Implicit complexity for coinductive data: a proof-theoretic characterization of primitive corecursion

Daniel Leivant and Ramyaa Ramyaa
Indiana University
{*leivant,ramyaa*}*@cs.indiana.edu*

Abstract

We extend the framework of intrinsic theories, which we developed for inductive data, to encompass coinductive data as well. We extend the fundamental theorem of intrinsic theories for program termination (which links computability to model-theory) from systems of inductive data to coinductive data, that is: an equational program P is type-correct in the canonical (intended) structure iff P (construed as a formula) entails the type-correctness of P in all "data-exact" structures. For inductive data type-correctness means termination, and for coinductive data it means fairness.

Our main technical result is an implicit characterization of primitive-corecursion: A function is definable using primitive corecursion iff its data-correctness is probable (in the intrinsic theory) using coinduction only for formulas in which data-predicates do not occur negatively. This is an analog for corecursion of Parson's 1960 characterization of the primitive recursive functions by provability in arithmetic with induction restricted to existential formulas.

1 Introduction

Coinductive algebras have been recognized for long as an important foundational approach to processes and dynamic systems. The significance of coinductive methods for general computing over data has been illustrated more recently, for example [2, 3] have demonstrated the use of streams of signed digits to carry exact real-number computations.

Here we consider equational programs over coinductive data, and more generally over "data-systems", which may be defined using both inductive and co-inductive constructions. Equational programs are particularly attractive to foundational studies, because of their immediate kinship with formal theories: programs can be viewed as axioms, and computation as derivations in equational logic. Consequently, equational programs are attractive for developing methods of implicit complexity, that is machine-independent characterizations of computational complexity classes. For instance, there are illuminating characterizations of various complexity classes in terms of the strength of proof methods needed to prove termination (see e.g. [8, 11]). Such methods are of particular interest for coinductive data, because they are likely to clarify complexity notions that are dual to traditional notions of computational complexity (such as P-Time) for symbolic, i.e. inductive, data.

In this paper we first develop several building blocks for this project. Referring to a notion of data-systems, we consider the global semantics of programs, that is their behavior as "uninterpreted programs" over all structures for the vocabulary of any given data-system. We developed this approach for inductive data [10], and we consider here its extension to data-systems in general, including coinductive constructions. This approach is orthogonal to the use of category theoretical methods in the study of coinduction, and focuses instead on computational aspects. It seems natural, however, to expect that natural relations would be found between the two approaches, Category Theory and Global Semantics.

The main technical result of this introdcutory discussion is an Adequacy Theorem for the global semantics of equational programs (over arbitrary data-systems). This theorem establishes the equivalence of the correctness of programs as they execute over the intended algebraic/coalgebraic interpretation, and their correctness as they execute uniformly over all structures. Here by "program correctness" we mean, roughly, termination for inductive data, and fairness for coinductive data.

An important benefit of streamlined proof systems for reasoning about programs is their potential application to characterize major computational complexity classes. Such characterizations fall within the realm of *implicit computational complexity,* a research area that strives to delineate such classes without reference to computing models, thereby gaining insight into the significance of complexity classes, providing natural frameworks for programming within given complexity boundaries, suggesting static analysis tools for guaranteeing complexity, and suggesting notions of complexity for new paradigms.

We proceed in this paper with a framework for a

foundational study of computating over coinductive data in general, and using the schema of primitive correcurrence in particular. The primitive *recursive* functions, over the natural numbers, were characterized already in by Parsons [13], as being exactly the functions that are provable in Peano Arithmetic using induction for existential formulas only. In [9, 10] we developed a generic framework for reasoning about equational computing over inductive data, and in [11] we used it to show that the primitive recursive functions are precisely the ones that are proved (in a simple sense) using induction restricted to data-positive formulas, i.e. formulas in which atomic references to data do not occur in negative positions (i.e. within the negative scope of an odd number of implications and negations).

Here we prove a dual result for coinductive data: a primitive corecursive function over streams of digits is fair (maps streams to streams) iff it is provable using coinduction for positive formulas.

2 Intrinsic theories for inductive and coinductive data

2.1 Data systems

We start out by setting a general framework for dealing jointly with data-types that are defined using any mix of induction and coinduction. Such frameworks are well-known within typed lambda calculi, with operators μ for smallest fixpoint and ν for greatest fixpoint. Our approach here is greared toward proof theoretic analysis, based on equational programs, which we also examine in a model-theoretic context.

We define a *constructor-vocabulary* as a finite set $\mathcal{C}$ of function identifiers, referred to as *constructors,* each assigned an *arity* ≥ 0. (Constant constructors have arity 0.) We posit an infinite set $\mathcal{X}$ of *variables,* and an infinite set $\mathcal{F}$ of function-identifiers, dubbed *program-functions,* and assigned arities ≥ 0 as well. The sets $\mathcal{C}$, $\mathcal{X}$ and $\mathcal{F}$ are, of course, disjoint.

If $\mathcal{E}$ is a set consisting of function-identifiers and (possibly) variables, we write $\bar{\mathcal{E}}$ for the terms generated from $\mathcal{E}$ by application: if $g \in \mathcal{E}$ is a function-identifier of arity r, and $t_1 \dots t_r$ are terms, then so is $g\, t_1 \cdots t_r$. We use informally the parenthesized notation $g(t_1, \dots, t_r)$, when convenient. We refer to $\bar{\mathcal{C}}$, $\overline{\mathcal{C} \cup \mathcal{X}}$ and $\overline{\mathcal{C} \cup \mathcal{X} \cup \mathcal{F}}$ as the *data-terms, base-terms* and *program-terms,* respectively.

A *data-system* (over $\mathcal{C}$) consists of

1. A list $D_1 \dots D_k$ (the order matters) of unary relation-identifiers, where each D_n is designated as either an *inductive-predicate* or a *coinductive-predicate,* and associated a set $\mathcal{C}_n \subseteq \mathcal{C}$ of constructors.

2. For each inductive-predicate D_n an *inductive definition:* for $\mathbf{c} \in \mathcal{C}_n$, of arity r say, a *data-introduction* rule, of the form

$$\frac{Q_1(x_1) \quad \cdots \quad Q_r(x_r)}{D_n(\mathbf{c}\, x_1 \cdots x_r)}$$

where each Q_i is one of $D_1 \dots D_n$. These rules delineate the intended meaning of an inductive D_n from below.

3. For each coinductive-predicate D_n a *coinductive definition:* for $\mathbf{c} \in \mathcal{C}_n$, of arity r say, r *data-elimination* rules, of the form

$$\frac{D_n(\mathbf{c}\, x_1 \cdots x_r)}{Q_i(x_i)}$$

where each Q_i is one of $D_1 \dots D_n$. These rules delineate the intended meaning of a coinductive D_n from above.

Example. Let $\mathcal{C}$ consist of the identifiers $\mathtt{0}, \mathtt{1}, [], \mathtt{s}, \mathtt{t}$, and $\mathtt{c}$, of arities 0,0,0,1,1, and 2, respectively. Consider the following data-system, with two inductive definitions, for the booleans and for the natural numbers, followed by a coinductive definition for the infinite $\mathtt{s}/\mathtt{t}$-words and for the streams of natural numbers, and finally an inductive definition of the lists of such streams.

1. $\dfrac{}{B(\mathtt{0})} \qquad \dfrac{}{B(\mathtt{1})}$

2. $\dfrac{}{N(\mathtt{0})} \qquad \dfrac{N(x)}{N(\mathtt{s}x)}$

3. $\dfrac{W(\mathtt{s}x)}{W(x)} \qquad \dfrac{W(\mathtt{t}x)}{W(x)}$

4. $\dfrac{S(\mathbf{c}xy)}{N(x)} \qquad \dfrac{S(\mathbf{c}xy)}{S(y)}$

5. $\dfrac{}{L([])} \qquad \dfrac{S(x) \quad L(y)}{L(\mathbf{c}xy)}$

Note that constructors are being reused for different data-types. This is in agreement with our untyped, generic approach, with the intended type-information conveyed by the data-predicates. □

The *algebraic-interpretation* $[\![\mathcal{D}]\!]$ of a data-system $\mathcal{D}$ is defined by successively interpreting each inductive-predicate D_n as the initial algebra for $\mathcal{C}_n$,

based on the interpretations of $D_1 \dots D_{n-1}$, and — dually — each coinductive D_n as the final coalgebra for $\mathcal{C}_n$. Thus, the algebraic interpretation is the intended model of the data-system.

2.2 Intrinsic Theories

Intrinsic theories, introduced in [9, 10] for inductive data, are skeletal first-order theories whose interest lies in a natural and streamlined formalization of reasoning about equational computing. For example, the intrinsic theory for the natural numbers is perfectly suited for incorporating equational programs as axioms, and while it has the same provably computable functions as Peano Arithmetic, it has a far more immediate formalization of the notion of provable computability. For background, rationale, and examples, please see [10].

The *intrinsic theory* for a data-system $\mathcal{D}$, $\mathbf{IT}(\mathcal{D})$, has

1. The rules of $\mathcal{D}$,
2. *Injectiveness axioms* stating that the constructors are injective, i.e. for each $\mathbf{c} \in \mathcal{C}$, of arity r,
$$\forall x_1 \dots x_r\, y_1 \dots y_r \;\; \mathbf{c}(\vec{x}) = \mathbf{c}(\vec{y}) \rightarrow \bigwedge_i x_i = y_i$$
3. *Separation axioms* stating that the constructors have disjoint images:
$$\forall \vec{x}\, \vec{y} \;\; \mathbf{c}\vec{x} \neq \mathbf{d}\vec{y}$$
for each distinct constructors $\mathbf{c}, \mathbf{d}$.
4. Rules complementary to the rules of $\mathcal{D}$:
   - For each inductive data-predicate D_n as above, a data-elimination (i.e. Induction) rule: for any formula* $\varphi \equiv \varphi[x]$, the rule infering $\varphi[t]$ from
     (a) $D_n(t)$,
     (b) For each $\mathbf{c} \in \mathcal{C}_n$ a derivation of $\varphi[\mathbf{c}x_1 \cdots x_r]$ from the assumptions $Q_1^{\varphi}(x_1) \dots Q_r^{\varphi}(x_r)$, where $Q_i^{\varphi}(u)$ is $\varphi[u]$ if Q_i is D_n, and is $Q_i(u)$ otherwise. (These open assumptions are closed by the inference.) We say then that $\mathbf{c}$ has type $(Q_1, \dots, Q_r) \rightarrow D_n$.
   - For each coinductive data-predicate D_n, a data-introduction (i.e. Coinduction) rule: for any formula $\varphi[x]$, the rule infers $D_n(t)$ from
     (a) $\varphi[t]$
     (b) For each $\mathbf{c} \in \mathcal{C}_n$ (of arity r) and $i = 1 \dots r$, a derivation of $Q_i^{\varphi}(x_i)$ from the assumption $\varphi[\mathbf{c}x_1 \cdots x_r]$.
     (c) A derivation of
$$\textit{In-}D_n(x) \;\equiv\; \bigvee_{\mathbf{c} \in \mathcal{C}_n} \exists y_1 \dots y_r \;\; x = \mathbf{c}\, y_1 \cdots y_r$$
     from $\varphi[x]$. We refer to this implication as the *bounding condition.*
     Again, we say then that $\mathbf{c}$ has type $(Q_1, \dots, Q_r) \rightarrow D_n$.

Note that each constructor $\mathbf{c}$ has as many types as there are $\mathcal{C}_n$ in which it appears.

Note. Our general approach here is model theoretic, that is we consider arbitrary structures for the given vocabulary (= signature). In particular, equality is interpreted in each structure as true equality. It follows that the bounding-condition in the statement of Coinduction is necessary. Consider for example the coinductive data W built from unary function identifiers $\mathbf{s}$ and $\mathbf{t}$, i.e. the infinite words over $\{s, t\}$. Taking the eigen formula φ of Coinduction to be $x = x$, we would get, absent the bounding-condition, that $\forall x\, W(x)$, which is not valid in models of the intrinsic theory for W.

From the injectiveness and separation axioms it follows that it is innocuous to introduce identifiers for destructor functions, $\delta_{n,i}$ ($n = 1 \dots k$, $i \leq$ the maximal arity of constructors in $\mathcal{C}_n$), and posit as axioms the equations

$$\delta_{n,i}(\mathbf{c}\, x_1 \cdots x_r)) = x_i$$

The injectiveness axioms can then be dispensed with (they become provable), and the formulas $\textit{In-}D_n$ in the bounding-conditions above can be simplified by stating the closure-subderivations using destructors rather than the constructors.

*We use the bracket notation $\varphi[t]$ to stand for the correct substitution in φ of t for the free occurrences of some fixed variable z.

2.3 Equational programs

As in [9, 10], we use an equational computation model, in the style of Herbrand-Gödel, familiar from the extensive literature on algebraic semantics of programs. There are easy translations between equational programs and the program-terms of Moschovakis's $\mathbf{FLR}_0$ [12]; however, equational programs integrate easily into logical calculi, because they can be construed as equational theories; codifying them as terms is a redundancy, since the computational behavior of such terms is itself spelled out using equations or rewrite-rules.

Given a data-system as above, a *program-equation* is an equation of the form $\mathbf{f}(t_1 \dots t_k) = q$, where $\mathbf{f}$ is a program-function of arity k, $t_1 \dots t_k$ are base-terms, and q is a program-term. The identifier $\mathbf{f}$ is dubbed the *lead-function* of the equation, and the tuple $\langle t_1 \dots t_k \rangle$ its *case.* Two program-equations are *compatible* if they have distinct lead functions, or else have cases that cannot be unified. A *program-body* is a finite set of pairwise-compatible program-equations. A program $(P, \mathbf{f})$ consists of a program-body P and a program-function $\mathbf{f}$, dubbed the program's *principal-function.* We identify a program with its program-body when in no danger of confusion. We write V_P for the vocabulary of P, i.e. $\mathcal{C}$ augmented with the program-functions used in P.

It is easy to define the denotational semantics of an equational program for (the algebraic interpretation of) inductive data, since such data is finite. For example, if $(P, \mathbf{f})$ is a program for a unary function over $\mathbb{N}$, then it computes the partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ when $f(p) = q$ just in case the equation $\mathbf{f}(\bar{p}) = \bar{q}$ is derivable from P in equational logic, where $\bar{n}$ is the n'th numeral, i.e. $\mathbf{ss} \cdots \mathbf{s0}$ with n $\mathbf{s}$'s (see e.g. [10] for generalizations and detail).

Since coinductive data is (in general) infinite, defining the semantics of equational programs must refer to finite information about the output. It must also refer to the data-objects not as syntactic terms that can be present in equations, but as values bound to variables.

We posit then a structure $\mathcal{S}$ (for the vocabulary of a data-system $\mathcal{D}$), and define the partial function computed by a program $(P, \mathbf{f})$ by refering to the interpretation of the constructors in $\mathcal{S}$. We say that a program $(P, \mathbf{f})$ (with $\mathbf{f}$ unary, say) computes over $\mathcal{S}$ the partial-function[†] $f : |\mathcal{S}| \rightharpoonup |\mathcal{S}|$ if the following are equivalent:

1. $f(a) = b$, for $a, b \in |\mathcal{S}|$;
2. $\mathcal{S}^*, [x, y := a, b] \models \mathbf{f}(x) = y$ for all expansions of $\mathcal{S}$ to the vocabulary of P, which satisfy P.

By the semantic completeness of equational logics this is equivalent to to the following definition. Consider fresh auxiliary variables, one variable $\mathtt{v}_a$ for each $a \in |\mathcal{S}|$. Let the *diagram* of $\mathcal{S}$ be the theory

$$\mathrm{Diag}(\mathcal{S}) \quad = \quad \{\mathtt{v}_a = \mathbf{c}\mathtt{v}_{b_1} \cdots \mathtt{v}_{b_r} \mid a = \mathbf{c}b_1 \cdots b_r\}$$

Then $(P, \mathbf{f})$ computes $f : |\mathcal{S}| \rightharpoonup |\mathcal{S}|$ iff for every $a, b \in |\mathcal{S}|$ we have $f(a) = b$ iff the equation $\mathbf{f}(\mathtt{v}_a) = \mathtt{v}_b$ is derivable in equational logic from $P \cup \mathrm{Diag}(\mathcal{D})$.

Note. A program $(Q, \mathbf{g})$ whose principal-function $\mathbf{g}$ is of arity 0 is a *data-definition.* For example, the singleton programs $\{\mathbf{f} = \mathbf{ss0}\}$ and $\{\mathbf{g} = \mathbf{sg}\}$ are data-definition of the natural number 2 (in unary) and of the infinite word $\mathbf{sss} \cdots$, respectively. The singleton program $\{\mathbf{h} = \mathbf{j}(\mathbf{0})\}$ is also a data-definition, as is the program consisting of $\mathbf{h} = \mathbf{j}(\mathbf{0})$ and of equations that engender a non-terminating computation of $\mathbf{j}$ for input $\mathbf{0}$. However, the former two data-definitions can be used to prove in the intrinsic theory that the objects defined are a natural number and an infinite word, respectively, in any structure, whereas nothing of interest can be said about the objects defined by the latter two data-definitions.

3 Global semantics of programs over inductive data

3.1 Global semantics for equational programs

The concept of *global relations,* which was present implicitly in mathematical logic for long, came to prominence in Finite Model Theory in the 1980s.

Let $\mathcal{C}$ be a collection of structures. A *global relation* (of arity r) over $\mathcal{C}$ is a mapping $\mathcal{P}$ that assigns to each structure $\mathcal{S}$ in $\mathcal{C}$ an r-ary relation over the universe $|\mathcal{S}|$ of $\mathcal{S}$. For example, if $\mathcal{C}$ is the collection of all structures over a given vocabulary V, then a first-order V-formula φ, with free variables among $x_1 \dots x_r$, defines the predicate $\lambda x_1 \dots x_r \varphi$ that to each V-structure $\mathcal{S}$ assigns the relations[‡].

$$\{\langle a_1 \dots a_r \rangle \mid \mathcal{S}, [\vec{x} := \vec{a}] \models \varphi\}$$

[†] We write $|\mathcal{S}|$ for the universe of $\mathcal{S}$, and use $\rightharpoonup$ for partial functions

[‡] The notion that a formula delineates uniformly subsets of structures is implicit already in early formalizations of Set Theory, for instance in Frege's Comprehension Principle and, in particular, in Fraenkel and Skolem's Axiom of Replacement. In relation to collections of first order structures the notion was used by Tarski [15] and in [1]. Alternative phrases used for it include *generalized relations*, *data base queries*, *global relations*, *global predicates*, *uniformly defined relations*, *predicates over oracles*, and *predicates.*

A *global r-ary function* over $\mathcal{C}$ is defined analogously. For example, each λ-term with identifiers in V as primitives, defines a global function over the class of V-structures. E.g., if `c`, `f` and `g` are V-identifiers for functions of arity 0,1 and 2 respectively, then the term $\lambda x_{,1}\, , x_2\ \mathtt{g}(\mathtt{f}(x_1), \mathtt{g}(x_2, \mathtt{c}))$ defines the global function that to each V-structure $\mathcal{S}$ assigns the mapping $\langle a_1, a_2\rangle \mapsto g(f(x_1), g(x_2, c))$, where c, f and g are the interpretations in $\mathcal{S}$ of the identifiers $\mathtt{c}, \mathtt{f}$ and ttg.

The starting point of Descriptive Computational Complexity [6] is that programs used as acceptors define global relations. When those global relations can be defined also by certain logical formulas, one obtains machine-independent characterizations of computational complexity classes. For instance, Fagin [5] and Jones & Selman [7] proved that a predicate $\mathcal{P}$ over finite structures is defined by a program running in nondeterministic polynomial time (NP) iff it is defined by a purely existential second order formula.

We focus here on global semantics for ("uninterpreted") equational programs, for arbitrary data-systems. That approach, developed for inductive data in [10], is of interest for a number of reasons. Generally, the global semantics approach is an interesting alternative to the "intended-structure" approach: the latter tackles the issue of program divergence through the development of Domain Theory, whereas the former bypasses the issue by invoking uniformity through all structures (see Corollary 2 below).

Also, under the global semantics approach the notion of *correctness* of programs is simple, direct, and informative. Here a program over inductive data is said to be *correct* if it maps, in every structure, inductive data to inductive data. This turns out to be equivalent to the program termination (for all input) in the intended structure (e.g. $\mathbb{N}$). For program over co-inductive data, which we address here, correctness will turn out to be equivalent to fairness: if the input is a stream, then the program will have an output as a stream, without stalling in producing new entries.

The simple, global, and uniform notion of program-correctness also enables a simple proof theoretic treatment of program-correctness. For example, one can define for programs over $\mathbb{N}$ a rudimentary theory of data (the "intrinsic theory of $\mathbb{N}$" in [10]), with no reference to numeric functions such as addition or multiplication, with the result that the provably-correct programs compute precisely all the provably-recursive functions of Peano Arithmetic.

Our aim here is to initiate a similar approach for equational computing over co-inductive data.

[§]We write $\bar{n}$ for the n'th numeral, i.e. $\mathbf{ss}\cdots\mathbf{s0}$ with n $\mathbf{s}$'s.

3.2 Adequacy of Global semantics for inductive data

It is easy to define the intended semantics of equational programs over *inductive* data, such as natural numbers or strings. The universe is the initial algebra, and computation proceeds using equations as rewrite rules. For instance, if $(P, \mathbf{f})$ is a program for a unary function over $\mathbb{N}$, (using a vocabulary V that includes the identifiers `s` and `0` for successor and zero), then P defines over the canonical structure $\mathbb{N}$ the partial function f determined by

$$f(p) = q \quad \text{IFF} \quad P \vdash^{=} \mathtt{f}(\bar{p}) = \bar{q}$$

where provability is in equational logic.

The global semantics of such equational programs is defined analogously, within structures, with the difference that the principal function-identifier of the program is always interpreted as a total function. That is, if $\mathcal{S}$ is a V-structure which models P (i.e. such that all equations in P are valid in $\mathcal{S}$), then $f(p) = q$ implies that $\mathcal{S} \models \mathtt{f}(\bar{p}) = \bar{q}$.[§], but the term $\mathtt{f}(\bar{p})$ will always have a value in $\mathcal{S}$, albeit not the denotation of a numeral.

The converse implication also holds, leading to the following (see [10] for generalizations):

THEOREM **1** *(*Semantic Adequacy Theorem for Inductive Data*) [10] If $(P, \mathtt{f})$ is an equational program that computes a partial function $f : \mathbb{N} \rightharpoonup \mathbb{N}$, then $f(p) = q$ iff $\mathcal{S} \models \mathtt{f}(\bar{p}) = \bar{q}$ in all models $\mathcal{S}$ of P.*

Let $\mathcal{S}$ be a structure over the functional vocabulary V above, augmented with a single unary relational identifier `N`. We say that $\mathcal{S}$ is *data-correct* if

$$[\![\mathtt{N}]\!]_{\mathcal{S}} = \{[\![\bar{n}]\!]_{\mathcal{S}} | n \in \mathbb{N}\}$$

that is, if `N` is interpreted in $\mathcal{S}$ as the set of denotations of numerals. A consequence of Theorem 1 is

COROLLARY **2** *An equational program $(P, \mathtt{f})$ over $\mathbb{N}$ computes a total function iff the formula $N(x) \rightarrow N(\mathtt{f}(x))$ is valid in every data-correct model of P.*

The proof in [10] of the non-trivial direction of Theorem 1 proceeds by constructing a "test-model" for the program P. One starts with an extended term model, using all identifiers of the program P as well the the constructor, and taking its quotient over are the equivalence relation of equality-derived-from P.

4 Global semantics of program

for data systems

4.1 Global semantics in the presence of coinductive data

The canonical semantics of equational programs over data-systems is obtained, of course, by the incremental construction of algebraic interpretations for the data-types, as initial algebras or final co-algebras.

We define the *global* semantics of such programs, using equational logic as an operational model. Here we need to refer to possibly infinitely many structure elements. For example, if a is a 01-stream, and $(P, \mathtt{f})$ is a program, the value computed within a V-structure $\mathcal{S}$ for the stream $\mathtt{f}(a)$ is generally a stream, whose finite initial segments are obtained in equational logic using P, as well as a finite amount of information about initial segments of a. Thus, the equational computing must refer to names for iterated tails of the inputs stream a and of the output streams $\mathtt{f}(a)$ in $\mathcal{S}$. That is, given a V-structure $\mathcal{S}$ we posit a collection of fresh constant identifiers $\mathbf{v}_a$ for each a in the universe $|\mathcal{S}|$ of $\mathcal{S}$. The *Diagram* of $\mathcal{S}$ is the set *Diag*($\mathcal{S}$) consisting of the equations $\mathbf{v}_a = \mathbf{c}\mathbf{v}_{b_1} \cdots \mathbf{v}_{b_r}$ that are true in $\mathcal{S}$.

If t and t' are closed terms, $\mathcal{E}$ is a set (not necessarily finite) of equations, we write $\mathcal{E} \vdash t =^1 t'$ if there is a constructor $\mathbf{c}$ (of arity r) and terms $t_1 \ldots t_r, t'_1 \ldots t'_r$ such that the equations $t = \mathbf{c}t_1 \ldots t_r$ and $t' = \mathbf{c}t'_1 \ldots t'_r$ are derivable in equational logic from $\mathcal{E}$. We write $\mathcal{E} \vdash^* t = t'$ if $\mathcal{E} \vdash \Delta t =^1 \Delta t'$ for every multi-destructor Δ, where a *multi-destructor* is the composition of destructors. Intuitively, $\mathcal{E} \vdash^* t = t'$ when for every subterm-address Δ $\mathcal{E}$ provides a computation that establish that t and t' have the same constructor at that address. When t and t' evaluate to purely inductive (i.e. finite) values, then $\mathcal{E} \vdash^*_= t = t'$ can be shown to be equivalent to $\mathcal{E} \vdash_= t = t'$ (i.e. straightforward provability in equational logic).

4.2 Adequacy of global semantics for general data-systems

We now proceed to construct, given a program $(P, \mathbf{f})$ over a data-system $\mathcal{D}$, a canonical model $\mathcal{M}(P)$ to serve as a "test-structure" for the program. We start with the admitting fresh constant-identifiers $\mathtt{e}_a$ for each element a of the algebraic interpretation of the data-system. We define the *formal diagram* of $\mathcal{D}$ to be the set of equations

$$\mathit{FDiag}(\mathcal{D}) = \{\mathtt{e}_a = \mathbf{c}\mathtt{e}_{b_1} \cdots \mathtt{e}_{b_r} \mid a = \mathbf{c}b_1 \cdots b_r\}$$

Let $T(P)$ be the set of formal terms generated from the identifiers $\mathtt{e}_a$ using the program-functions occurring in P and the constructors.

Consider the relation $\approx_P$ that holds between two terms $t, t' \in T(P)$ if $P, \mathrm{FDiag}(\mathcal{D}) \vdash^*_= t = t'$. Clearly, this is an equivalence relation. The universe of $\mathcal{M}(P)$ is the quotient $T(P)/(\approx_P)$, The constructors and program-functions are interpreted in $\mathcal{M}(P)$ via symbolic application: the interpretation of a function-identifier $\mathbf{g}$, unary say, maps an equivalence-class $[t]_\approx$ to the equivalence class $[\mathbf{g}t]_\approx$.

Finally, each data-predicate D_n is interpreted as

$$[\![D_n]\!]_{\mathcal{M}} \quad = \quad \{[\mathtt{e}_a] \mid a \in [\![D_n]\!]\}$$

Next we generalize the definition of data-correct structures to arbitrary $\mathcal{D}$. We say that $\mathcal{S}$ is *data-correct* if

1. Each inductive data-predicate D_n is interpreted in $\mathcal{S}$ as the smallest subset of $|\mathcal{S}|$ which is closed under the data-introduction rules of the data-system $\mathcal{D}$ (for D_n).

2. Each coinductive data-predicate D_n is interpreted in $\mathcal{S}$ as the largest subset of $|\mathcal{S}|$ which is

   (a) Contained in the image of all constructors in $\mathcal{C}_n$, i.e. a subset of $\cup_{c \in \mathcal{C}_n} \mathbf{c}(|\mathcal{S}|)$; and

   (b) Is closed under the data-elimination rules of the data-system $\mathcal{D}$ (for D_n).

   (Note that $\mathcal{S}$ may have a multitude of elements corresponding to the same element of $[\![D_n]\!]$.)

LEMMA **3** *$\mathcal{M}(P)$ is a data-correct model of P.*

THEOREM **4** *Let $(P, \mathbf{f})$ be a program over a data-system $\mathcal{D}$, with $\mathbf{f}$ k-ary. The following are equivalent.*

1. *The program $(P, \mathbf{f})$ computes on $[\![\mathcal{D}]\!]$ a partial-function that maps inputs $a_1 \in D_{n_1}, \ldots, a_k \in D_{n_k}$ to an output in D_{n_0}. (I.e. non-terminating or aborting computations might occur only for inputs of different data-types.)*

2. *The formula*

$$\forall x_1 \ldots x_k \bigwedge_i D_{n_i}(x_i) \quad \rightarrow \quad D_{n_i}(\mathbf{f}\vec{x})$$

   is true in $\mathcal{M}(P)$.

3. *The formula above is true in all data-correct models of P.*

Proof Outline. (1) implies (3) since the equational computation of P over $[\![\mathcal{D}]\!]$ remains correct in every data-correct model of P.

To prove that (3) implies (2), we use the Lemma above. The quotient construction is designed precisely to achieve that.

Finally, (2) implies (1) by the way we define the interpretation in $\mathcal{M}(P)$ of the predicates D_n. □

Theorem 4 justifies a concept of *provable* correctness of programs: $(P, \mathbf{f})$ is provable correct in a given formal theory if the formula above is not merely true in all data-exact models of P, but is indeed provable from (the universal closure of) P, as an axiom. We will present such theories and analyze their proof theoretic strength in a future paper.

In the next section we prove that the provable functions of the intrinsic theory for streams of digits are precisely the functions over such streams that are definable using the schema of primitive corecursion.

5 Primitive Corecursion

5.1 The class of primitive corecursive functions

We say that a function f over $\mathbb{N}$ is defined by *recurrence* if it is given by the schema

$$\begin{aligned} f(0, \vec{y}) &= g_0(\vec{y}) \\ f(\mathbf{s}(z), \vec{y}) &= g_s(f(z, \vec{y}, z, \vec{y}) \end{aligned}$$

This conveys the algorithmic paradigm of decomposing the input by eager evaluation, so as to rebuild the output from initial components. Slightly more generally, in a data system as above with just one predicate, which is inductive, the definition has, for each constructor $\mathbf{c}$ of type $D^r \to D$ a clause

$$f(\mathbf{c}(x_1 \dots x_r), \vec{y}) = g_c(e_1 \dots e_r, \vec{y})$$

where each e_i is $f(x_i, \vec{y})$. Here each g_c is an already-defined function of the correct arity; that is, for each data-object x we obtain $f(x, \vec{y})$ by interpreting each occurrence of a constructor $\mathbf{c}$ as the function g_c (parameterized by $\vec{y}$). A function over the given data-system is *primitive-recursive* if it is obtained by repeated use of explicit definitions and the schema of recurrence.

The schema of *corecurrence* is the dual of recurrence: it conveys the algorithmic paradigm of building up the output while decomposing the (possibly infinite) input by a lazy evaluation. This is easily expressed for simple forms of coinductive data. Suppose C is a coalgebra (a unique coinductive predicate D) based on just one constructor c, of type $(Q, D) \to D$, where Q is not D; then corecurrence is the schema

$$f(x, \vec{y}) = \mathsf{c}(f(g(x, \vec{y}), \vec{y}), g'(x, \vec{y}))$$

Since c is the only available constructor here, this is conveyed referring to the two destructors for $\mathbf{c}$, say δ_0 and δ_1:

$$\begin{aligned} \delta_0\, f(x, \vec{y}) &= f(g(x, \vec{y}), \vec{y}) \\ \delta_1\, f(x, \vec{y}) &= g'(x, \vec{y}) \end{aligned}$$

However, using destructors prevents us from specifying the output's main constructor, whenever there is more than just one constructor present. While we focus in the sequel on the case of a single constructor (CONS for lists of digits), let us state a corecurrence principle that applies to a coinductive predicate D_n based on a set $\mathcal{C}_n$ consisting of k constructors. We invoke the function $Case(u_0, u_1, \dots, u_k)$ which returns u_i if the main constructor of u_0 is the i-th constructor in $\mathcal{C}_n$ (under some canonical ordering).

A function f is then *defined by corecurrence* from the functions h, g_i below if

$$f(x, \vec{y}) = Case(h(x, \vec{y}), e_1, \dots, e_k)$$

where each e_i is of the form $\mathbf{d}(b_1 \dots b_r)$ for some constructor $\mathbf{d}$ with type $(Q_1, \dots, Q_r) \to D$, with each b_j being of form $f(g_j(x, \vec{y}), \vec{y})$ if Q_j is D, and of the form $g_j(x, \vec{y})$ otherwise.[¶] A function over the given data-system is *primitive corecursive* if it is obtained by repeated use of explicit definitions and the schema of corecurrence.

Example. Digit Streams form a simple data system of the kind mentioned above: CONS is the unique non-constant constructor, which we denote by an infixed colon. The remaining constructs are the 0-ary $\mathbf{0}$ and $\mathbf{1}$, and the data-predicates are the inductive (and finite) D (digits) and the coinductive S (streams))). The rules are

$$\frac{}{D(\mathbf{0})} \qquad \frac{}{D(\mathbf{1})} \qquad \frac{D(x) \quad S(y)}{S(x : y)}$$

The constructor CONS has the the two destructors $hd\colon S \to D$ and $tl\colon S \to S$.

Since there is a single non-consant constructor here, corecurrence can be formulated using the destructors, as the template:

$$\begin{aligned} hd(f(x, \vec{y})) &= g_0(x, \vec{y}) \\ tl(f(x, \vec{y})) &= f(g_1(x, \vec{y}), \vec{y} \end{aligned}$$

[¶]This definition is for a directed data-system. A further refinement is needed for the fully general case.

For example, we can define by corecurrence a function *even*:

$$hd(even(x)) = hd(x); \quad tl(even(x))even(tl(tl(x))).$$

The function *even* is fair, in the sense that it maps streams to streams. More precisely, in every model $\mathcal{S}$ of the data-system, expanded to interpret *even* while satisfying its equational definition, if $S(x)$ holds for x bound to an object a, then $S(even(x))$.

The fairness of *even* can be proved using the fact that $hd(even(x))$ and $tl(tl(x)))$ do not use $even(x)$. This is the guardedness condition described in [4]

We consider here only streams over a finite sets, or more generally coinductive data built from initial predicates that are finite; this excludes for example streams of natural numbers, which are studied in a category-theoretic setting for example in [16, 17].

5.2 Data-positive coinduction captures corecurrence

Consider intrinsic theories, as defined above. A formula is *data positive* if no data-predicate occurs in it in a negative position. (Recall that a position in a formula is negative if it is in the negative (premise) scope of an odd number of implications (where we consider each negation $\neg\varphi$ to be $\varphi \to \bot$). A formula is *data-free* if it has no occurrence of a data-predicate. In [11] we showed that a function over $\mathbb{N}$ is primitive recursive iff it is provably correct in the intrinsic theory for $\mathbb{N}$, with induction restricted to data-positive formulas.[‖]

Here we prove a similar characterization for the class of primitive corecursive functions. For simplicity, we refer to the intrinsic theory for the data-system $\mathcal{S}m$ of streams of booleans, based on minimal logic. We write $\mathbf{IT}^+$ for that theory, with coinduction restricted to data-positive formulas.

LEMMA **5** *If f is defined by corecurrence from functions provable in $\mathbf{IT}^+$ the f is provable in $\mathbf{IT}^+$.*

Proof. Suppose f defined by

$$hd(f(x)) = g_0(x) \quad tl(f(x)) = f(g_1(x))$$

Let (P_0, g_0) and (P_1, g_1) be programs (with no common function-identifiers) that are provable in $\mathbf{IT}^+$, with $\mathcal{D}_0$ a derivation of $D(g_0(x))$ from $S(x)$ and P_0, and $\mathcal{D}_1$ deriving $S(g_1(x))$ from $S(x)$ and P_1. Consider (P, f) where P is $P_0 \cup P_1$ augmented with the corecursive definition of f from g_0 and g_1. Then $S(f(x))$ is derived from $S(x)$ and P as follows. Let $\varphi[z]$ be $\exists y\, S(y) \wedge f(y) = z$. Note that φ is data-positive. Then $S(f(x))$ is derived from assumptions $S(x)$ and P by coinduction on φ, since we have the three premises of coinduction:

- From $S(x)$ we have $\varphi[f(x)]$,
- $\mathcal{D}_0$ establishes $\varphi[u] \to D(g_0(u))$.
- To prove $\varphi[u] \to \varphi[tl(u)]$, assume $\varphi[u]$, i.e. $S(v) \wedge f(v) = u$ for a fresh variable v. Then $S(g_1(v))$ by $\mathcal{D}_1$, and $tl(u) = tl(f(v)) = f(g_1(v))$ by So $\varphi[tl(u)]$ with y taken to be $g_1(v)$.

□

6 Corecurrence captures data-positive coinduction

6.1 Derivations as data

A known technique for showing that the provable functions of a given deductive formalism have a certain complexity is by showing that two proof-manipulation processes fall within that complexity class: the conversion of deductions of function-correctness into some normal form, and the extraction from such normal form the output of the functions considered.

The latter step is particularly transparent when data is represented directly in proof structure. This is particularly clear for intrinsic theories for *inductive* data, such as the natural numbers. For example, writing N for the data-predicate for $\mathbb{N}$, a normal proof of $N(\mathbf{t})$ for a closed term $\mathbf{t}$ consists of a sequence of formulas $N(\mathbf{t}_0) \dots N(\mathbf{t}_k)$ (where $\mathbf{f}_0$ is 0 and $\mathbf{t}_k$ is $\mathbf{t}$), where each $N(\mathbf{t}_{i+1})$ follows from $N(\mathbf{t}_i)$ either by the rule $N(x)/N(\mathbf{s}(x))$ or by an instance of the Equality Rule deriving with main premise $\mathbf{t}_i = \mathbf{t}_{i+1}$. Thus the value of $\mathbf{t}$ can be trivially extracted from the normal proof (and corresponds formally to the structure of the derivation via a suitable variant of the Curry-Howard morphism). A proof $\Pi[x]$ of $N(x) \to N(f(x))$ (from an equational program for f), with x a free variable, can then be used as an algorithm for computing f: for input k, we augment[**] $\Pi[\bar{k}]$ by the trivial proof of $N(\bar{k})$ to obtain a proof of $N(f(\bar{k})$; normalizing that proof we obtain a proof from which the value of $f(k)$ is extracted trivially.

[‖] This holds regardless to whether the logic is classical, constructive, or minimal. For classical logic the result holds also if induction is allowed for an analogous notion of data-negative formulas.

[**] Recall that $\bar{k}$ is the numeral $\mathbf{s}^{[k]}(\mathbf{0})$.

To apply this technique to coinductive data, we are naturally led to consider derivations whose structure represents infinite data, i.e. infinite derivations. Well-founded infinite derivations (such as ω-proofs of arithmetic or ω-logic) are unproblematic and well-known, but the derivations we need are inherently non-well-founded. Of course, non-well-founded derivations are generally inconsistent, since any formula φ can be derived by a non-well-founded natural deduction that alternates between deriving φ from $\varphi \wedge \top$ and vice versa. However, by insisting on simple structural conditions on non-well-founded derivations, we will be able to ensure their soundness (let alone consistency).

We consider as a separate rule a degenerate form of coinduction, deriving $S(\mathbf{t})$ (for a term $\mathbf{t}$) from $D(hd(\mathbf{t}))$ and $S(tl(bft))$, which we dub stream-reconstruction, or *Reconstruction* for short. Reconstruction is easily justified by coinduction for the (data-positive) formula $\varphi[z] \equiv D(hd(z)) \wedge S(tl(z))$. (Note that more general forms of coinduction derive $S(\mathbf{t})$ without actually evaluating $\mathbf{t}$, and thus have no bearing on the use of infinite branches to emulate data.) We then consider derivations in which every (infinite) branch eventually consist of reconstruction rules only.[††] We dub such derivations *admissible.* Admissible derivations are sound, because infinite sequences of reconstructions are obviously sound.

The collection of all derivations with finite and infinite branches, whether sound or not, is trivially definable as coinductive data, over which we can (and will) use corecurrence.

We refer to the usual notion of logical detours in natural deductions, and the corresponding reduction operations [14, 11]. as well as data-detour, consisting of data-introduction (coinduction) followed by data-elimination. Such detours are easily eliminated (by reductions dual to the ones used for induction [14]. For the case of streams of digits we have, for example,

$$\dfrac{\dfrac{\begin{array}{ccc}\varphi[z] & & \varphi[z]\\ \vdots & & \Pi[z]\\ \varphi[\mathbf{a}:\mathbf{t}] & D(hd(z) & \varphi[tl(z)]\end{array}}{S(\mathbf{a}:\mathbf{t})}}{S(\mathbf{t})}$$

reduces to

$$\dfrac{\begin{array}{ccc}\vdots & & \\ \varphi[\mathbf{a}:\mathbf{t}] & & \\ \Pi'[\mathbf{a}:\mathbf{t}] & \varphi[z] & \varphi[z]\\ \varphi[\mathbf{t}]] & \vdots & \vdots\\ & D(hd(z) & \varphi[tl(z)]\end{array}}{S(\mathbf{t})}$$

where $\Pi'[\mathbf{a}:\mathbf{t}]$ is $\Pi[\mathbf{a}:\mathbf{t}]$ suitably augmented with equality rules.

However, in contrast to derivations for *inductive* data, the absence of detours in intrinsic theories for coinductive data does not yet allow direct data-extraction. A formula $S(\mathbf{t})$ for closed $\mathbf{t}$ can be derived by an instance of coinduction, with the evaluation of $\mathbf{t}$ implied abstractly rather than concretely. This is a direct manifestation of the lazy evaluation of the output stream, and is similar to an unevaluated thunk in programs.

Just as the concrete value of the stream is obtained by actually evaluating its entries, a "concrete proof" of $S(\mathbf{t})$ is obtained from the coinduction deriving $S(\mathbf{t})$ by unfolding it into an infinite tower of instances of Reconstruction. To start,

$$\dfrac{\begin{array}{ccc} & \varphi[z] & \varphi[z]\\ \Pi_0 & \Pi_1[z] & \Pi_2[z]\\ \varphi[\mathbf{t}] & D(hd(z)) & \varphi[tl(z)]\end{array}}{S(\mathbf{t})}$$

reduces to

$$\dfrac{\begin{array}{cccc} & \Pi_0 & & \\ \Pi_0 & \varphi[\mathbf{t}] & \varphi[z] & \varphi[z]\\ \varphi[\mathbf{t}] & \Pi_2[\mathbf{t}] & \Pi_1[z] & \Pi_2[z]\\ \Pi_1[\mathbf{t}] & \varphi[tl(\mathbf{t})] & D(hd(z)) & \varphi[tl(z)]\\ D(hd(\mathbf{t})) & & S(tl(\mathbf{t}) & \end{array}}{S(\mathbf{t})} \qquad (1)$$

And repeating the process we unfold the given coinduction into an infinite derivation built from instances of Reconstruction. Note that if the given derivations Π_i are admissible (let alone finite), then the resulting infinite derivation will also be admissible.

We are thus interested in *strictly-normal* derivations, defined to be admissible derivations with no logical detours, no coinductions, and no data-detour (but possibly with instances of Reconstruction). A strictly-normal derivation of $S(x)$, with no data-positive assumptions, and proving $S(\mathbf{t})$ for some term $\mathbf{t}$, must end with an instance of Equality Elimination or Reconstruction. In either case, we obtain from the derivation the successive values of the digits in $\mathbf{t}$.

Recall that we wish to obtain from a derivation $\Pi[x]$ of $S(fx)$ from the assumption $S(x)$ (x a variable) a method of calculating, given an input stream σ, the output stream $f\sigma$ (determined by the equational program P for f). It remains to explain how to treat an input stream σ to be bound to x: since σ is arbitrary, it has no syntactic representation in the formalism. The conceptually easiest approach is to think of the binding of x to an arbitrary σ semantically. That is, we treat x as a constant identifier, and consider

[††]More general forms of derivations would remain sound, but they are not needed for our purpose.

the theory $\mathbf{IT}^+[\sigma]$, obtained by augmenting the theory $\mathbf{IT}^+$ with the set $Diag(\sigma)$ (the "diagram" of σ) consisting of all equations of the form $hd^{[n]}(x) = \sigma_n$, where $hd^{[n]}(z)$ stands for $hd(tl\cdots tl(z)\cdots)$ with n tl's, and where σ_n is the n'th digit in σ. To calculate $f\sigma$ we then refer to the derivation $\Pi[x]$, and normalize it while replacing derivations of formulas $D(hd^{[n]}(x))$ from assumption $S(x)$ by the Equation Elimination

$$\frac{hd^{[n]}(x) = \sigma_n \quad D(\sigma_n)}{D(hd^{[n]}(x))}$$

This yields a derivation $\Pi'[x]$ of $S(f(x))$, based on binding x to σ, from which we can read the value $f(\sigma)$.

Note that virtually the same algorithm can be construed syntactically (in the spirit of the main proof in [11]), as follows. Assuming given the diagram $Diag(\sigma)$, we can describe an admissible derivation Π_σ of $S(x)$, and therefore of the derivation $\Pi"[x]$ obtained from the given derivation $\Pi[x]$ of $S(f(x))$ from $S(x)$ by augmenting each $S(x)$ as assumption by its proof Π_σ. The normal form of $\Pi"[x]$ is then essentially the same as the normal form of $\Pi'[x]$ above.

6.2 Data-positive provability implies primitive corecurrence

THEOREM **6** (Normalization) *Let Π be a derivation of $\mathbf{IT}^+$, proving a formula φ from open assumptions Γ. There is a function ν_Π, defined by corecurrence on arbitrary derivations (see §6.1) that maps normal admissible derivations for the data-positive formulas in Γ to a strictly-normal admissible derivation Π^* of the formula φ from the data-free formulas in Γ.*

A proof outline is given in the appendix.

If the main inference of Π is Coinduction, with immediate subderivations Π_0, Π_1, Π_2, then $\nu_\Pi(\vec{\Xi})$ is obtained by defining corecursively, as in the reduction (1) above, the unfolding of the augmented derivation, using the strictly-normal derivations $\nu_{\Pi_i}(\vec{\Xi})$. Note that the resulting infinite derivation has no new detours, in contrast to the unfolding of inductive data (see [11]). □

It should be noted that Π^* can be specified as a tree of inference rules, without displaying the entire syntactic object at each node of the proof-tree. This is important in the proof of the following.

THEOREM **7** (Extraction) *Let Π be a strictly-normal admissible derivation of formula $S(f(x))$ from some equational program P and $S(x)$. Further assume that Π as a tree of (names of) inference-rules, is primitive corecursive (over streams of digits). Then function f computed by P is primitive corecursive (over streams of digits).*

A proof outline is given in the appendix.

Combining the two theorems above we have:

THEOREM **8** *Let (P, f) be an equational program, such that $S(x) \to S(fx)$ is provable in $\mathbf{IT}^+$ augmented with P. Then the function f computed by P is primitive corecursive.*

References

[1] Jon Barwise and Yanis Moschovakis. Global inductive definability. *Journal of Symbolic Logic*, 43:521–534, 1978.

[2] Ulrich Berger. From coinductive proofs to exact real arithmetic. In *CSL*, pages 132–146, 2009.

[3] Alberto Ciaffaglione and Pietro Di Gianantonio. A certified, corecursive implementation of exact real numbers. *Theor. Comput. Sci.*, 351(1):39–51, 2006.

[4] Thierry Coquand. Infinite objects in type theory. In *TYPES*, pages 62–78, 1993.

[5] Ronald Fagin. Generalized first order spectra and polynomial time recognizable sets. In R. Karp, editor, *Complexity of Computation*, pages 43–73. SIAM-AMS, 1974.

[6] Neil Immerman. Descriptive and computational complexity. In *FCT*, pages 244–245, 1989.

[7] N.G. Jones and A.L. Selman. Turing machines and the spectra of first-order formulas. *Journal of Symbolic Logic*, 39:139–150, 1974.

[8] Daniel Leivant. A foundational delineation of poly-time. *Information and Computation*, 110:391–420, 1994.

[9] Daniel Leivant. Intrinsic theories and computational complexity. In D. Leivant, editor, *Logic and Computational Complexity*, LNCS, pages 177–194, Berlin, 1995. Springer-Verlag.

[10] Daniel Leivant. Intrinsic reasoning about functional programs I: First order theories. *Annals of Pure and Applied Logic*, 114:117–153, 2002.

[11] Daniel Leivant. Intrinsic reasoning about functional programs II: unipolar induction and primitive-recursion. *Theor. Comput. Sci.*, 318(1-2):181–196, 2004.

[12] Yiannis N. Moschovakis. The formal language of recursion. *J. Symb. Log.*, 54(4):1216–1252, 1989.

[13] Charles Parsons. On a number-theoretic choice schema and its relation to induction. In A. Kino, J. Myhill, and R. Vesley, editors, *Intuitionism and Proof Theory*, pages 459–473. North-Holland, Amsterdam, 1970.

[14] D. Prawitz. *Natural Deduction*. Almqvist and Wiksell, Uppsala, 1965.

[15] Alfred Tarski. Some notions and methods on the borderline of algebra and metamathematics. In *Proceedings of the International Congress of Mathematicians I*, pages 705–720, Providence, RI, 1952. American Mathematical Society.

[16] Tarmo Uustalu and Varmo Vene. Primitive (co)recursion and course-of-value (co)iteration, categorically. *Informatica*, 10:5–26, 1999.

[17] Varmo Vene and Tarmo Uustalu. Functional programming with apomorphisms (corecursion). In *In 9th Nordic Workshop on Programming Theory*, 1998.

Appendix: Technical proofs.

Proof outline for Theorem 6

Let Π be the given normal derivation of $\mathbf{IT}^+$. We reason by discourse-level structural induction on Π.

If Π is an axiom, data-free assumption, or Equality Introduction, then ν_Π is Π itself.

If Π is a data-positive assumption, then ν_Π is the identity function.

Π extends its immediate subderivation Π' by $\vee$-Introduction, then ν_Π maps the input derivations Ξ_i to the derivation obtained by $\vee$-introduction from the derivation $\nu_{\Pi'}(\vec{\Xi})$. The definition of ν_Π is similar when the main inference of Π is any other logical introduction rule, or Equality Elimination.

If the main inference of Π is elimination, then it may itself engender a detour when derivations of data-positive assumptions are grafted on Π. If that happens, then the reduct of the resulting detour can be construed as the value of $\nu_{\Pi'}$ for suitable input, for some immediate subderivation Π' of Π, and we invoke $\nu_{\mathbf{P}'}$ (see [11] for detail). If no new detour appear at the root, then ν_Π is defined as for the case for the introduction-rules above.

If the main inference of Π is Coinduction, with immediate subderivations Π_0, Π_1, Π_2, then $\nu_\Pi(\vec{\Xi})$ is obtained by defining corecursively, as in the reduction (1) above, the unfolding of the augmented derivation, using the strictly-normal derivations $\nu_{\Pi_i}(\vec{\Xi})$. Note that the resulting infinite derivation has no new detours, in contrast to the unfolding of inductive data (see [11]). □

Proof outline for Theorem 7

Since Π is strictly normal, all formulas therein are equations or of the form $S(\mathbf{t})$ for some term $\mathbf{t}$, and all inferences are instances of Reconstruction and Equation rules. (Recall that strictly normal derivations use no coinduction, which we replace by streams of reconstructions). Moreover, we can assume w.l.o.g. (by contracting cascading equations) that subsequent instances of Reconstruction are separated by at most one Equation Elimination.

This Π consists of a trunk of instances of Reconstructions, with spikes consisting of finite subderivations for formulas $D(\cdot)$, all having the same structure. The assumption $S(x)$ is used in Π to infer $D(\mathbf{t})$ for terms $\mathbf{t} = hd^{[n]}(x)$; the value of each such term is given as σ_n. □