, f \mapsto 1_l + 1_A \times f\). This functor is used to recursively constructs sets of lists of type \(A\), starting with the singleton set \([[]]\) containing the empty list \([]\). (Recall that in \(\text{Set}\), any singleton set is a terminal object, denoted \(1\).) So, for example, \(F_A(1) = K_1(1) + K_A(1) \times I(1) = 1 + A \times 1 \cong 1 + A\) (essentially, corresponding to a set containing the empty list and all one-element lists). Since \(1 + A\) is in the domain of \(F_A\) we also have \(F_A(1 + A) = 1 + A \times (1 + A) \cong 1 + A + A \times A\) (essentially, corresponding to a set containing the empty list, all one-element lists, and all two-element lists). In this way, \(F_A\) constructs lists of arbitrary lengths. Functions over these lists are similarly constructed recursively. However, their relevance to recursive domains requires additional definitions, so examples are deferred to the next section.
Natural transformations

Definition (Natural transformation). A natural transformation \( \eta : F \rightarrow G \) between a domain functor \( F : C \rightarrow D \) and a codomain functor \( G : C \rightarrow D \) consists of \( D \)–maps \( \eta_A : F(A) \rightarrow G(A) \) for each object \( A \) in \( C \), such that \( G(f) \circ \eta_A = \eta_B \circ F(f) \), as indicated by commutative diagram

\[
\begin{array}{ccc}
F(A) & \xrightarrow{\eta_A} & G(A) \\
\downarrow{F(f)} & & \downarrow{G(f)} \\
F(B) & \xrightarrow{\eta_B} & G(B)
\end{array}
\]  

Definition (Natural isomorphism/natural equivalence). A natural transformation is a natural isomorphism, or natural equivalence if and only if each \( \eta_A \) is an isomorphism. That is, for each \( \eta_A : F(A) \rightarrow G(A) \) there exists a \( \eta_A^{-1} : G(A) \rightarrow F(A) \) such that \( \eta_A^{-1} \circ \eta_A = 1_{F(A)} \) and \( \eta_A \circ \eta_A^{-1} = 1_{G(A)} \).

Natural transformations are morphisms of functors, and so can be composed. Note that isomorphic functor and isofunctor are not the same thing. Isomorphic functors are functors that are isomorphic to each other and the map between them is a natural equivalence. Isofunctors are maps between isomorphic categories. We will provide examples of natural transformations when we investigate the relationship between functors that underlie different recursive domains.

\( F \)-algebras, initial algebras, and catamorphisms

Here we provide the definition of a category of \( F \)-algebras. Then, we build up category-theoretic models in terms of \( F \)-algebras for three recursive domains involving numbers, lists and trees. In each case, we start with a (bi)functor, possibly parameterized by an object identifying the type of the elements to which the corresponding operations apply, yielding a polynomial functor \( F \) that specializes to operations pertaining to the respective domains.

Definition \( (F \)-algebra). For an endofunctor \( F : C \rightarrow C \), an \( F \)-algebra is a pair \( (A, \alpha) \), where \( A \) is an object and \( \alpha : F(A) \rightarrow A \) is a morphism in \( C \).

Example \( (F \)-algebra). Let \( C \) be the category \textbf{Set}, \( F : \textbf{Set} \rightarrow \textbf{Set} \) be the functor \( F : S \mapsto S \times S, f \mapsto f \times f \), \( A \) be the set of natural numbers \( \mathbb{N} \) (so, \( \mathbb{N} \in |\textbf{Set}| \)), and \( \alpha : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N} \) be the morphism \( \alpha : (m, n) \mapsto m + n \). Then, \((\mathbb{N}, \alpha)\) is an \( F \)-algebra.
**Definition (F-algebra homomorphism).** An *F-algebra homomorphism* \( h : (A, \alpha) \to (B, \beta) \) is a morphism \( h : A \to B \) (in \( C \)) such that the following diagram commutes:

\[
\begin{array}{ccc}
F(A) & \xrightarrow{\alpha} & A \\
\downarrow F(h) & & \downarrow h \\
F(B) & \xrightarrow{\beta} & B \\
\end{array}
\]

**Example (F-algebra homomorphism).** Let \((A, \alpha)\) be the \( F \)-algebra \((\mathbb{N}, \alpha)\) from the previous example, \((B, \beta)\) be the \( F \)-algebra \((\mathbb{N}, \beta)\), where \( \beta : (n, m) \mapsto nm \) (i.e., multiplication), and \( h : \mathbb{N} \to \mathbb{N} \) be the power function \( h : n \mapsto 2^n \). Then, \( h \) is an \( F \)-algebra homomorphism (since \( 2^{n+m} = 2^n 2^m \)).

\( F \)-algebra homomorphisms can be composed, and there are identity \( F \)-algebra homomorphisms. Hence, the collection of \( F \)-algebras and \( F \)-algebra homomorphisms is a category.

**Definition (Category of F-algebras).** For any endofunctor \( F : C \to C \), the \( F \)-algebras form the category \( \text{Alg}(F) \) with \( F \)-algebras \((A, \alpha)\) for objects, and \( F \)-algebra homomorphisms \( h : (A, \alpha) \to (B, \beta) \) for morphisms.

**Definition (Initial algebra).** An *initial \( F \)-algebra* \((A, \text{in})\), hereafter simply called an *initial algebra*, is an initial object in the category of \( F \)-algebras \( \text{Alg}(F) \). That is, there exists a unique \( F \)-algebra homomorphism from \((A, \text{in})\) to every \( F \)-algebra in \( \text{Alg}(F) \).

The presence of an initial algebra in a category \( \text{Alg}(F) \) depends on the particular functor \( F \), and is not guaranteed to exist in every case.

**Proposition.** If \( F : C \to C \) is a polynomial functor, then the category of \( F \)-algebras \( \text{Alg}(F) \) has an initial algebra.

**Proof.** See [3], chapters 10 and 11.

**Definition (Catamorphism).** A *catamorphism* \( h : (A, \text{in}) \to (B, \beta) \) is the unique \( F \)-algebra homomorphism from initial \( F \)-algebra \((A, \text{in})\) to \( F \)-algebra \((B, \beta)\). That is, \( h \circ \text{in} = \beta \circ F(h) \) and the uniquely specified \( h \) for each such \( \beta \) is denoted \( \text{cata} \beta \) (i.e., \( h = \text{cata} \beta \)), as indicated in the following diagram:

\[
\begin{array}{ccc}
F(A) & \xrightarrow{\text{in}} & A \\
\downarrow F(\text{cata} \beta) & & \downarrow \text{cata} \beta \\
F(B) & \xrightarrow{\beta} & B \\
\end{array}
\]
where cata $\beta$ is also denoted $\langle \beta \rangle$ using banana brackets (see [4]).

The following result, due to Lambek [2], is important to our explanation of systematicity.

**Lemma (Lambek).** Given a functor $F : C \to C$ as the basis for a category of $F$-algebras $\text{Alg}(F)$ with an initial algebra $(A, in)$, then $in : F(A) \to A$ is a unique isomorphism ($F(A) \cong A$).

**Proof.** We must show that there exists a morphism $h : A \to F(A)$ such that $in \circ h = 1_A$ (right inverse) and $h \circ in = 1_{F(A)}$ (left inverse), and that $in$ is unique. Right inverse: Consider the diagram

$$
\begin{array}{cccccc}
F(A) & \xrightarrow{in} & A \\
\downarrow{F(h)} & & \downarrow{h} \\
F(F(A)) & \xrightarrow{F(in)} & F(A) \\
\downarrow{F(in)} & & \downarrow{in} \\
F(A) & \xrightarrow{in} & A
\end{array}
$$

The existence of $h : A \to F(A)$ such that the upper inner square commutes is guaranteed, since $(F(A), F(in))$ is an $F$-algebra (by endofunctor $F$) and $(A, in)$ is an initial algebra in $\text{Alg}(F)$. The arrows in the lower inner square are already given, since $F$ is a functor. Hence the outer square commutes too. Thus, $in \circ h : A \to A$ is an $F$-algebra homomorphism. $1_A$ and $1_{F(A)}$ also make the outer square commute (see curved arrows). Since $(A, in)$ is an initial algebra, the $F$-algebra homomorphism from $A$ to $A$ is unique, so $in \circ h = 1_A$.

Left inverse: From the upper inner commutative square of Diagram 10, we have $h \circ in = F(in) \circ F(h)$ (since $F$ is a functor) $= F(in \circ h) = F(1_A) = 1_{F(A)}$. That is, $h \circ in = 1_{F(A)}$. So, $in$ is an isomorphism. Since $h$ is unique (being a catamorphism) and $in = h^{-1}$, $in$ must also be unique. That is, $in$ is the unique ($F$-algebra) isomorphism $F(A) \to A$.

Examples of initial algebras and associated catamorphisms are given in the subsequent sections. An initial algebra $(A, in)$ in $\text{Alg}(F)$ on a functor $F : C \to C$ is an example of a universal construction. Specifically, it is an instance of a couniversal morphism (see Diagrams 43). That is, given an object $X = F(A)$ in $C$ and a (identity) functor $F = I : C \to C$, $(A, in)$ is a couniversal morphism from $F(A)$ to $I$, since for every object $Y = B$ in $C$ and every morphism $f : X \to F(Y)$, which is the morphism $\beta \circ F(h) : F(A) \to \beta$ (in Diagram 9), there exists a unique morphism, by definition of catamorphism,
$h = \langle \beta \rangle : A \to B$ (see Diagram 9) such that the corresponding diagram

$$
\begin{array}{ccc}
A & \xrightarrow{T_F(A)} & C \\
\downarrow \beta & & \downarrow I \\
F(A) & \xrightarrow{\text{in}} & (F(A) \downarrow I)
\end{array}
$$

(11)

commutes.

Numbers

For computation with regard to numbers, we start with the polynomial functor $F = K_1 + I : \text{CPO} \to \text{CPO}; S \mapsto 1 + S, f \mapsto 1_1 + f$, where $K_1 : S \mapsto 1, f \mapsto 1_1$ is a constant functor respectively sending all objects $S$ and morphisms $f$ to terminal object $1$ and its identity morphisms $1_1$, and $I$ is the identity functor for $\text{CPO}$. In this case, the functor is not parameterized. The $F$-algebras for the category $\text{Alg}(F)$ are the pairs $(S, [I_v; f])$, where $S$ is a set, $[I_v; f] : 1 + S \to S$ is a function from coproduct (disjoint union) $1 + S$ to $S$, $I_v : 1 \to S$ is a nullary function (equivalently, a constant) returning $v \in S$, and $f : s \mapsto s'$ is a unary function. An initial algebra in this category is $(N, [\text{zero}; \text{succ}])$, where $N$ is a set that corresponds to the natural numbers ($\mathbb{N}$), $[\text{zero}, \text{succ}] : 1 + N \to N$, $\text{zero} : 1 \to N$ is a nullary function (equivalently, a constant) returning the element $\text{Zero} \in N$, and $\text{succ} : N \to N; n \mapsto \text{succ}(n)$ is a unary function returning the successor of element $n \in N$. Initial algebra $(N, [\text{zero}, \text{succ}])$ is an $F$-algebra $(S, [I_v, f])$, where $S$ is the set $N$, $I_v$ is the nullary function $\text{zero} = I_{\text{zero}}$, and $f$ is the unary function $\text{succ}$. Thus, $N$ contains $\text{Zero}, \text{succ}(\text{Zero}), \text{succ}(\text{succ}(\text{Zero})), \text{etc}$; and $\text{succ}(\text{Zero})$ corresponds to $1$, $\text{succ}(\text{succ}(\text{Zero}))$ to $2$, and so on. This characterization of the natural numbers based on $\text{Zero}$ and $\text{succ}$ relies on the well-known work of Peano.

There is a catamorphism from initial algebra $(N, [\text{zero}, \text{succ}])$ to each $F$-algebra $(S, [I_v, f])$ in $\text{Alg}(F)$ called $\text{foldN}$ (i.e., fold for $N$), defined as $\text{foldN} [I_v, f] : N \to S, \text{Zero} \mapsto v, \text{succ}(n) \mapsto \text{foldN} [I_v(f(n)), f] (n)$, where $f : S \to S$ is a unary function, and $v \in S$ is a constant. We also define $\text{foldN}' [I_v, f] : N \to S, \text{Zero} \mapsto v, \text{succ}(n) \mapsto f(\text{foldN}' [I_v, f] (n))$.

**Proposition.** $\text{foldN} [I_v, f] = \text{foldN}' [I_v, f]$. 
**Proof (by induction).** Proof by induction has a common pattern consisting of a proof for the base case (Base), an inductive assumption for an arbitrary non-base case (Assumption), and a proof of the case that succeeds the assumption case (Induction step). Since we are working in the category CPO, full induction also requires proving the cases that involve the bottom element ⊥ (see [7]). The proofs provided here are easily extended to such cases, so the details are omitted.

Base—Zero: Immediate.

Assumption—n: \(\text{fold}^N[I_v, f](n) = \text{fold}^N[I_v, f](n)\)

Induction step—\(\text{succ}(n)\):

\[
\begin{align*}
\text{fold}^N[I_v, f](\text{succ}(n)) &= \text{fold}^N[I_v, f] \circ [\text{zero}, \text{succ}](n) \quad \text{(composition)} \\
&= f(\text{fold}^N[I_v, f](n)) \quad \text{(F-algebra morphism, see Diagram 12)} \\
&= f(\text{fold}^N'[I_v, f](n)) \quad \text{(by the inductive assumption)} \\
&= \text{fold}^N'[I_v, f](\text{succ}(n)) \quad \text{(definition of fold}^N')
\end{align*}
\]

The condition for \(\text{fold}^N[I_v, f]\) to be an F-algebra morphism is indicated by the commutative diagram

\[
\begin{array}{ccc}
1 + N & \xrightarrow{[\text{zero}, \text{succ}]} & N \\
\downarrow_{1 + \text{fold}^N[I_v, f]} & & \downarrow_{\text{fold}^N[I_v, f]} \\
1 + S & \xrightarrow{[I_v, f]} & S
\end{array}
\]

(12)

Next, we show that \(\text{fold}^N'[I_v, f]\) (therefore, \(\text{fold}^N[I_v, f]\)) is a catamorphism. We prove for \(\text{fold}^N'[I_v, f]\), because the proof is slightly simpler than for \(\text{fold}^N[I_v, f]\).

**Proposition.** \(\text{fold}^N'[I_v, f]\) (and \(\text{fold}^N[I_v, f]\)) is a catamorphism.

**Proof (by induction).** First, we need to show that \(\text{fold}^N'[I_v, f]\) exists (i.e., Diagram 12 commutes). Then, we need to show that \(\text{fold}^N'[I_v, f]\) is unique (i.e., if \(g : N \to S\) is other F-algebra homomorphism, then \(g = \text{fold}^N'[I_v, f]\)).

Existence (commutativity):
Base—(1, *): i.e., where \((1, *) \in 1 + N\) is being mapped, as opposed to \((2, n)\) where \(n \in N\).

\[\leadsto: foldN'[l_v, f] \circ [zero, succ](1, *) = foldN'[l_v, f](Zero)\] (definition of \([zero, succ]\))

\[= v\] (definition of \(foldN'\))

\[\leadsto: [l_v, f] \circ (1_1 + foldN'[l_v, f])(1, *) = [l_v, f](1, *)\] (definition of \(1_1 + foldN'[l_v, f]\))

\[= v\] (definition of \([l_v, f]\))

Assumption—(2, \(n\)):

\[foldN'[l_v, f] \circ [zero, succ](2, n) = [l_v, f] \circ (1_1 + foldN'[l_v, f])(2, n)\] (Diagram 12)

\[foldN'[l_v, f](\text{succ}(n)) = f(foldN'[l_v, f](n))\] (LHS/RHS reductions, by def.)

Induction step—(2, \((\text{succ}(n))\)):

\[\leadsto: foldN'[l_v, f] \circ [zero, succ](2, (\text{succ}(n))) = foldN'[l_v, f](\text{succ}(\text{succ}(n)))\] (definition of \([zero, succ]\))

\[= f(foldN'[l_v, f](\text{succ}(n)))\] (definition of \(foldN'\))

\[= f(f(foldN'[l_v, f](n)))\] (by inductive assumption)

\[\leadsto: [l_v, f] \circ (1_1 + foldN'[l_v, f])(2, \text{succ}(n)) = [l_v, f](\text{succ}(\text{succ}(n)))\] (definition of \(1_1 + \ldots\))

\[= [l_v, f](f(foldN'[l_v, f](n)))\] (by inductive assumption)

\[= f(f(foldN'[l_v, f](n)))\] (definition of \([l_v, f]\))

Uniqueness: Suppose \(g\) is an \(F\)-algebra homomorphism \(g : N \to S\). The definition of \(F\)-algebra homomorphism forces \(g(\text{Zero}) = v\), and \(g(\text{succ}(n)) = f(g(n))\).

Base—\(\text{Zero}\): \(g(\text{Zero}) = v\), so \(g(\text{Zero}) = foldN'[l_v, f](\text{Zero})\).

Assumption—\(n\): \(g(n) = foldN[l_v, f](n)\).
Induction step—$\text{succ}(n)$:

\[
g(\text{succ}(n)) = f(g(n)) \quad \text{(definition of } g)\]
\[
= f(\text{foldN}'[IV, f](n)) \quad \text{(by the inductive assumption)}\]
\[
= \text{foldN}'[IV, f](\text{succ}(n)) \quad \text{(definition of } \text{foldN}')\]

Hence, $\text{foldN}[IV, f]$ is also a catamorphism. □

The two versions differ in their order of evaluation: $\text{foldN}$ evaluates its argument before the recursive call, effectively progressing from $\text{Zero}$ to argument $n$; $\text{foldN}'$ evaluates its argument after the recursive call, effectively progressing from $n$ to $\text{Zero}$.

We also define two versions of $\text{foldn}$ (i.e., fold for numbers) to work directly with natural numbers. The definitions and proofs (not shown) follow the same patterns as for the two versions of $\text{foldN}$. They are defined as $\text{foldn}[IV, f]: N \rightarrow S; 0 \mapsto v; (n + 1) \mapsto \text{foldn}[If(v), f](n)$, and $\text{foldn}'[IV, f]: N \rightarrow S; 0 \mapsto v; (n + 1) \mapsto f(\text{foldn}[IV, f](n))$.

**Examples**

The initial algebra $(N, [\text{zero}, \text{succ}])$ is a model for the natural numbers. For example, the elements $\text{Zero}, \text{succ}(\text{Zero}), \text{succ}(\text{succ}(\text{Zero})) \in N$ correspond to the elements $0, 1, 2 \in \mathbb{N}$, respectively. And, being an initial algebra, the catamorphisms from this algebra provide models for operations over natural numbers. For example, addition of two natural numbers is modeled as $\text{plusN}: N \times N \rightarrow N$, where $\text{plusN } m : n \mapsto \text{foldN}[I_m, \text{succ}](n)$. In effect, $\text{plusN}(m)(-)$ is a function that adds $m$ to its $(-)$ argument. For example, $\text{plusN}(\text{succ}(\text{Zero}))(\text{succ}(\text{Zero})) = \text{succ}(\text{succ}(\text{Zero}))$, which corresponds to $1 + 1 = 2$.

**Lists**

A list (e.g., constructed from the letters: ‘s’, ‘n’, ‘o’, and ‘c’) can be represented internally by a recursive construction that prepends items to the front of a list (as in “cons”), or by a recursive construction that postpends items to the back of a list (as in “snoc”). (By postpend we mean adding an item to the end of a list to create a new list, in contrast to append, which usually means adding a list to a list to create a new list in a functional programming language.) Each construction is an initial algebra in a particular category of $F$-algebras based on a specific functor. We provide both functors. Though
our focus is the more commonly used prepend construction, the results extend naturally (via a natural isomorphism) to the other version. The presence of multiple constructions raises the question, which we have alluded to in the main text, How does the cognitive system know which algebra to use? The process for constructing/destructing lists is intrinsically linked by a dual relationship between $F$-algebra and $F$-coalgebra. This relationship is detailed in the final section.

Prepend version: We start with a polynomial functor $F_A : \text{CPO} \to \text{CPO}$ parameterized by an object $A$ in $\text{CPO}$ as the basis for a category of algebras $\text{Alg}(F_A)$ on functor $F_A$. An initial algebra in this category is a model for lists of objects of type $A$, and the catamorphisms from this initial algebra are models for recursive computations over such lists. The functor $F_A = K_1 + K_A \times I : \text{CPO} \to \text{CPO}; S \mapsto 1 + A \times S, f \mapsto 1_1 + 1_A \times f$, where $K_A : S \mapsto A, f \mapsto 1_A$ is a constant functor respectively sending all objects $S$ to object $A$ and all morphisms $f$ to identity morphism $1_A$. The $F$-algebras in the category $\text{Alg}(F_A)$ on $F_A$ are the objects $(S, [l_v, f])$, where $[l_v, f]: 1 + A \times S \to S, 1_v : 1 \to S$ always returns $v \in S$, and $f : A \times S \to S, (a, s) \mapsto s'$ is an arbitrary binary function. An initial algebra in this category is $(L, [\text{empty}, \text{cons}])$, where $[\text{empty}, \text{cons}] : 1 + A \times L \to L$, $\text{empty} : 1 \to L$ returns the empty list $[]$, and $\text{cons} : A \times L \to L; (a, l) \mapsto a \cdot l$ returns the list with element $a \in A$ prepended ($\cdot$) to list $l \in L$. Initial algebra $(L, [\text{empty}, \text{cons}])$ is an $F$-algebra $(S, [l_v, f])$, where $S$ is the set of lists $L$ (of type $A$), $1_v$ is the function $\text{empty} = [\cdot]$, and $f$ is the binary function $\text{cons}$. (Historically, $\text{cons}$ stands for “construct”.)

Postpend version: This version is based on the polynomial functor $F_A = K_1 + I \times K_A : \text{CPO} \to \text{CPO}; S \mapsto 1 + S \times A, f \mapsto 1_1 + f \times 1_A$. The $F$-algebras in the category based on this functor are the objects $(S, [l_v, f])$, where $[l_v, f]: 1 + S \times A \to S$ and $f : S \times A \to S, (s, a) \mapsto s'$ is an arbitrary binary function. An initial algebra in this category is $(L, [\text{empty}, \text{snoc}])$, where $[\text{empty}, \text{snoc}] : 1 + L \times A \to L$, and $\text{snoc} : L \times A \to L; (l, a) \mapsto l \cdot a$ returns the list with element $a \in A$ postpended ($\cdot$) to list $l \in L$. For the remainder of this section, we focus on the prepend version of lists, which is more commonly used in functional programming. Nonetheless, the postpend version may be more natural for particular cognitive capacities, which we also discuss in this section. The formal relationship between the two versions is given in the next section, where we also present the relationships between other $F$-algebras.

Somewhat like $\text{foldN}/\text{foldN}'$ (foldl/foldl’) for numbers, there are two recursively defined functions for processing prepend-constructed lists. For readers familiar with the functional programming language Haskell [8], they are called $\text{foldl}$ (i.e., fold lists from the right/front) and $\text{foldr}$ (i.e., fold lists from the left/back). Unlike the catamorphisms for number, though, only $\text{foldr}$ is a general catamorphism.
for prepend-constructed lists. The function \( \text{foldl} \) is equivalent to \( \text{foldr} \) only under certain restricted conditions (see below). If we require a catamorphism that processes lists from the left, then there is such a catamorphism for postpend-constructed lists, which we will call \( \text{foldL} \). Since our primary concern is initial algebras and their associated catamorphisms, we will refer to \( \text{foldr} \) as \( \text{foldL} \), by analogy with the naming convention \( \text{foldN} / \text{foldN}' \) (\( \text{foldn} / \text{foldn}' \)) for numbers. Our interest in \( \text{foldl} \) is only expository.

First, we present the definitions for \( \text{foldl} \) and \( \text{foldr} \) (\( \text{foldL} / \text{foldL}' \)). Then, we show that \( \text{foldL} \) is a catamorphism, and finally, we present \( \text{foldL} \), which is the general catamorphism for postpend-constructed lists.

The function \( \text{foldl} \) is defined as

\[
\text{foldl} \equiv \{ I_v ; f \} : \mathbb{L} \to \mathbb{S}; [ ] \mapsto v; (a \cdot l) \mapsto \text{foldl} \{ f(v,a) ; f \} (l),
\]

where \( f : \mathbb{S} \times \mathbb{A} \to \mathbb{S} \) is a binary function, and \( v \in \mathbb{S} \) is a constant. The function \( \text{foldr} \) is defined as

\[
\text{foldr} \equiv \{ I_v ; f \} : [ ] \mapsto v; (a \cdot l) \mapsto f(a; \text{foldr} \{ I_v ; f \} (l)),
\]

where \( f : \mathbb{A} \times \mathbb{S} \to \mathbb{S} \) is a binary function, and \( v \in \mathbb{S} \) is a constant. If \( f \) is restricted to be an associative function, i.e., \( \mathbb{S} = \mathbb{A} \), so \( f : \mathbb{A} \times \mathbb{A} \to \mathbb{A} \), and \( f(a,f(b,c)) = f(f(a,b),c) \), then \( \text{foldl} \equiv \text{foldr} \) (see [7]).

Proposition. If \( f : \mathbb{A} \times \mathbb{S} \to \mathbb{S} \) is an arbitrary function, then \( \text{foldL}' \) (i.e., \( \text{foldr}[I_v,f] \)) is a catamorphism, indicated by the commutative diagram

\[
\begin{array}{ccc}
1 + \mathbb{A} \times \mathbb{L} & \xrightarrow{[\text{empty}, \text{cons}]} & \mathbb{L} \\
\downarrow{1_1 + 1 \times \text{foldL}'[I_v,f]} & & \downarrow{\text{foldL}'[I_v,f]} \\
1 + \mathbb{A} \times \mathbb{S} & \xrightarrow{[I_v,f]} & \mathbb{S}
\end{array}
\]

Proof (by induction). The proof follows the same pattern as the proof that \( \text{foldN} / \text{foldN}' \) is a catamorphism: i.e., we first show that \( \text{foldL}' \) exists (Diagram 13 commutes), and then show uniqueness.

Existence (commutativity):

Base—(1, *):

\[
\begin{align*}
\land : \text{foldL}'[I_v,f] \circ [\text{empty}, \text{cons}](1, *) &= \text{foldL}'[I_v,f]([]) \\
&= v \\
\land : [I_v,f] \circ (1_1 + 1 \times \text{foldL}'[I_v,f])(1, *) &= [I_v,f](1, *) \\
&= v
\end{align*}
\]
Assumption—\((2, (a, l))\):

\[
foldL'[l_v, f] \circ [\text{empty}, \text{cons}](2, (a, l)) = [l_v, f] \circ (1_A \times foldL'[l_v, f])(2, (a, l)) \tag{Diagram 13}
\]

\[
foldL'[l_v, f](a \cdot l) = f(a, foldL'[l_v, f](l)) \tag{L/RHS reductions}
\]

Induction step—\((2, (a', (a \cdot l)))\): abbreviated to \((-\))

\[
\bowtie: foldL'[l_v, f] \circ [\text{empty}, \text{cons}](a \cdot l) = [l_v, f](a \cdot (a \cdot l)) \tag{def. of [empty, cons]}
\]

\[
= f(a', foldL'[l_v, f](a - l)) \tag{definition of foldL'}
\]

\[
= f(a', f(a, foldL'[l_v, f](l))) \tag{inductive assumption}
\]

\[
\bowtie: [l_v, f] \circ (1_A \times foldL'[l_v, f]) (a \cdot l) = [l_v, f](2, (a', foldL'[l_v, f](a \cdot l))) \tag{definition of 1 + \ldots}
\]

\[
= [l_v, f](2, (a', f(a, foldL'[l_v, f](l)))) \tag{inductive assumption}
\]

\[
= f(a', f(a, foldL'[l_v, f](l))) \tag{definition of [l_v, f]}
\]

Uniqueness: Suppose we have an \(F\)-algebra homomorphism \(g : L \to S\), indicated by the commutative diagram

\[
\begin{array}{ccc}
1 + A \times L & \xrightarrow{[\text{empty, cons}]} & L \\
\downarrow{1_A \times g} & & \downarrow{g} \\
1 + A \times S & \xrightarrow{[l_v, f]} & S
\end{array}
\]  

Base—\([\cdot] \) : \( \bowtie: g \circ [\text{empty}, \text{cons}](1, *) = g([\cdot]) \). \( \bowtie: [l_v, f] \circ (1_A \times g)(1, *) = [l_v, f](1, *) = v \). Since \( g \) is an \( F\)-algebra homomorphism, \( g([\cdot]) = v \).

Assumption—\( l \) : \( g(l) = foldL'[l_v, f](l) \).

Induction step—\((a \cdot l)\):

\[
g(a \cdot l) = f(a, g(l)) \tag{g is an \( F\)-algebra homomorphism, Diagram 14}
\]

\[
= f(a, foldL'[l_v, f](l)) \tag{by the inductive assumption}
\]

\[
= foldL'[l_v, f](a \cdot l) \tag{definition of foldL'}
\]
Therefore, \( g = foldL' [l_0, f] \).

For functor \( F_A : \text{CPO} \to \text{CPO}, S \mapsto 1 + S \times A, f \mapsto 1 + f \times 1_A \), the \( F \)-algebra homomorphism \( foldL \) is defined as \( foldL[l_0, f] : [] \mapsto v, (l \cdot a) \mapsto f(foldL[l_0, f](l), a) \), where \( f : S \times A \to S \) is a binary function, and \( v \in S \) is a constant.

**Proposition.** If \( f : S \times A \to S \) is an arbitrary function, then \( foldL[l_0, f] \) is a catamorphism, indicated by the commutative diagram

\[
\begin{array}{ccc}
1 + L \times A & \xrightarrow{[\text{empty}, \text{snoc}]} & L \\
1 + foldL[l_0, f] \times 1_A & \downarrow & \downarrow \text{foldL[l_0, f]} \\
1 + S \times A & \xrightarrow{[l_0, f]} & S
\end{array}
\] (15)

**Proof.** Reverse the roles of the objects/morphisms associated with the products in the above proof. □

**Examples**

The following are concrete examples of \( F \)-algebras and associated catamorphisms. For the first and second examples, the parameter \( A \) to the functor \( F_A \) is the set of natural numbers \( \mathbb{N} \), so the example deals with lists of natural numbers. For the third example, the parameter \( A \) is an arbitrary set. For the first example, we also include the postpend version, which extends analogously to the second and third examples. Note that \( \mathbb{N} \) is also a CPO, with 0 as the bottom element, \( \infty \) as the LUB and \( \leq \) for numbers as the partial (total) order.

For the first example, we specialize object \( A \) in Diagram 13 to \( \mathbb{N} \) to work with lists of natural numbers. If needed, we could also work with the set \( \mathbb{N} \) as a model of the natural numbers (see Diagram 12), in which case the following examples can be straightforwardly translated. For expository purposes, \( \mathbb{N} \) is more succinct. The functor \( F_\mathbb{N} = K_1 + K_\mathbb{N} \times I : \text{CPO} \to \text{CPO}; S \mapsto 1 + \mathbb{N} \times S, f \mapsto 1 + 1_\mathbb{N} \times f \).

An initial algebra in the category \( \text{Alg}(F_\mathbb{N}) \) is the pair \((L, [\text{empty}, \text{cons}])\), where \( L \) is the set of lists of numbers, and \( \text{cons} \) is the corresponding function for constructing such lists. The pair \((\mathbb{N}, [l_0, (+)])\), where \( l_0 : * \mapsto 0 \) and \( (+) : (x, y) \mapsto x + y \) is also an \( F \)-algebra in this category. The two algebras are related by...
the catamorphism $foldL'[l_0, (+)] : L \to \mathbb{N}$, and catamorphism is indicated in commutative diagram

$$
\begin{array}{c}
1 + \mathbb{N} \times L & \xrightarrow{[\text{empty}, \text{cons}]} & L \\
\downarrow_{1 + 1_\mathbb{N} \times foldL'[l_0, (+)]} & & \downarrow_{foldL'[l_0, (+)]} \\
1 + \mathbb{N} \times \mathbb{N} & \xrightarrow{[l_0, (+)]} & \mathbb{N}
\end{array}
$$

(16)

This catamorphism computes the sum of a list of numbers: 1, 2, 3, and 4, which is constructed as: $\text{cons}(4, \text{cons}(3, \text{cons}(2, \text{cons}(1, \text{empty})))) = [4, 3, 2, 1]$. From the definition of $foldL'$ we have, for list instance, $foldL'[l_0, (+)][[[4, 3, 2, 1]]] = 4 + (3 + (2 + (1 + (0)))) = 10$.

Prepend-constructed lists of natural numbers are based on the functor $F_\mathbb{N} : \text{CPO} \to \text{CPO}; S \mapsto 1 + S \times \mathbb{N}, f \mapsto 1 + f \times 1_\mathbb{N}$. An initial algebra in the category based on this functor is the pair $(L, [\text{empty}, \text{snoc}])$. The catamorphism for summing such lists is $foldL[l_0, (+)] : L \to \mathbb{N}$, indicated in commutative diagram

$$
\begin{array}{c}
1 + L \times \mathbb{N} & \xrightarrow{[\text{empty}, \text{snoc}]} & L \\
\downarrow_{1 + foldL[l_0, (+)] \times 1_\mathbb{N}} & & \downarrow_{foldL[l_0, (+)]} \\
1 + \mathbb{N} \times \mathbb{N} & \xrightarrow{[l_0, (+)]} & \mathbb{N}
\end{array}
$$

(17)

The list: 1, 2, 3, and 4 is constructed as: $\text{snoc}(\text{snoc}(\text{snoc}(\text{snoc}(\text{empty}(), 1), 2), 3), 4) = [1, 2, 3, 4]$. The sum of this list is computed as: $foldL[l_0, (+)][[1, 2, 3, 4]] = (((((0) + 1) + 2) + 3) + 4) = 10$.

For the second example, another $F$-algebra in the category $\text{Alg}(F_\mathbb{N})$ is the pair $(\mathbb{N}, [l_0, (\times)])$, where $l_1 : * \mapsto 1$ and $(\times) : (x, y) \mapsto x \times y$, and catamorphism is indicated in commutative diagram

$$
\begin{array}{c}
1 + \mathbb{N} \times L & \xrightarrow{[\text{empty}, \text{cons}]} & L \\
\downarrow_{1 + 1_\mathbb{N} \times foldL'[l_1, (\times)]} & & \downarrow_{foldL'[l_1, (\times)]} \\
1 + \mathbb{N} \times \mathbb{N} & \xrightarrow{[l_1, (\times)]} & \mathbb{N}
\end{array}
$$

(18)

This catamorphism computes the product of a list of numbers. For instance, $foldL'[l_1, (\times)][[4, 3, 2, 1]] = 4 \times (3 \times (2 \times (1 \times (1)))) = 24$.

For the third example, an $F$-algebra for an arbitrary set $A$ is the pair $[l_0, (\text{succ} \circ p_2)]$, where $\text{succ} : n \mapsto n + 1$ increments a natural number, and $p_2 : A \times \mathbb{N} \to \mathbb{N}$ is the projection onto the second factor. A
Categorial compositionality III

The catamorphism for computing list length is indicated in commutative diagram

\[
1 + A \times L \xrightarrow{\text{[empty, cons]}} L \\
\downarrow_{1 + A \times \text{foldL'}[b_0, (\text{succ} \circ p_2)]} \quad \downarrow_{\text{foldL'}[b_0, (\text{succ} \circ p_2)]} \\
1 + A \times N \xrightarrow{\text{[b_0, (\text{succ} \circ p_2)]}} N
\]

(19)

For instance, \(\text{foldL'}[b_0, (\text{succ} \circ p_2)]([\text{cat}, \text{dog}, \text{mouse}]) = \text{succ}(\text{succ}(0)) = 3\).

Some other examples of catamorphisms for the same category of \(F\)-algebras on functor \(F_A\) include:

- **smallest** The smallest number of a list: \(\text{foldL}[I_{\infty}, \text{min}]\), where \(\text{min}: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}; (x, y) \mapsto x \text{ if } x \leq y \text{ else } y\), and \(\text{min}(x, \infty) = x\) for every \(x\).

- **largest** The largest number of a list: \(\text{foldL}[I_{-\infty}, \text{max}]\), where \(\text{max}: (x, y) \mapsto x \text{ if } x \geq y \text{ else } y\), and \(\text{max}(x, -\infty) = x\) for every \(x\). Here, we assume the set of integers \(\mathbb{Z}\). In the case of \(\mathbb{N}\), \(-\infty\) can be replaced with 0.

- **any** Determining whether any proposition is true: \(\text{any}: ps \mapsto \text{foldL}[I_{\text{False}}, \lor](ps)\), where \(\lor\) is logical disjunction, and \(ps\) is a list of propositions.

- **every** Determining that all propositions are true: \(\text{every}: ps \mapsto \text{foldL}[I_{\text{True}}, \land](ps)\), where \(\land\) is logical conjunction.

- **reverse** Reversing a list of items: \(\text{reverse}: xs \mapsto \text{foldL}[I_{[\ ]}, \text{rev}](xs)\), where \(\text{rev}: (x, xs) \mapsto xs \oplus [x]\), and \(\oplus\) appends (concatenates) two lists.

- **concatenate** Concatenating lists of items into a single list: \(\text{concat}: [xs] \mapsto \text{foldL}[I_{[\ ]}, \oplus]([xs])\), where \([xs]\) is a list of lists.

- **filter** Filtering a list to retain only those items that satisfy some criterion: \(\text{filter} f: xs \mapsto \text{foldL}'[[]](xs)\), where \(f: A \to \text{Bool}; a \mapsto \text{True if } f(a), a \mapsto \text{False if } \neg f(a)\) is the criterion, and \(f': (x, xs) \mapsto x \oplus xs\) if \(f(x)\), \((x, xs) \mapsto xs\) if \(\neg f(x)\).

For further examples, see [7].
Trees

For computation with regard to trees, we start with the polynomial functor $F_A = K_A + I \times I : \text{CPO} \to \text{CPO}$; $S \mapsto A + S \times S$, $f \mapsto 1_A + f \times f$. The $F$-algebras for the category $\text{Alg}(F_A)$ are the pairs $(S, [f, g])$, where $S$ is a set, $[f, g] : A + S \times S \to S$, $f : A \to S$ is a unary function, and $g : S \times S \to S$ is a binary function. An initial algebra in this category is $(T, [\text{leaf}, \text{branch}])$, where $T$ is the set of trees of type $A$, $[\text{leaf}, \text{branch}] : A + T \times T \to T$, $\text{leaf} : A \to T; a \mapsto (a)$ returns a tree consisting of a single leaf $a \in A$, and $\text{branch} : T \times T \to T; (l, r) \mapsto (l, r)$ returns a tree consisting of a left branch $l$ and a right branch $r$, where $l, r \in T$. That is, initial algebra $(T, [\text{leaf}, \text{branch}])$ is an $F$-algebra $(S, [f, g])$, where $S$ is the set of all trees $T$, $f$ is the unary function $\text{leaf}$, and $g$ is the binary function $\text{branch}$. The tuple notation for trees is recursive, e.g., $((a_1), (a_2), (a_3))$ is a tree whose left branch is the leaf $a_1$, and whose right branch is the (sub)tree whose left branch is the leaf $a_2$ and right branch is the leaf $a_3$, depicted as

$$
\begin{array}{c}
a_1 \\
\downarrow \\
a_2 \\
\downarrow \\
a_3
\end{array}
$$

A catamorphism from initial algebra $(T, [\text{leaf}, \text{branch}])$ to an arbitrary $F$-algebra $(S, [f, g])$ in $\text{Alg}(F_A)$ is the recursive function $\text{fold}T$ (i.e., fold for trees), defined as follows. The (higher-order) function $\text{fold}T$ takes a unary function $f : A \to S$ and a binary function $g : S \times S \to S$ and returns the function $\text{fold}T[f, g] : T \to S$, where $\text{fold}T[f, g] : \langle a \rangle \mapsto f(a), \langle l, r \rangle \mapsto g(\text{fold}T[f, g](l), \text{fold}T[f, g](r))$, as indicated in commutative diagram

$$
\begin{array}{c}
A + T \times T \xrightarrow{1_A + (\text{fold}T[f, g] \times (\text{fold}T[f, g]))} T \\
\downarrow \\
A + S \times S \xrightarrow{[f, g]} S
\end{array}
$$

**Proposition.** If $f : A \to S$ and $g : S \times S \to S$ are arbitrary functions, $T$ and $S$ are sets, $\text{fold}T[f, g] : T \to S$ is a catamorphism.

**Proof.** First, we proved existence (commutativity), then uniqueness (by induction).
Existence (commutativity):

Leaf—(1, a):

\( \circlearrowleft: \text{foldT}[f, g] \circ [\text{leaf}, \text{branch}](1, a) = \text{foldT}[f, g]((a)) \)  
  (definition of [\text{leaf}, \text{branch}])

\( = f(a) \)  
  (definition of \text{foldT}[f, g])

\( \circlearrowleft: [f, g] \circ (1_A + (\text{foldT}[f, g]) \times (\text{foldT}[f, g]))(1, a) = [f, g](1, a) \)  
  (definition of 1_A + …)

\( = f(a) \)  
  (definition of [f, g])

Branch—(2, (l, r)): abbreviated to (−)

\( \circlearrowleft: \text{foldT}[f, g] \circ [\text{leaf}, \text{branch}](−) = \text{foldT}[f, g]((l, r)) \)  
  (def. of [\text{leaf}, \text{branch}])

\( = g(\text{foldT}[f, g](l), \text{foldT}[f, g](r)) \)  
  (def. of \text{foldT}[f, g])

\( \circlearrowleft: [f, g] \circ (\cdots \times \text{foldT}[f, g])(−) = [f, g](2, (\text{foldT}[f, g](l), \text{foldT}[f, g](r))) \)  
  (def. of 1_A + …)

\( = g(\text{foldT}[f, g](l), \text{foldT}[f, g](r)) \)  
  (def. of [f, g])

Uniqueness: Suppose we have an \( F \)-algebra homomorphism \( h : T \to S \), indicated by the commutative diagram

\[
\begin{array}{ccc}
A + T \times T & \xrightarrow{[\text{leaf}, \text{branch}]} & T \\
\downarrow 1_A + h \times h & & \downarrow h \\
A + S \times S & \xrightarrow{[f, g]} & S
\end{array}
\]  
  (22)

Base—(1, a): \( \circlearrowleft: h \circ [\text{leaf}, \text{branch}](1, a) = h((a)). \)  
  (definition of [\text{leaf}, \text{branch}])

\( \circlearrowleft: [f, g] \circ (1_A + h \times h)(1, a) = [f, g](1, a) = f(a) \). Since \( h \) is an \( F \)-algebra homomorphism, \( h((a)) = f(a) = \text{foldT}[f, g]((a)). \)

Inductive assumption—(2, t): \( h(t) = \text{foldT}[f, g](t). \)

Induction step, left—(2, ((a), t)):
The commutativity of Diagram 22 means that \( h \circ [\text{leaf}, \text{branch}](2, ((a), t)) = g(h((a)), h(t)) \). So,

\[
\begin{align*}
  h((a), t) &= g(h((a)), h(t)) & \text{(reduction of both sides)} \\
  &= g(foldT[f, g]((a)), foldT[f, g](t)) & \text{(base and inductive assumption)} \\
  &= foldT[f, g]((a), t) & \text{(definition of foldT)}
\end{align*}
\]

Induction step, right—\((2, ((t), (a)))\): Swap argument roles.

Therefore, \( h = foldT[f, g] \).

**Examples**

Computing the sum of the numbers in the leaves of a tree of numbers is the catamorphism given in commutative diagram

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
N + T \times T \\
1_A + foldT[1_N, (+) \times foldT[1_N, (+)]]
\end{array}
\end{array}
\end{array} \xrightarrow{[\text{leaf}, \text{branch}]} \begin{array}{c}
\begin{array}{c}
T \\
foldT[1_N, (+)]
\end{array}
\end{array}
\end{array}
\]

(23)

For instance, \( foldT[1_N, (+)](\langle(1), (2, 3)\rangle) = +((1), +(2, 3)) = 6 \).

Computing tree depth (i.e., the number of branches from the root of the tree to the “deepest” leaf) is the catamorphism given in commutative diagram

\[
\begin{array}{c}
\begin{array}{c}
\begin{array}{c}
A + T \times T \\
1_A + foldT[1_A, (\text{succ} \circ \text{max}) \times foldT[1_A, (\text{succ} \circ \text{max})]]
\end{array}
\end{array}
\end{array} \xrightarrow{[\text{leaf}, \text{branch}]} \begin{array}{c}
\begin{array}{c}
T \\
foldT[1_A, (\text{succ} \circ \text{max})]
\end{array}
\end{array}
\end{array}
\]

(24)

For instance, \( foldT[1_A, (\text{succ} \circ \text{max})](\langle(1), (2, 3)\rangle) = \text{succ} \circ \text{max}(1, 1) = 2 \).
Counting the number of leaves in a tree is the catamorphism given in commutative diagram

\[
\begin{array}{ccc}
A + T \times T & \xrightarrow{[\text{leaf}, \text{branch}]} & T \\
\downarrow 1_A + (\text{fold} T[1, (+)] \times \text{fold} T[1, (+)]) & & \downarrow \text{fold} T[1, (+)] \\
A + N \times N & \xrightarrow{[1, (+)]} & N
\end{array}
\]

For instance, \(\text{fold} T[1, (+)]((0, (1, 2, 3))) = +(1, +(1, 1)) = 3\).

**Relation between numbers, lists, and trees**

Under certain circumstances, operations pertaining to numbers are special cases of operations pertaining to lists, and list operations are special cases of operations pertaining to trees. The precise relationship between these domains is revealed by a category theory treatment of recursion.

**Numbers from lists**

For the category \(\text{Alg}(F)\) on the functor \(F : \text{CPO} \to \text{CPO}; S \mapsto 1 + S, f \mapsto 1 + f\), and an arbitrary element \(a \in A\), where \(A\) is a non-empty set in \(\text{CPO}\), there is an initial algebra \((L, [\text{empty}, \text{cons}_a])\), where \(L\) is the set of lists constructed solely of the element \(a\), \(\text{empty} : 1 \to L\) is a function that returns the empty list (in effect, it is a constant), and \(\text{cons}_a : L \to L; l \mapsto a \cdot l\) is a unary function returning the list with constant element \(a\) prepended to \(l\). Hence, \(L\) is a set of lists where each list consists of zero or more repetitions of the element \(a\). The catamorphism associated with this initial algebra is the function \(\text{fold}_a[I_v, f] : L \mapsto S; [] \mapsto v, (a \cdot l) \mapsto f(\text{fold}_a[I_v, f][l])\), as indicated in commutative diagram

\[
\begin{array}{ccc}
1 + L & \xrightarrow{[\text{empty}, \text{cons}_a]} & L \\
\downarrow 1_1 + \text{fold}_a[I_v, f] & & \downarrow \text{fold}_a[I_v, f] \\
1 + S & \xrightarrow{[1, f]} & S
\end{array}
\]

The proof that \(\text{fold}_a[I_v, f]\) is a catamorphism follows the same pattern for \(\text{fold}_N, \text{fold}_L\), etc. Since \((L, [\text{empty}, \text{cons}_a])\) is an initial algebra, we have the following corollary.

**Corollary.** \((L, [\text{empty}, \text{cons}_a]) \cong (N, [\text{zero}, \text{succ}])\).

**Proof.** Initial objects are isomorphic.
The isomorphic relationship between \( (L, [\text{empty}, \text{cons}_a]) \) and \( (N, [\text{empty}, \text{cons}]) \), indicated in the following commutative diagram:

\[
\begin{align*}
1 + L & \xrightarrow{[\text{empty}, \text{cons}_a]} L \\
1 + \text{fold}_n & \xrightarrow{1 + \text{fold}(\text{zero}, \text{succ})} \text{fold}(\text{zero}, \text{succ}) \\
1 + N & \xrightarrow{[\text{zero}, \text{succ}]} N
\end{align*}
\]  

(27)

yields examples of primitive forms of counting and repetition.

An example of the close relationship between number and repetition follows. A list of three repeated items can be counted using \( \text{fold}_a \):

\[
\text{fold}_a([\text{zero}, \text{succ}]) ([a; a; a]) = \text{succ}(\text{succ}(\text{succ}(\text{Zero}))),
\]

which corresponds to 3. Three repetitions can be generated using \( \text{fold}_n \):

\[
\text{fold}_n([\text{empty}, \text{cons}_a])(\text{succ}(\text{succ}(\text{succ}(\text{Zero})))) = \text{cons}_a(\text{cons}_a(\text{cons}_a([]))) = [a; a; a].
\]

\( F \)-algebras are constructed from functors, and functors are compared by natural transformations. Thus, to see the relationship between various \( F \)-algebras, we first view them as natural transformations. In general, for \( F \)-algebras on an endofunctor \( F : C \to C \), if we have a family \( \{\alpha_A : F(A) \to A\} \) of \( C \) - morphisms, then for any \( F \)-algebra homomorphism \( f : A \to B \), the definition of \( F \)-algebra homomorphism implies that Diagram 28 commutes. That is, \( \alpha_A \) is a natural transformation, as indicated by commutative diagram

\[
\begin{array}{ccc}
A & \xrightarrow{F(A)} & A \\
\downarrow f & & \downarrow f \\
B & \xrightarrow{F(B)} & B
\end{array}
\]

(28)

To compare initial algebras associated with number and with list we must use a common framework. For number, this means using an initial algebra built from the same object \( L \) used for lists, i.e. \( (L, [\text{empty}, \text{cons}_a]) \cong (N, [\text{zero}, \text{succ}]) \). From this common framework, we see that the relationship
between initial algebras and catamorphisms is indicated by commutative diagram 29

\[
1 + A \times L \xrightarrow{1 + p_2} 1 + L \xrightarrow{\text{[empty, cons op}_p_2]} L
\]

\[
1 + A \times S \xrightarrow{1 + p_2} 1 + S \xrightarrow{\text{[I}_v; f]} S
\]

Recall that \( p_2 : A \times L \to L \) is the projection onto the second factor of the product. Then, from the uniqueness property of catamorphisms, we have \( \text{fold}_a[I_v; f] = \text{fold}_L[I_v; f \circ p_2] \). Hence, every catamorphism in number is a catamorphism in list.

**Lists from trees**

A list can also be represented as a special kind of tree: i.e., a tree with either left-only, or right-only branches. To represent lists as trees, the formulation for a tree must be modified to yield the equivalent of lists as a special case: essentially, we use a tree where the values are embedded at the roots, instead of the leaves. This modification is straightforward, and achieved using the polynomial functor \( F_A = K_1 + K_A \times I \times I : \text{CPO} \to \text{CPO} ; S \mapsto 1 + A \times S \times S, f \mapsto 1 + 1_A \times f \times f \). (Compare the earlier version of trees, which used a different polynomial functor, also called \( F_A = K_A + I \times I \). Note the inclusion of a terminal object to model empty lists.) The associated initial algebra for the category of \( F \)-algebras on this functor, and catamorphism are given in commutative diagram 30

\[
1 + A \times T \times T \xrightarrow{\text{[empty, branch]}} T
\]

\[
1 + A \times S \times S \xrightarrow{\text{[I}_v; f]} S
\]

where:

- **empty** : \( 1 \to T \) returns an empty tree \( \{\} \);

- **branch** : \( A \times T \times T \to T ; (a, l, r) \mapsto (a, l, r) \) returns a tree with root value \( a \in A \), left branch \( l \in T \) and right branch \( r \in T \);
• \( \text{fold}_T[v,f] : T \to S; (\cdot) \mapsto v, (a,l,r) \mapsto f(a,l,r); \)

• \( I_v : 1 \to S \) returns constant \( v \in S; \) and

• \( f : A \times S \times S \to S; (a,s_l,s_r) \mapsto s \) is a ternary function.

Such trees contain values at their branches, depicted as follows:

\[
\begin{array}{c}
\text{a}_1 \\
\downarrow \\
\text{a}_2 \\
\downarrow \\
\text{a}_3 \\
\end{array}
\]

The corresponding catamorphism for counting leaves, which includes counting left/right-only list-like trees is \( \text{fold}_T[l_0,f] \), where \( f : A \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}; (a,n_l,n_r) \mapsto 1 + n_l + n_r. \) For instance, the list \([\text{John}, \text{Mary}, \text{Sue}]\) is represented by the left-only branching tree \( \langle \text{John}, \langle \text{Mary}, \langle \cdot, \cdot, \cdot \rangle, \cdot \rangle, \cdot \rangle \rangle. \) The number of items in this tree (list) is counted as \( \text{fold}_T[l_0,f](\langle \text{John}, \langle \text{Mary}, \langle \cdot, \cdot, \cdot \rangle, \cdot \rangle, \cdot \rangle \rangle) = (1 + (1 + (1 + 0 + 0) + 0) + 0) = 3. \)

The right branch of a left-only branching tree is always empty, so the morphism (ternary function) \( \text{branch} \) simplifies to the morphism (binary function) \( \text{branch}_l : A \times T \to T; (a,l) \mapsto (a,l,\cdot) \). (An analogous simplification also applies to right-only branching trees, i.e., \( \text{branch}_r : (a,r) \mapsto (a,\cdot,r) \).) This morphism is the significant constituent of an initial algebra \( (T, [\text{empty}, \text{branch}_l]) \), indicated in commutative diagram

\[
\begin{array}{c}
1 + A \times T \\
\downarrow_{1+1_A \times \text{fold}_T[l_v,f]} \\
1 + A \times S \\
\downarrow_{[v,f]} \\
S \\
\end{array}
\]

where \( \text{fold}_T[l_v,f] : T \to S; (\cdot) \mapsto v, (a,l,\cdot) \mapsto f(a,l), \) where \( f : A \times S \to S \) is a function. Essentially, the right branch of every tree is ignored. Since \( (T, [\text{empty}, \text{branch}_l]) \) is also an initial algebra, we have \( (T, [\text{empty}, \text{branch}_l]) \cong (L, [\text{empty}, \text{cons}]). \)

Just like the comparison of numbers and lists, to compare initial algebras for lists and trees, we require \( (T, [\text{empty}, \text{branch}_l]) \) as a common framework. The relationship between tree and list algebras is given by
Hence, every catamorphism in list is a catamorphism in tree.

**Numbers from trees**

Having shown that the initial algebras for numbers and lists, and lists and trees are related by natural transformations, the initial algebras for numbers and trees must also be related by natural transformations, since natural transformations are morphisms in a functor category, and so natural transformations compose as morphisms compose. The initial algebra for number from trees is \( (T, [\text{empty}, \text{branch}_a]) \), where \( \text{branch}_a : T \to T; t \mapsto (a, t, \{\}) \). In this case, items are added (prepended) to the root of the tree.

**No natural equivalences**

Recall that two functors are related by a natural equivalence whenever every member of a family of morphisms that constitute a natural transformation is an isomorphism. Notice that there is also a natural transformation from the endofunctor for number, \( F : 1 + S, 1_1 + f \) to the endofunctor for list, \( F_A : 1 + A \times S, 1_1 + A \times f \), that is indicated in the following commutative diagram

\[
\begin{array}{ccc}
1 + L & \xrightarrow{1_1 + (1_1, 1_L)} & 1 + A \times L \\
\downarrow{1_1 + f} & & \downarrow{1_1 + A \times f} \\
1 + S & \xrightarrow{1_1 + (1_1, 1_S)} & 1 + A \times S
\end{array}
\]

However, in general, the composition of these arrows with the ones in the reverse direction is not an identity, e.g., \((1_1 + p_2) \circ (1_1 + (1_1, 1_S)) \not= 1_1 + A \times S\): essentially, the information regarding object \( A \) is lost. The same situation applies to the relationships between the other pairs of endofunctors. Therefore, these
endofunctors are not related by natural equivalences.

**F-coalgebras, final coalgebras, and anamorphisms**

F-coalgebra, final coalgebra, and anamorphism are (respectively) dual to F-algebra, initial algebra, and catamorphism. That is, their definitions are obtained by reversing the direction of the arrows in the dual construct. Here, we list the definitions.

**Definition (F-coalgebra).** For an endofunctor $F : C \to C$, an F-coalgebra is a pair $(A, \alpha)$, where $A$ is an object and $\alpha : A \to F(A)$ is a morphism in $C$.

**Example (F-coalgebra).** Let $C$ be the category $\text{Set}$, $F : \text{Set} \to \text{Set}$ be the functor $F : S \mapsto S \times S$, $f \mapsto f \times f$ (where $f : S \to T$ is a function), $A$ be the set of natural numbers $\mathbb{N}$ (so, $\mathbb{N} \in \text{Set}$), and $\alpha : \mathbb{N} \to \mathbb{N}$ be the morphism $\alpha : n \mapsto (p, q)$, where $p$ is the largest prime factor of $n$, and $n = pq$. Then, $(\mathbb{N}, \alpha)$ is an $F$-coalgebra.

**Definition (F-coalgebra homomorphism).** An F-coalgebra homomorphism $h : (B, \beta) \to (A, \alpha)$ is a morphism $h : B \to A$ (in $C$) such that the following diagram commutes:

$$
\begin{array}{c}
B & \xrightarrow[\beta]{F(B)} \\
\downarrow{h} & \\
A & \xrightarrow[\alpha]{F(A)}
\end{array}
$$

(35)

**Definition (Category of F-coalgebras).** For an endofunctor $F : C \to C$, the category of F-coalgebras $\text{CoAlg}(F)$ has F-coalgebras $(A, \alpha)$ for objects, and F-coalgebra homomorphisms $h : (B, \beta) \to (A, \alpha)$ for morphisms.

**Definition (Final coalgebra).** A final F-coalgebra $(A, \text{fin})$, hereafter also simply called a final algebra, is a final (i.e., terminal) object in the category of F-coalgebras $\text{CoAlg}(F)$. That is, there exists a unique F-coalgebra homomorphism from every F-coalgebra in $\text{CoAlg}(F)$ to $(A, \text{fin})$.

Like initial algebras, the presence of a final coalgebra in a category $\text{CoAlg}(F)$ depends on the particular functor $F$, and is not guaranteed to exist in every case.

**Proposition.** If $F : C \to C$ is a polynomial functor, then the category of F-coalgebras $\text{CoAlg}(F)$ has a final coalgebra.

**Proof.** See [3], chapters 10 and 11.
**Definition (Anamorphism).** An anamorphism $h : (B, \beta) \to (A, \text{fin})$ is the unique $F$-coalgebra homomorphism from $F$-algebra $(B, \beta)$ to final $F$-coalgebra $(A, \text{fin})$. That is, $F(h) \circ \beta = \text{fin} \circ h$ and the uniquely specified $h$ for each such $\beta$ is denoted $\text{ana} \beta$ (i.e., $h = \text{ana} \beta$), as indicated in the following diagram:

\[
\begin{align*}
B \xrightarrow{\beta} F(B) \\
\downarrow \text{ana} \beta \quad \quad \quad \downarrow F(\text{ana} \beta) \\
A \xrightarrow{\text{fin}} F(A)
\end{align*}
\]

where $\text{ana} \beta$ is also denoted $[\beta]$ using lens brackets (see [4]).

A final coalgebra $(A, \text{fin})$ in $\text{CoAlg}(F)$ on a functor $F : C \to C$ is an example of a universal construction. Specifically, it is an instance of a universal morphism (see Diagram 42).

**Distributive categories and conditional functions**

Before presenting specific coalgebras that we use in our explanation of systematicity, we need the definitions of distributive categories and conditional functions (see [5], p66–69, for further details).

**Definition (Distributive categories).** A distributive category $C$ is a category with all products and coproducts such that we have natural isomorphisms

\[
A \times (B + C) \cong (A \times B) + (A \times C)
\]

\[
A \times 0 \cong 0
\]

The associated morphisms are called

\[
distr : A \times (B + C) \to (A \times B) + (A \times C)
\]

\[
undistr : (A \times B) + (A \times C) \to A \times (B + C)
\]

\[
null : A \times 0 \to 0
\]

\[
unnul : 0 \to A \times 0
\]

Distributive categories may be thought of an analogous to the distributive law in arithmetic (and other mathematical structures such as rings and vector spaces).
**Set** is a distributive category, with, for example, \( \text{distr} : (a, (1, b)) \mapsto (1, (a, b)) \) and \( (a, (2, c)) \mapsto (2, (a, c)) \) for \( a \in A, b \in B, c \in C \). CPO is also a distributive category.

**Definition (Condition function).** In a category \( C \) whose objects are sets (possibly with structure) with a boolean object \( \text{Bool} = \{ \text{True}, \text{False} \} \), a condition function \( p : A \to \text{Bool} \) tests whether \( a \in A \) satisfies condition (predicate) \( p \), i.e. whether \( p(a) = \text{True} \).

**Definition (Conditional function).** In a distributive category \( C \) whose objects are sets (possibly with structure), a conditional function \( (p \to f; g) : A \to B + C \) is a function \( p^? : A \to A + A \), i.e., \( p^?(a) = (1, a) \) if \( p(a) = \text{True} \) and \( p^?(a) = (2, a) \) if \( p(a) = \text{False} \), where \( p \) is the underlying condition function, together with two functions \( f, g : A \to B \) associated with condition \( p : A \to \text{Bool} \) so that \( (p \to f, g) : a \mapsto f(a) \) if \( p(a) \) else \( g(a) \). By definition, \( (p \to f, g) = [f, g] \circ p^? \), whose construction is indicated in commutative diagram

\[
\begin{array}{ccc}
A & \xrightarrow{(1_a, p)} & A \times \text{Bool} \\
p^? \downarrow & & \downarrow \text{distr} \\
A + A & \xrightarrow{p_1 + p_1} & (A \times 1) + (A \times 1) \\
f + g \downarrow & & (f \times 1) + (g \times 1) \\
B + B & \xrightarrow{p_1 + p_1} & (B \times 1) + (B \times 1)
\end{array}
\]

where \( p_1 \) is the usual projection onto the first factor associated with a product.

**Lists**

We saw how lists are constructed (put together) by initial algebras. Final coalgebras are dual to initial algebras [3]; lists are destructed (pulled apart) by final coalgebras. The resulting data structure is sometimes called a colist (see, e.g., [9]). Hence, the category theory development of final coalgebras for lists derives from the dual definition of initial algebra in the category of \( F \)-algebras \( \text{Alg}(F_A) \) on the functor \( F_A : \text{CPO} \to \text{CPO}; S \mapsto 1 + A \times S, f \mapsto 1_1 + 1_A \times f \). That is, we start with the dual category \( \text{CoAlg}(F_A) \) whose objects are the \( F_A \)-coalgebras \( (S, \alpha : S \to 1 + A \times S) \). An arbitrary \( \alpha \) in this category has the form \( (p \to 1_1, (f, g)), \) where \( 1_1 : S \to 1 \) is a constant function and \( * \) is the unnamed element, \( f : S \to A, g : S \to S, \) and \( p : S \rightarrow \text{Bool} \), where \( p : s \mapsto \text{True} \), if \( p(s) \) else \( \text{False} \), so that \( (p \to 1_1, (f, g)) : s \mapsto *, \) if \( p(s) \) else \( \langle f(s), g(s) \rangle \). That is, if \( p(s) \) is true, then \( \alpha(s) = (1,*) \), else \( \alpha(s) = (2, (f(s), g(s))) \).
The morphisms in $\text{CoAlg}(F_A)$ are $F_A$-coalgebra homomorphisms. A final coalgebra in this category is $(L, p_e \to 1_*, \langle \text{hd}, \text{tl} \rangle)$, where $L$ is the set of lists over $A$, condition $p_e : L \to \text{Bool}$ tests whether $l \in L$ is empty (i.e., $l = []$), and $\text{hd} : L \to A; [\cdot] \mapsto *$ and $\text{tl} : L \to A; [\cdot] \mapsto [\cdot]$ return the head and tail of a list (respectively). Functions $\text{hd}$ and $\text{tl}$ are extended to empty lists to satisfy the general definition of a conditional function (see above) where the components are total functions. The unique $F_A$-coalgebra homomorphism associated with this final algebra is the anamorphism called $\text{unfold}_L$ (i.e., unfold for lists), defined as $\text{unfold}_L(p \to 1_*, (f, g)) : S \to L$

$$s \mapsto [] \quad \text{if } p(s)$$

$$s \mapsto f(s) \cdot \text{unfold}_L(p \to 1_*, (f, g))(g(s)) \quad \text{otherwise}$$

The final coalgebra and associated anamorphism are shown in commutative diagram

\[
\begin{array}{c}
S & \xrightarrow{p \to 1_*, (f, g)} & 1 + A \times S \\
\downarrow \text{unfold}_L(p \to 1_*, (f, g)) & & \downarrow 1 + 1_A \times \text{unfold}_L(p \to 1_*, (f, g)) \\
L & \xrightarrow{p_e \to 1_*, \langle \text{hd}, \text{tl} \rangle} & 1 + A \times L
\end{array}
\] (38)

To show that $\text{unfold}_L(p \to 1_*, (f, g))$ is indeed an anamorphism, we employ the approximation lemma [7], which is used to prove coalgebraic properties (see also [10]).

**Approximation lemma (Bird).** The approximation lemma is based on the function $\text{approx}$. For lists, this function is defined as $\text{approx} : \mathbb{N} \times L \to L$, where:

$$\text{approx}((n + 1), []) = []$$

$$\text{approx}((n + 1), (x \cdot xs)) = x \cdot \text{approx}(n, xs)$$

By case exhaustion, $\text{approx}(0, xs) = \bot$. That is, $\text{approx}$ takes on the value “undefined”. For two (finite, partial, or infinite) lists $xs$ and $ys$ the approximation lemma is given by the following equivalence:

$$xs = ys \iff \forall n. \text{approx}(n, xs) = \text{approx}(n, ys)$$

That is, the two lists $xs$ and $ys$ are the same list if and only if their applications to the function $\text{approx}$
agree for all natural numbers $n$.

The approximation lemma affords proofs of coalgebraic properties via induction over the natural numbers. The \textit{generic approximation lemma} \cite{11} does essentially the same thing for a general class of data structures, including lists and trees.

**Proposition.** If $(p \rightarrow I_\ast, (f, g)) : S \rightarrow 1 + A \times S$ is an arbitrary conditional function, then $\text{unfold}L'(p \rightarrow I_\ast, (f, g))$ is an anamorphism.

**Proof.** First we prove existence (commutativity of Diagram 38), then uniqueness (by case analysis and the approximation lemma). (See also \cite{10,12} for another general method of proving coalgebraic properties, called \textit{coinduction}, which is dual to induction).

Existence (commutativity):

$p(s)$:

\[
\begin{align*}
\triangledown & : (1 + 1_A \times \text{unfold}L'(p \rightarrow I_\ast, (f, g))) \circ (p \rightarrow I_\ast, (f, g))(s) \\
& = (1 + 1_A \times \text{unfold}L'(p \rightarrow I_\ast, (f, g)))(*) \\
& = (1, *) \quad \text{(definition of } p \rightarrow I_\ast, (f, g)\text{)} \\
\triangledown & : (p_\epsilon \rightarrow I_\ast, (\text{hd, tl})) \circ \text{unfold}L'(p \rightarrow I_\ast, (f, g))(s) \\
& = (p_\epsilon \rightarrow I_\ast, (\text{hd, tl}))(*) \quad \text{(definition of } \text{unfold}L'(p \rightarrow I_\ast, (f, g))\text{)} \\
& = (1, *) \quad \text{(definition of } p_\epsilon \rightarrow I_\ast, (\text{hd, tl})\text{)}
\end{align*}
\]
\( \neg p(s): \)

\[
\land: (1_1 + 1_A \times \text{unfold}'(p \rightarrow 1_*, (f, g))) \circ (p \rightarrow 1_*, (f, g))(s) \\
= (1_1 + 1_A \times \text{unfold}'(p \rightarrow 1_*, (f, g)))(f(s), g(s)) \\
= (2, (f(s), \text{unfold}'(p \rightarrow 1_*, (f, g))(g(s)))) \\
\]

Uniqueness: We prove uniqueness by supposing the existence of another \( F \)-coalgebra homomorphism \( h : S \rightarrow L \), indicated by the commutative diagram

\[
\begin{array}{c}
S \\
\downarrow h \\
L
\end{array} \quad \begin{array}{c}
p \rightarrow 1_*, (f, g) \\
\rightarrow \text{unfold}'(p \rightarrow 1_*, (f, g))(s) \\
p_e \rightarrow 1_*, (hd, tl)
\end{array} \quad \begin{array}{c}
1 + A \times S \\
\downarrow 1_1 + 1_A \times h \\
1 + A \times L
\end{array}
\]

and then showing that \( h = \text{unfold}'(p \rightarrow 1_*, (f, g)) \) for all \( s \in S \). For the case that \( p(s) \) is true, evaluation of \( h \) does not involve recursion, and equality is essentially established by the assumption that \( h \) is an \( F \)-coalgebra homomorphism. For the case that \( p(s) \) is false, evaluation of \( h \) involves recursive evaluation, and equality is established by the approximation lemma. Next, we show equality for each case.

\( p(s): \) For the case where \( p(s) = \text{True} \), going clockwise in Diagram 39, since \( p(s) = \text{True} \), \( s \mapsto (1, 1_*(s)) = (1, *) \). Since Diagram 39 commutes by assumption, this means that the anticlockwise path evaluates to \( (1, *) \) as well. The only way this could happen is if \( p_e(h(s)) \) is True, i.e. if \( h(s) = [] \). Hence, the definition of \( F \)-coalgebra homomorphism forces \( h(s) = [] \), if \( p(s) \). So,

\[
\land: (1_1 + 1_A \times h) \circ (p \rightarrow 1_*, (f, g))(s) = (1, *) \\
\land: (p_e \rightarrow 1_*, (hd, tl)) \circ h(s) = (1, *) \\
\]

\( (h \) is an \( F \)-coalgebra homomorphism)
\neg p(s): For the case where \( p(s) = \text{False} \), we have

\[
\therefore (1 + 1_A \times h) \circ (p \to 1_\ast, (f, g))(s) = (1 + 1_A \times h)(f(s), g(s)) \quad \text{(definition)}
\]
\[
= (2, (f(s), h(g(s)))) \quad \text{(definition)}
\]
\[
\therefore (p \to 1_\ast, (hd, tl)) \circ h(s) = (2, (hd(h(s)), tl(h(s)))) \quad \text{(definition)}
\]

Since \( h \) is an \( F \)-coalgebra homomorphism, so that Diagram 39 commutes, we have the following equalities for all \( s \in S \), such that \( \neg p(s) \):

\[
hd(h(s)) = f(s)
\]
\[
tl(h(s)) = h(g(s))
\]
\[
\Rightarrow h(s) = f(s) \cdot h(g(s)) \quad (*)
\]

We need to show that \( \text{approx}(n, \text{unf}oldL'(p \to 1_\ast, (f, g))(s)) = \text{approx}(n, h(s)) \) by induction on \( n \) to use the approximation lemma.

Base—0: Immediate, since \( \text{approx}(0, xs) = \bot \) for all \( xs \).

Assumption—\( n \): \( \text{approx}(n, \text{unf}oldL'(p \to 1_\ast, (f, g))(s)) = \text{approx}(n, h(s)) \)

Induction step—\( n + 1 \):

\[
\begin{align*}
\text{approx}((n + 1), \text{unf}oldL'(p \to 1_\ast, (f, g))(s)) \\
= \text{approx}((n + 1), (f(s) \cdot \text{unf}oldL'(p \to 1_\ast, (f, g))(g(s)))) & \quad \text{(definition of unf}oldL') \\
= f(s) \cdot \text{approx}(n, \text{unf}oldL'(p \to 1_\ast, (f, g))(g(s))) & \quad \text{(definition of approx)} \\
= f(s) \cdot \text{approx}(n, h(s)) & \quad \text{(by the inductive assumption)} \\
= \text{approx}((n + 1), (f(s) \cdot h(g(s)))) & \quad \text{(definition of approx)} \\
= \text{approx}((n + 1), h(s)) & \quad \text{(by equality (*), above)}
\end{align*}
\]

Therefore, by the approximation lemma, \( h = \text{unf}oldL'(p \to 1_\ast, (f, g)) \).
There is a corresponding anamorphism for postpend-constructed lists, \( \text{unfoldL}(p \rightarrow 1_*, (f, g)) : S \rightarrow L \), i.e., for the category of coalgebras on the functor \( F_A : S \mapsto 1 + S \times A, f \mapsto 1_1 + f \times 1_A \). The definition and proof follow \( \text{unfoldL}' \), but with the arguments associated with the product reversed.

Trees

Like lists, the development of a final coalgebra for trees derives from its dual initial algebra (see [3]). That is, as the dual of an initial algebra in the category of \( F \)-algebras \( \text{Alg}(F_A) \) on the functor \( F_A : \text{CPO} \rightarrow \text{CPO} ; S \mapsto A + S \times S, f \mapsto 1_A + f \times f \). A final coalgebra in this category is \((T, (p_\langle \rightarrow fmleaf, fmbranch)) \), where condition \( p_\langle : T \rightarrow \text{Bool} \) tests whether \( t \in T \) is a leaf (i.e., \( t = \langle a \rangle, a \in A \)), \( fmleaf : T \rightarrow A, (a) \mapsto a \) returns the value from a leaf, and \( fmbranch : T \rightarrow T \times T, (l, r) \mapsto (l, r) \) return the left and right subtrees from a branch. The dual category \( \text{CoAlg}(F_A) \) has as objects the \( F_A \)-coalgebras \((S, (p \rightarrow f, g)) \), where \( f : S \rightarrow A \) and \( g : S \rightarrow S \times S \), and \( F_A \)-coalgebra homomorphisms as morphisms. The anamorphism associated with this final coalgebra is called \( \text{unfoldT} \) (i.e., unfold for trees), defined recursively as \( \text{unfoldT}(p \rightarrow f, g) : S \rightarrow T \)

\[
\begin{align*}
\text{unfoldT}(p \rightarrow f, g) & : S \rightarrow T \\
S & \xrightarrow{p \rightarrow f, g} A + S \times S \\
 & \xrightarrow{1_A + \text{unfoldT}(p \rightarrow f, g) \times \text{unfoldT}(p \rightarrow f, g)} A + T \times T \\
\end{align*}
\]

**Proposition.** If \( (p \rightarrow f, g) : S \rightarrow A + S \times S \) is an arbitrary conditional function, then \( \text{unfoldT}(p \rightarrow f, g) \) is an anamorphism.

**Proof.** The proof follows a similar pattern to \( \text{unfoldL}' \). First we prove existence (commutativity of Diagram 40), then uniqueness (by case analysis and the approximation lemma, using the \text{approx} function for trees, see below).

Existence (commutativity):
\[ y : (1 + \text{unfold}_T(p \to f;g)) \circ (p \to f;g)(s) \]
\[ = (1 + \text{unfold}_T(p \to f;g))(1, f(s)) \] (definition)
\[ = f(s) \] (definition)
\[ \sim : (p_1 \to \text{fmleaf, fmbranch}) \circ \text{unfold}_T(p \to f;g)(s) \]
\[ = (p_1 \to \text{fmleaf, fmbranch})(f(s)) \] (definition)
\[ = f(s) \] (definition)

\[ \neg p(s) : \]
\[ \sim : (1 + \text{unfold}_T(p \to f;g)) \circ (p \to f;g)(s) \]
\[ = (1 + \text{unfold}_T(p \to f;g))(2, g(s)) \] (definition)
\[ = (\text{unfold}_T(p \to f;g)(p_1 \circ g(s)), \text{unfold}_T(p \to f;g)(p_2 \circ g(s))) \] (definition)
\[ \sim : (p_1 \to \text{fmleaf, fmbranch}) \circ \text{unfold}_T(p \to f;g)(s) \]
\[ = (p_1 \to \text{fmleaf, fmbranch})(\text{unfold}_T(p \to f;g)(p_1 \circ g(s)), \text{unfold}_T(p \to f;g)(p_2 \circ g(s))) \] (definition)
\[ = (\text{unfold}_T(p \to f;g)(p_1 \circ g(s)), \text{unfold}_T(p \to f;g)(p_2 \circ g(s))) \] (definition)

Uniqueness: Following the same form of reasoning as for lists, we prove uniqueness by supposing another \( F \)-coalgebra homomorphism \( h = S \to T \), indicated by the commutative diagram

\[ S \xrightarrow{p \to f;g} A + S \times S \]
\[ h \downarrow \]
\[ T \xrightarrow{p_1 \to \text{fmleaf, fmbranch}} A + T \times T \] (41)

and then showing that \( h = \text{unfold}_T(p \to f;g) \) for all \( s \in S \). We show equality for the two cases, i.e.,
where \( p(s) \) is true, and \( p(s) \) is false.

\( p(s) \): For the case where \( p(s) = \text{True} \), going clockwise in Diagram 41, \( s \mapsto f(s) \). Since Diagram 41 commutes by assumption, the anticlockwise path evaluates to \( f(s) \) too, which can only happen if \( h(s) = f(s) \). So,

\[
\land: (1_A + h \times h) \circ (p \rightarrow f, g)(s) = f(s) \quad \text{(definition)}
\]

\[
\land: (p(s) \rightarrow \text{fmlf}, \text{fmbnch}) \circ h(s) = f(s) \quad \text{\( h \) is \( F \)-coalgebra homomorphism)}
\]

\( \neg p(s) \): For the case where \( p(s) = \text{False} \), we have

\[
\land: (1_A + h \times h) \circ (p \rightarrow f, g)(s) = (1_A + h \times h)(2, (p_1 \circ g(s), p_2 \circ g(s))) \quad \text{(definition)}
\]

\[
= (2, (h \circ p_1 \circ g(s), h \circ p_2 \circ g(s))) \quad \text{(definition)}
\]

\[
\land: (p(s) \rightarrow \text{fmlf}, \text{fmbnch}) \circ h(s) = (2, \text{fmbnch}(h(s))) \quad \text{(definition)}
\]

Since \( h \) is an \( F \)-coalgebra homomorphism, so that Diagram 41 commutes, we have the following equality:

\[
\text{fmbnch}(h(s)) = (h \circ p_1 \circ g(s), h \circ p_2 \circ g(s))
\]

\[
\Rightarrow \quad h(s) = (h \circ p_1 \circ g(s), h \circ p_2 \circ g(s)) \quad (**)
\]

We need to show that \( \text{approx}(n, \text{unfoldT}(p \rightarrow f, g)(s)) = \text{approx}(n, h(s)) \) by induction on \( n \). To do so, we first need to provide the function \( \text{approx} \) for trees, which is defined as \( \text{approx} : \mathbb{N} \times T \rightarrow T \) where:

\[
\text{approx}((n + 1), \langle a \rangle) = (a)
\]

\[
\text{approx}((n + 1), \langle l, r \rangle) = (\text{approx}(n, l), \text{approx}(n, r))
\]

By case exhaustion, \( \text{approx}(0, t) = \bot \). Induction on \( n \) proceeds as follows.

Base—0: Immediate, since \( \text{approx}(0, t) = \bot \) for all \( t \).

Assumption—\( n \): \( \text{approx}(n, \text{unfoldT}(p \rightarrow f, g)(s)) = \text{approx}(n, h(s)) \)
Induction step—$n + 1$:

\[
\text{approx}((n + 1), \text{unfold}(p \to f, g)(s))
\]

\[
= \text{approx}((n + 1), (\text{unfold}(p \to f, g)(p_1 \circ g(s))),
\]

\[
\text{unfold}(p \to f, g)(p_2 \circ g(s)))
\]

\[
= (\text{approx}(n, \text{unfold}(p \to f, g)(p_1 \circ g(s))),
\]

\[
\text{approx}(n, \text{unfold}(p \to f, g)(p_2 \circ g(s)))
\]

\[
= (\text{approx}(n, h(p_1 \circ g(s))), \text{approx}(n, h(p_2 \circ g(s))))
\]

\[
= \text{approx}((n + 1), (h \circ p_1 \circ g(s), h \circ p_2 \circ g(s)))
\]

\[
= \text{approx}((n + 1), (h(s)))
\]

(by equality **, above)

Therefore, by the approximation lemma, $h = \text{unfold}(p \to f, g)$.

\[\square\]

**Universal constructions**

**(Co)Universal morphisms**

**Definition (Universal morphism).** Given a functor $F : A \to C$ and an object $Y \in |C|$, a universal morphism from $F$ to $Y$ is a pair $(A, \phi)$ where $A$ is an object of $A$, and $\phi$ is a morphism in $C$, such that for every object $X \in |A|$ and every morphism $f : F(X) \to Y$, there exists a unique morphism $h : X \to A$, such that $\phi \circ F(h) = f$, as indicated by commutative diagram

\[
\begin{array}{ccc}
X & \rightarrow & F(X) \\
\downarrow h & & \downarrow f \\
A & \rightarrow & F(A) \\
\downarrow \phi & & \downarrow Y \\
& & \\
\end{array}
\]

**Definition (Couniversal morphism).** Given an object $X \in |C|$ and a functor $F : B \to C$, a couniversal morphism from $X$ to $F$ is a pair $(B, \psi)$ where $B$ is an object of $B$, and $\psi$ is a morphism in $C$, such that for every object $Y \in |B|$ and every morphism $f : X \to F(Y)$, there exists a unique morphism
k : B → Y, such that \( F(k) \circ \psi = f \), as indicated by commutative diagram

\[
\begin{array}{c}
X \xrightarrow{\psi} F(B) \\
\downarrow f \\
F(Y) \xrightarrow{\psi} Y
\end{array}
\]

\[
\begin{array}{c}
B \\
\downarrow k \\
Y
\end{array}
\]

(43)

Definition (Universal construction). A *universal construction* is either a universal morphism, or (its dual) a couniversal morphism.

Adjunctions

A special kind of universal construction is called an *adjunction*, or *adjoint situation*. Our explanations for systematicity with regard to non-recursive domains employed this form of universal construction. Here, we provide definitions for comparison with initial algebras as the basis for our explanation of systematicity with regard to recursive domains.

Definition (Adjunction—unit version). An adjunction from categories \( C \) to \( D \) is a triple \((F, G, \eta) : C \rightarrow D \) where \( F : C \rightarrow D \) and \( G : D \rightarrow C \) are functors, and \( \eta : 1_C \rightarrow G \circ F \) is a natural transformation, such that for every \( X \in |C| \), \((F(X), \eta_X)\) is a couniversal morphism from \( X \) to \( G \), as indicated in the following commutative diagram:

\[
\begin{array}{c}
X \xrightarrow{\eta_X} G \circ F(X) \\
\downarrow f \\
G(Y) \xrightarrow{\eta_Y} Y
\end{array}
\]

(44)

The relationship between \( F \) and \( G \) is called an *adjoint situation*, where \( F \) is the *left adjoint* of \( G \) (written \( F \dashv G \)), and \( G \) is the *right adjoint* of \( F \).

Definition (Adjunction—counit version). An adjunction from categories \( C \) to \( D \) is a triple \((F, G, \epsilon) : C \rightarrow D \) where \( F : C \rightarrow D \) and \( G : D \rightarrow C \) are functors, and \( \epsilon : F \circ G \rightarrow 1_D \) is a natural transformation, such that for every \( Y \in |D| \), \((G(Y), \epsilon_Y)\) is a universal morphism from \( F \) to \( Y \), as indicated in the following
Recall that, given objects $A$ and $B$ in a category $\mathbf{C}$, the set $\mathbf{C}(A,B)$ is the set of all morphisms from $A$ to $B$. This set is sometimes referred to as a *hom-set*, where “hom” is short for homomorphism even though a hom-set may contain morphisms that are not specifically homomorphisms.

**Definition (Adjunction—hom-set version).** An adjunction from categories $\mathbf{C}$ to $\mathbf{D}$ is a triple $(F,G,\psi) : \mathbf{C} \to \mathbf{D}$, where $F : \mathbf{C} \to \mathbf{D}$ and $G : \mathbf{D} \to \mathbf{C}$ are functors, and $\psi$ is a natural isomorphism sending each pair of objects $X,Y \in \mathbf{C}^{\text{op}} \times \mathbf{D}$ to a bijection of hom-sets $\mathbf{D}(F(X),Y) \cong \mathbf{C}(X,G(Y))$ that is natural in $X$ and $Y$.

Various universal constructions are unified as special cases of a category theory construction called a *comma category*. Hence, the relationship between the universal construction employed here to explain systematicity with respect to recursive domains, and systematicity with respect to non-recursive domains [13, 14] is made explicit and formal via comma categories. More complete details of comma categories and their relationships to various universal constructions are given in [14] (Text S2).

**References**

1. Mac Lane S (2000) Categories for the working mathematician. Graduate Texts in Mathematics. New York, NY: Springer, 2nd edition.

2. Awodey S (2006) Category theory. Oxford Logic Guides. New York, NY: Oxford University Press.

3. Manes EG, Arbib MA (1986) Algebraic approaches to program semantics. New York, NY: Springer-Verlag.

4. Meijer E, Fokkinga M, Paterson R (1991) Functional programming with bananas, lenses, envelopes and barbed wire, Berlin, Germany: Springer-Verlag, volume 523 of *Lecture notes in computer science*. pp. 125–144.

5. Bird R, de Moor O (1997) Algebra of programming. Harlow, England: Prentice Hall.
6. Pierce BC (1991) Basic category theory for computer scientists. Foundations of Computing. Cambridge, UK: MIT Press.

7. Bird R (1998) Introduction to functional programming using Haskell. Prentice Hall Series in Computer Science. Harlow, England: Prentice Hall, 2nd edition.

8. Jones SP, editor (2003) Haskell 98 language and libraries: the revised report. Cambridge, UK: Cambridge University Press.

9. Uustalu T, Vene V (1999) Primitive (co)recursion and course-of-value (co)iteration, categorically. Informatica 10: 5–26.

10. Gibbons J, Hutton G (2005) Proof methods for corecursive programs. Fundamenta Informaticae 66: 353–366.

11. Hutton G, Gibbons J (2001) The generic approximation lemma. Information Processing Letters 79: 197–201.

12. Jacobs B, Rutten J (1997) A tutorial on (co)algebras and (co)induction. Bulletin of the European Association for Theoretical Computer Science 62: 222–259.

13. Phillips S, Wilson WH (2010) Categorial compositionality: A category theory explanation for the systematicity of human cognition. PLoS Computational Biology 6: e1000858.

14. Phillips S, Wilson WH (2011) Categorial compositionality II: Universal constructions and a general theory of (quasi-)systematicity in human cognition. PLoS Computational Biology 7: e1002102.