Higher-Kinded Data Types: Syntax and Semantics

Patricia Johann
Department of Computer Science
Appalachian State University
Email: johannp@appstate.edu

Andrew Polonsky
Department of Computer Science
Appalachian State University
Email: andrew.polonsky@gmail.com

***Abstract*—We present a grammar for a robust class of data types that includes algebraic data types (ADTs), (truly) nested types, generalized algebraic data types (GADTs), and their higher-kinded analogues. All of the data types our grammar defines, as well as their associated type constructors, are shown to have fully functorial initial algebra semantics in locally presentable categories. Since local presentability is a modest hypothesis, needed for such semantics for even the simplest ADTs, our semantic framework is actually quite conservative. Our results thus provide evidence that if a category supports fully functorial initial algebra semantics for standard ADTs, then it does so for advanced higher-kinded data types as well.**

To give our semantics we introduce a new type former called Lan that captures on the syntactic level the categorical notion of a left Kan extension. We show how left Kan extensions capture propagation of a data type's syntactic generators across the entire universe of types, via a certain completion procedure, so that the type constructor associated with a data type becomes a *bona fide* functor with a canonical action on morphisms. A by-product of our semantics is a precise measure of the semantic complexity of data types, given by the least cardinal λ for which the functor underlying a data type is λ-accessible. The proof of our main result allows this cardinal to be read off from a data type definition without much effort. It also gives a sufficient condition for a data type to have semantic complexity ω, thus characterizing those data types whose data elements are effectively enumerable.

I. Introduction

It is widely accepted that initial algebra semantics provides a principled, uniform, robust, and clarifying theory of algebraic data types (ADTs) such as lists, trees, the natural numbers, etc. In initial algebra semantics, every ADT is interpreted as the carrier of the initial algebra — i.e., the least fixed point — of an endofunctor on the semantic category $\mathcal{A}$ interpreting types. For example, the ADT `Nat = Zero | Succ Nat` is interpreted as the least fixed point μN for the endofunctor $NX = 1 + X$, and the ADT `List a = Nil | Cons a (List a)` is interpreted as μF for $FX = 1 + a \times X$, where the object a of $\mathcal{A}$ interprets `a`. Iterating the syntax of an ADT enumerates its data elements.

Because data types behave uniformly in their indices, they also induce type constructors. For example, the induced type constructor `List` uniformly constructs from each type `a` the new type `List a`. If types are interpreted as objects of $\mathcal{A}$, then type constructors are interpreted as endofunctors on $\mathcal{A}$. Programming-wise this means that every type constructor `D` induced by an ADT must have a functorial action that lifts any `f :: a → b` to a function `map f :: D a → D b` that can uniformly change the data in a structure but must preserve its shape. Importantly, the type `a` can itself be of the form `D c` for some type `c`, so that *nested* ADTs such as `D (D c)` not only make sense syntactically, but also induce type constructors whose `map` functions are obtained by suitably nesting calls to the `map` function for the ADT being nested. The `map` function for `List . List`, for example, is given by `map (map f) :: List (List c) → List (List d)`, where `f :: c → d` and `map` is as usual for lists.

ADTs are, of course, ubiquitous in programming, but many modern languages now support syntactic generalizations of ADTs such as nested types, generalized algebraic data types (GADTs), and other higher-kinded data types. A higher-kinded data type can be thought of intuitively as one whose data elements are obtained by the same iterative process as for ADTs, except that now the iteration is not just over *types* — which have kind $\star$ — but also over *type operators* that have kinds like $\star \to \star$ and $(\star \to \star) \to (\star \to \star) \to \star$ and themselves take as arguments either types or type operators of lower rank. GADTs in particular are often used to enforce sophisticated correctness properties of programs, including properties about the size or shape of data, the state of program components, and other data invariants. For example, the nested type[1]

```
data PSeq a where
  PSLeaf :: a → PSeq a
  PSNode :: PSeq (a, a) → PSeq a
```

comprises sequences whose lengths are powers of 2, while

```
data Seq a where
  SConst :: a → Seq a
  SPair  :: Seq a → Seq b → Seq (a, b)
  SSeq   :: (Int → Seq a) → Seq (Int → a)
```

enforces more general well-formedness conditions for sequences. Such constraints cannot be coded using ADTs alone.

Given the the success of initial algebra semantics for ADTs and the aforementioned computational intuition for higher-kinded data types, it is natural to try to extend initial algebra semantics for the former to the latter. Such semantics should not only interpret higher-kinded data types as carriers of initial algebras of endofunctors, but should moreover interpret the type constructors they induce as endofunctors, as described above. For some non-algebraic data types this is easily achieved. Nested types, e.g., are well-known to have interpretations as carriers of initial algebras of endofunctors on categories $[\mathcal{A}, \mathcal{A}]$ of appropriately cocontinuous endofunctors on categories $\mathcal{A}$ interpreting types [11]. For example, `PSeq` can

[1]Nested types comprise the special case of GADTs in which the return types of all data constructors are precisely the type being defined. Nested types with nested calls to themselves are sometimes called *truly nested types*.

978-1-7281-3608-0/19/$31.00

be interpreted as μP for the functor $P\,F\,X = X + F(X \times X) : [\mathcal{A},\mathcal{A}] \to [\mathcal{A},\mathcal{A}]$. One attempt to give a similar endofunctor initial algebra semantics for more general GADTs was made in [10]. However, it was argued there that such a semantics that moreover interprets the GADT itself as an endofunctor is not always possible because an arbitrary GADT need not have a functorial action on morphisms, i.e., a `map` function. For example, the clause of the function `map :: (a → b) → Seq a → Seq b` for `Seq` would be `map f (SPair x y) :: Seq b` for `f :: (c, d) → b`. Recalling that `map` functions for ADTs preserve the shapes of data structures while uniformly changing their data, and extending this intuition to GADTs, we expect the clause of `map` for `SPair` to have the form `map f (Pair x y) = Pair u v`. But `b` need not have a product structure and `f` need not be a pair of functions, so it was unclear how this could be achieved. Ultimately, [10] gives an initial algebra semantics for GADTs in terms of functors from the *discrete* category $|\mathcal{A}|$ to the category $\mathcal{A}$ interpreting types, rather than *endo*functors on $\mathcal{A}$.

There is, however, no *a priori* reason to expect GADTs to have discrete functor semantics when ADTs and nested types have *endo*functor semantics. Moreover, in discrete semantics, `map` functions become trivial, making it impossible to interpret GADTs that call themselves, such as the (truly) nested type

```
data Bush a where
  BNil  :: Bush a
  BCons :: a → Bush (Bush a) → Bush a
```

The clause of `map` for `BCons` should be `map f (BCons x t) = BCons (f x) (map (map f) t)`, but in discrete semantics `map f` is undefined if $\mathtt{f} \neq \mathtt{id}$. The upshot is that no discrete functor initial algebra semantics for GADTs can possibly be specialized to treat nested types, or even ADTs, involving nesting. Since GADTs are, as their name suggests, intended to generalize *all* ADTs, this indicates that something is amiss.

In this paper we remedy this situation by first giving a grammar for a very general class of higher-kinded data types that includes a wide variety of ADTs, (truly) nested types, and (nested) GADTs, and then showing that

Theorem. *Every higher-kinded data type defined by our grammar has an initial algebra interpretation as the least fixed point of an appropriately cocontinuous higher-kinded endofunctor over an appropriate category interpreting types.*

It follows that the type constructors induced by the higher-kinded data types defined by our grammar also have (appropriately cocontinuous) endofunctor interpretations. An important aspect of our semantics is that it naturally specializes to an endofunctor initial algebra semantics for GADTs that itself naturally specializes to the usual initial algebra semantics for ADTs and nested types. We thus show, at last, that GADTs really *are* generalizations of ordinary ADTs, and hence are worthy of their name. The precise form of the above theorem (Theorem 11 below) further guarantees that our semantics for higher-kinded types is *effectively computable* — i.e., that elements of these types can be exhaustively generated by an algorithm — in exactly the same settings in which the usual initial algebra semantics for ADTs are. This provides strong evidence that the semantics gives the "right" way to understand higher-kinded data types.

The initial algebra semantics we give in this paper provides exactly the same kind of principled, uniform, robust, and clarifying semantics for higher-kinded data types we already had for ADTs. Remarkably, all of our results are consequences of our taking seriously the idea that, in a good semantics, *all* data types — even GADTs and other higher-kinded ones — should be interpreted as least fixed points of actual *endo*functors.

The remainder of this paper is organized as follows. In Section II we describe our *methodological contributions*, namely our new notion of functorial completion for data types, and our use of left Kan extensions of *endo*functors to compute functorial completions in locally presentable categories. Next, we show how our methodology leads to the paper's *technical contributions*. In Section III we give our grammar for higher-kinded data types, and in Section IV we recall the categorical background needed in Section V to extend the usual endofunctor initial algebra semantics for ADTs to the GADTs and other higher-kinded data types the grammar defines. We also identify conditions under which our semantics is effectively computable, and show, via the examples in Section VI, that it corresponds precisely to the intuitive understanding of data types and how their data elements are enumerated.

II. Key Idea and Semantic Setting

The Key Idea: Functorial Completion We achieve our endofunctor initial algebra semantics for higher-kinded data types by moving from a primarily syntactic view of data types to a more fundamentally semantic one. Our key insight is that *data types are, in general, underspecified by their syntax*. This is a *fundamentally new observation* that recognizes for the first time that even an ordinary GADT declaration need not specify all of the data elements it defines. Instead it specifies only the "syntactic seeds" from which those elements can be generated. This is in sharp contrast with ADT and nested type declarations, whose syntax is exhaustively enumerative.

To see that a data type's data elements need not be fully enumerated by its syntax, note that if a data type is to itself be interpreted as an endofunctor, then everything in the closure of the collection of data elements enumerated by a data type's syntax under its `map` function must also be considered data elements of the data type. Now, we have already seen that even an innocuous-looking GADT like `Seq` need not support a `map` function mapping structures of a particular syntactic shape to structures of that same syntactic shape. However, the notion of "shape" on which this analysis relies is entirely syntactic. If we instead take a more semantic view and consider a structure, regardless of its syntactic form, to have a data constructor's "shape" if it is in that constructor's *functorial closure* — i.e., if it can be obtained from data built using that constructor by applications of the data type's `map` function — then `map` functions can still be seen as preserving structures' shapes. This new, inherently semantic view of data types shows that, claims in [10] notwithstanding, requiring *all* data types to have

*endo*functor initial algebra semantics can indeed be reconciled with the expectation that data types induce type constructors with change-the-data-but-preserve-the-shape `map` functions.

To compute data types' functorial closures we will use left Kan extensions [13]. For functors $K : C \to C'$ and $H : C \to D$, the *left Kan extension of H along K*, denoted $Lan_K H$, is the best approximation to a functor from C' to D through which H factors via K. Left Kan extensions are already present in the GADT semantics of [10]. There, they give a bijection between types of the form `forall a. h a → f (k a)` and those of the form `forall a. Lan k h a → f a`, where `Lan k h a` is defined by

```
data Lan k h a where
  LanC :: (k b → a) → h b → Lan k h a
```

However, being applied to discrete functors, they have trivial functorial actions, and thus cannot perform the functorial closure we require here. By contrast, the left Kan extensions in this paper are applied to non-discrete functors, so even for just ordinary GADTs they extend the restriction of a GADT's `map` function to structures of the same syntactic form to a `map` function *on all of the data elements of the GADT*. More generally, left Kan extensions can be seen as "realizing" the (entire, possibly higher-kinded) data type (implicitly) specified by (the seeds of) a data type's syntax. For example, if a programming language included a syntactic construct for left Kan extensions, such as `Lan` above, then, letting `k a b` = `(a, b)` and `h a b` = `(Seq a, Seq b)`, the type of the data constructor `SPair` could be rewritten as `SPair :: Lan k h c → Seq c`, indicating that `Seq`'s semantics involves its entire functorial closure.

For ADTs, nested types, and other data types whose syntax is wholly enumerating, there is no distinction between purely syntactic `map` functions and their functorial closures; the left Kan extensions are always along identity functors in these cases. This explains how left Kan extensions arise in initial algebra semantics for higher-kinded data types that specialize to the standard ones for ADTs without (explicitly) appearing there: they are actually present, though degenerate, but have remained hidden until exposed by our semantic analysis.

Our first step is thus to expose in syntax the heretofore hidden presence of (sometimes degenerate) left Kan extensions in the semantics of data types. The grammar in Section III defines not just the (higher-kinded) data types in which we are interested, but also the (higher-kinded) type constructors they induce. In addition to the usual constructs for data types, it includes a new one, called *Lan*, whose applications will be interpreted as left Kan extensions, and that makes explicit the functorial completion implicit in precompositions with functors in data constructor return types.

The Semantic Setting: Locally Presentable Categories To state and prove the precise form of our main result (Theorem 11), we work with λ-accessible functors on locally λ-presentable categories, for regular cardinals λ [4]. Locally presentable categories [6] are of fundamental importance in category theory. They include many familiar semantic categories, such as Set, models of (essentially) algebraic theories (e.g., the theories of groups, rings, lattices, etc.), ω-CPO, Grothendieck toposes, and the category of Banach spaces and contractions, which figures in models of linear logic. In locally presentable categories, the left Kan extensions that will interpret the *Lan* construct can be reformulated as colimits: in such categories, the "large" colimit that is a left Kan extension is always determined by a "small" set of support.

A λ-accessible functor on a locally λ-presentable category can be thought of intuitively as one whose initial algebra is guaranteed to be computable in at most λ steps. Our proof of Theorem 11 below guarantees that, in a locally λ-presentable category, if K and H are λ-accessible functors then $Lan_K H$ is always λ'-accessible for some $\lambda' \geq \lambda$. Importantly, λ' can be *strictly larger* than λ here, depending on which functors H and K are extended and extended along. This makes perfect sense intuitively: even for GADTs, the functorial expression K being precomposed with in the return type of a data constructor clearly affects how quickly computation of the initial algebra interpreting that GADT will converge. Fortunately, analysis of the proof of Theorem 11 reveals that no increase in cardinal is required to interpret functorial expressions other than those involving *Lan*. We can therefore ensure that all functorial expressions have interpretations as λ-accessible functors *for the very same* λ by restricting the classes of functorial expressions F_1 and F_2 appearing in expressions of the form $Lan_{F_1} F_2$.

Taking $\lambda = \omega$ in particular, we have that, in locally finitely (i.e., ω-)presentable categories, left Kan extensions interpreting $Lan_{F_1} F_2$ for appropriately restricted functorial expressions F_1 and F_2 can be effectively computed using their colimit reformulations. This allows us to identify a large class of GADTs and other higher-kinded data types as being *effectively enumerable*. For example, the functors interpreting `Nat`, `List`, `PSeq`, and `Bush` are all ω-accessible, so their corresponding data types are all effectively enumerable in locally finitely presentable categories. Examples of effectively enumerable truly higher-kinded data types appear in Section VI. By contrast, the functor interpreting `Seq` is not ω-accessible, but there is a $\lambda > \omega$ for which it is λ-accessible. Thus, while data types constructed using `Seq` are not effectively enumerable in the manner they are often (incorrectly) assumed to be, in locally λ-presentable categories they *can* be completely enumerated in λ steps. Critically, Theorem 9 shows that, in such categories, the left Kan extension demanded by the rewritten type of `SPair` above can be computed as the colimit

$$\varinjlim_{(a,b)\in\mathcal{A}\times\mathcal{A}, f:(a,b)\to e} h\,(a,b) \to \texttt{Seq}\,e$$

for $h\,(a,b) = \texttt{Seq}\,a \times \texttt{Seq}\,b$. It is no coincidence that the interpretation f of the function `f :: (a, b) → e` whose absence motivated [10]'s discretization of functors appears in this colimit.

Almost incredibly, local finite presentability, which supports effective enumerability of the higher-kinded data types defined by our grammar, is *the minimal reasonable hypothesis* guaranteeing effective enumerability of ADTs and nested types. More generally, locally presentable categories and accessible functors on them provide a means of measuring how semantically

complex data types are: we can define the *semantic complexity* of a data type to be the smallest regular cardinal λ such that it can be interpreted as the carrier of the initial algebra of a λ-accessible endofunctor on a locally λ-presentable category over the category interpreting types.

III. Syntax of Higher-kinded Data Types

We present our grammar of higher-kinded data types in two stages. In Section III-A we introduce functorial expressions, of which ADTs, nested types, and GADTs are special cases. In Section III-B we extend these to higher kinds. In order to be as general as possible, the only assumption we make about the host language is that it includes the inductive types generated by our grammar. That is, we remain completely agnostic regarding other features the host language may support.

A. The grammar of functorial expressions

For each $k \geq 0$, we assume a countable set $\mathbb{T}^k$ of *functor variables of arity* k, disjoint for distinct k. We use lower case Greek letters for functor variables, and write φ^k to indicate that $\varphi \in \mathbb{T}^k$. We call $\mathbb{T}^0$ the set of *type variables*. When convenient we may write α, β, etc., rather than α^0, β^0, etc., for elements of $\mathbb{T}^0$. The set of all functor variables is $\mathbb{T} = \bigcup_{k\geq 0} \mathbb{T}^k$. We write $\boldsymbol{\varphi}$ for a set $\{\varphi_1, ..., \varphi_n\}$ of functor variables when the cardinality n of the set is unimportant. We further write $\mathcal{P}, \boldsymbol{\varphi}$ for $\mathcal{P} \cup \boldsymbol{\varphi}$. We omit the boldface for a singleton set φ.

Let $\mathcal{P} \subseteq \mathbb{T}$. The grammar $\mathcal{F}$ defining the set $\mathcal{F}^{\mathcal{P}}(k)$ of *functorial expressions of arity* k is given by

$$\begin{aligned}\mathcal{F}^{\mathcal{P}}(k) \;::=\; \lambda\alpha_1...\alpha_k.\quad & \alpha_i \mid \mathbb{0} \mid \mathbb{1} \mid \mathcal{P}\,\overline{\mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0)} \\ & \mid \mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0) + \mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0) \\ & \mid \mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0) \times \mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0) \\ & \mid \left(\mu\varphi^n.\mathcal{F}^{\mathcal{P},\boldsymbol{\alpha},\varphi}(n)\right)\overline{\mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0)} \\ & \mid \left(\mathit{Lan}^{\boldsymbol{\beta}}_{\mathcal{F}^{\boldsymbol{\beta}}(0)}\mathcal{F}^{\mathcal{P},\boldsymbol{\alpha},\boldsymbol{\beta}}(0)\right)\overline{\mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0)}\end{aligned}$$

We call functorial expressions of arity 0 *types*.

If $F \in \mathcal{F}^{\mathcal{P}}(k)$ then $\mathsf{FV}(F) \subseteq \mathcal{P}$. The intuitive meaning of the grammar $\mathcal{F}$, which will be made precise in Section V, is that each $F \in \mathcal{F}^{\mathcal{P}}(k)$ defines a λ-*cocontinuous* functor $[\![F]\!] : \mathcal{A}^{\mathcal{P}} \times \mathcal{A}^k \to \mathcal{A}$, where $\mathcal{A}$ is the corresponding locally λ-presentable category interpreting types, and $\mathcal{A}^{\mathcal{P}}$ is a locally λ-presentable category interpreting the free variables in $\mathcal{P}$. We think of F as a k-ary, $\mathcal{P}$-indexed functor that is λ-cocontinuous in *both* its k arguments and its context $\mathcal{P}$.

A number of observations are in order:

• The grammar $\mathcal{F}$ does not include any function types or polymorphic types. This is not meant as a restriction on the host language; rather, it reflects the focus of the present paper, namely, *(higher-kinded) data types* and their semantics. Other types the host language might support will naturally impose their own semantic hypotheses. To support function types, e.g., $\mathcal{A}$ will also need to be a cartesian closed category (ccc).[2]

[2]Cartesian closure interacts well with local presentability; several of the examples from the introduction are, in fact, cccs. Since locally presentable cccs are closed under the formation of categories of λ-cocontinuous functors, the extension to higher kinds in Section III-B is unproblematic.

• Application of an expression E is allowed only when E is a functor variable or a functorial expression whose head is either μ or Lan. Moreover, if E has arity n then E must be applied to exactly n arguments. That is, functorial expressions are always in η-*long normal form*. This avoids having to consider β-conversion on the level of type constructors. Similarly, the fact that the standard type formers are all defined pointwise avoids having to relate functorial expressions at different kinds. As discussed at the end of this subsection, there are other ways to remedy these problems for the grammar $\mathcal{F}$. But since these are not available for the higher-kinded extension $\mathcal{H}$ of $\mathcal{F}$ given in Section III-B, we choose the above presentation of $\mathcal{F}$.

• An overbar indicates a sequence of subexpressions whose length matches the arity of the functorial expression applied to it. For example, the clause $\mathcal{P}\,\overline{\mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0)}$ for functor variables means that, for all $k, n \geq 0$ and each $\varphi^n \in \mathcal{P}$, if $A_i \in \mathcal{F}^{\mathcal{P},\boldsymbol{\alpha}}(0)$ for $i = 1, ..., n$, then $\lambda\alpha_1, ..., \alpha_k.\, \varphi^n A_1 \cdots A_n \in \mathcal{F}^{\mathcal{P}}(k)$. A similar remark applies to the clause for μ, where the length of the sequence matches the arity n of the recursion variable φ^n bound by μ. Each subexpression $\mathit{Lan}_{\bar{K}} H$ also has an arity parameter, which is left implicit. It imposes the constraint that the number of expressions in the sequence $\bar{K}$ must match the number of arguments to which $\mathit{Lan}_{\bar{K}} H$ is applied.

• In a subexpression, $\mu\varphi^n.H$, the μ operator binds all occurrences of the variable φ^n in its body H. Similarly, the Lan operator binds all occurrences of the variables in $\boldsymbol{\beta}$ both in the subscript expressions and in the body of the Lan operator. In addition, the former can depend only on the variables in $\boldsymbol{\beta}$. As a consequence, nesting is not allowed in return types of data constructors. This will ensure that Lan-expressions can be interpreted by λ-accessible functors.

We introduce the abbreviations $\mathbb{2} := \mathbb{1} + \mathbb{1}$, $\mathbb{0}^k := \lambda\alpha_1...\alpha_k.\mathbb{0}$, $\mathbb{1}^k := \lambda\alpha_1...\alpha_k.\mathbb{1}$, $F +^k G := \lambda\alpha_1...\alpha_k.F\boldsymbol{\alpha} + G\boldsymbol{\alpha}$, and $F \times^k G := \lambda\alpha_1...\alpha_k.F\boldsymbol{\alpha} \times G\boldsymbol{\alpha}$. As usual, we may omit k when $k = 0$. ADTs are then expressible in $\mathcal{F}$ in the usual way. For example, the type constructors for `Nat` and `List` can be represented as

$$\begin{aligned}\texttt{Nat} &:= \mu\alpha.\mathbb{1} + \alpha \in \mathcal{F}^{\varnothing}(0) \\ \texttt{List} &:= \lambda\alpha.\mu\beta.\mathbb{1} + \alpha \times \beta \in \mathcal{F}^{\varnothing}(1)\end{aligned}$$

$\mathcal{F}$ is also sufficiently rich to express a large class of GADTs. In particular, the type constructors for `PSeq` and `Bush` can be represented by the following expressions in $\mathcal{F}^{\varnothing}(1)$:

$$\begin{aligned}\texttt{PSeq} &:= \lambda\alpha.\left(\mu\varphi^1.\lambda\beta.\beta + \varphi(\beta \times \beta)\right)\alpha \\ \texttt{Bush} &:= \lambda\alpha.\left(\mu\varphi^1.\lambda\beta.\mathbb{1} + \beta \times \varphi(\varphi\beta)\right)\alpha\end{aligned}$$

At first glance, it appears that `Seq` is *not* expressible in $\mathcal{F}$ because this grammar contains no function type constructor. However, function types with *constant* domains *can* be expressed in $\mathcal{F}$. In fact, if C is any closed type, then $C \to D$ can be represented as $(\mathit{Lan}^{\beta}_{C}\mathbf{1})D$. Thus, $\texttt{Seq} \in \mathcal{F}^{\varnothing}(1)$ can be represented by

$$\begin{aligned}\texttt{Seq} := \lambda\alpha.\Big(\mu\varphi^1.\lambda\beta.\beta + \Big(\mathit{Lan}^{\gamma_1,\gamma_2}_{\gamma_1\times\gamma_2}(\varphi\gamma_1 \times \varphi\gamma_2)\Big)\beta \\ + \Big(\mathit{Lan}^{\gamma}_{\texttt{Nat}\to\gamma}(\texttt{Nat} \to \varphi\gamma)\Big)\beta\Big)\alpha\end{aligned}$$

While excluding general arrow types and $\forall$-types may seem unnecessarily restrictive, there is good reason to do so. The expression $F(\alpha) = (\alpha \to 2) \to 2$, e.g., does not give rise to a λ-cocontinuous functor, for *any* λ, under the standard interpretation of type constructors in Set. And although GADTs with constructors involving arrow types frequently appear in the literature, these can also be problematic. We have already noted, e.g., that `Seq` does not give rise to an ω-cocontinuous functor, as would be required for it to have the computational behavior typically ascribed to GADTs in practice. As we will see in Section VI, the process of iterating the elements of `Seq` directly from its definition need not converge after ω steps. Even naively, this can be seen by noting that if we compute a sequence s_k of terms of type `Seq` each of which is added only in the k^{th} iteration, then we can diagonalize out of `Seq` using the `SSeq` constructor. Thus, even very familiar-looking GADTs involving arrow types with constant domains can fail to have the computational meanings they are intuitively assigned.

While mixing fixed points with function types very rapidly leads to pitfalls, nothing prevents the host language from supporting function types (or $\forall$-types) if the semantic category interpreting types supports them. In particular, since `List` will be interpreted by a functor on this category, it can be applied to *any* object X, including an object X that happens to interpret a function type or a polymorphic type. The restrictions on the grammar $\mathcal{F}$ therefore merely reflect a *separation of concerns*.

Finally, as promised, we consider the question of why the arity k is made an explicit parameter in the grammar $\mathcal{F}$, rather than part of the context $\mathcal{P}$. For $\mathcal{F}$, we could indeed require all arguments to be given in $\mathcal{P}$ and implicitly assume all expressions to have outputs of arity 0. Intuitively, each $E \in \mathcal{F}^{\mathcal{P}}$ would be interpreted by a λ-cocontinuous functor $[\![E]\!] : \mathcal{A}^{\mathcal{P}} \to \mathcal{A}$ and a k-ary functorial expression $G = \lambda\alpha_1 ... \alpha_k . E \in \mathcal{F}^{\varnothing}(k)$ would be represented as $E \in \mathcal{F}^{\{\alpha_1,\ldots,\alpha_k\}}$. Then, with k fixed at 0, $\mathcal{F}$ would become the new grammar $\mathcal{F}_0$ given by

$$\begin{aligned}\mathcal{F}_0^{\mathcal{P}} \;::=\; & \mathbb{0} \mid \mathbb{1} \mid \mathcal{P}\overline{\mathcal{F}_0^{\mathcal{P}}} \mid \mathcal{F}_0^{\mathcal{P}} + \mathcal{F}_0^{\mathcal{P}} \mid \mathcal{F}_0^{\mathcal{P}} \times \mathcal{F}_0^{\mathcal{P}} \\ & \mid \left(\mu\varphi^{k}.\lambda\alpha_1..\alpha_k.\mathcal{F}_0^{\mathcal{P},\boldsymbol{\alpha},\varphi}\right)\overline{\mathcal{F}_0^{\mathcal{P}}} \\ & \mid \left(\mathit{Lan}^{\boldsymbol{\beta}}_{\mathcal{F}_0^{\boldsymbol{\beta}}} \mathcal{F}_0^{\mathcal{P},\boldsymbol{\beta}}\right)\overline{\mathcal{F}_0^{\mathcal{P}}}\end{aligned}$$

But, as we see next, making the arity an explicit feature of the syntax, as we have done, eases the transition to higher kinds.

B. The grammar of general type constructors

The collection $\mathcal{K}$ of kinds comprises all simple types over the kind $\star$ of types. For each $\kappa \in \mathcal{K}$, we assume a countable set $\mathbb{T}^{\kappa}$ of *type constructor variables of arity* κ, disjoint for distinct κ. We use lower case Greek letters for type constructor variables, and write φ^{κ} to indicate that $\varphi \in \mathbb{T}^{\kappa}$. Each $\kappa \in \mathcal{K}$ has an arity $|\kappa| \geq 0$ and can be uniquely written as $\kappa = \kappa_1 \to \cdots \to \kappa_{|\kappa|} \to \star$. Type variables are elements of $\mathbb{T}^{\star}$, being "type constructors of arity $\star$"; we continue to denote them by α, β, etc. Similarly, every functor variable of arity k is in $\mathbb{T}^{\underline{k}}$, where $\underline{0} = \star$ and $\underline{k+1} = \star \to \underline{k}$. The set of all type constructor variables $\bigcup_{\kappa \in \mathcal{K}} \mathbb{T}^{\kappa}$ extends the set of functor variables from Section III-A. Hereafter, we let $\mathbb{T}$ denote this larger set.

The grammar $\mathcal{H}$ below implicitly uses the above unique decomposition of kinds in applications to multiple arguments. For example, assuming conventions for sets of type constructor variables analogous to those for functor variables in Section III-A, the clause $\mathcal{P}\overline{\mathcal{H}^{\mathcal{P}}(i)}$ means that, for all $\kappa, \nu \in \mathcal{K}$ and each $\varphi^{\nu} \in \mathcal{P}$, if $F_i \in \mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(\nu_i)$ for $i = 1, \ldots, |\nu|$ then $\lambda\varphi_1 ... \varphi_{|\kappa|} . \varphi^{\nu} F_1 \cdots F_{|\nu|} \in \mathcal{H}^{\mathcal{P}}(\kappa)$. Otherwise, the grammar $\mathcal{H}$ follows the familiar pattern:

$$\begin{aligned}\mathcal{H}^{\mathcal{P}}(\kappa) \;::=\; \lambda\varphi_1 ... \varphi_{|\kappa|}. \quad & \mathbb{0} \mid \mathbb{1} \mid (\mathcal{P} \cup \boldsymbol{\varphi})\overline{\mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(i)} \\ & \mid \mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(\star) + \mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(\star) \\ & \mid \mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(\star) \times \mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(\star) \\ & \mid \left(\mu\psi^{\nu}.\mathcal{H}^{\mathcal{P},\boldsymbol{\varphi},\psi}(\nu)\right)\overline{\mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(i)} \\ & \mid \left(\mathit{Lan}^{\boldsymbol{\psi}}_{\mathcal{H}^{\boldsymbol{\psi}}(\star)} \mathcal{H}^{\mathcal{P},\boldsymbol{\varphi},\boldsymbol{\psi}}(\star)\right)\overline{\mathcal{H}^{\mathcal{P},\boldsymbol{\varphi}}(\star)}\end{aligned}$$

The grammar $\mathcal{H}$ subsumes $\mathcal{F}$ with $\mathcal{F}^{\mathcal{P}}(k) = \mathcal{H}^{\mathcal{P}}(\underline{k})$. Observations analogous to those in Section III-A apply to $\mathcal{H}$. We define $\mathbb{0}^{\kappa}$, $\mathbb{1}^{\kappa}$, $+^{\kappa}$ and $\times^{\kappa}$ pointwise, as for functorial expressions.

IV. The Categorical Setup

As noted above, we intend to interpret each expression in $\mathcal{F}^{\mathcal{P}}(k)$ as a λ-cocontinuous functor of type $\mathcal{A}^{\mathcal{P}} \times \mathcal{A}^{k} \to \mathcal{A}$ for a locally λ-presentable category $\mathcal{A}$ interpreting types, and similarly for $\mathcal{H}$. To do so, we will need to know that each expression-forming construct can be interpreted as an operation on functors that preserves λ-cocontinuity. Since least fixed points — represented by μ-expressions in our grammar(s) — are the main construct in most data type definitions, we first focus on semantic settings that support initial algebras. Locally λ-presentable categories provide a very general such setting [1], so we require our semantic categories to be at least locally λ-presentable.

To interpret the other expressions in our grammar(s), we will also need to know that our semantic categories have the coproducts, products, and left Kan extension operations that will interpret the expression formers $+$, $\times$, and Lan, respectively. *A priori* we expect requiring the existence of each of these operations to impose additional constraints on the semantic categories, so that these categories will ultimately be some appropriate subcollection of locally λ-presentable ones. Surprisingly, however, the local λ-presentability imposed by μ-expressions is all that is needed to support *all* of the operations we need to interpret *all* of the expressions in the grammar(s). Indeed, as we show in this section, local λ-presentability guarantees not only that all the operations needed to interpret our expressions exist, but also that these operations, as well as functor application (which is used to interpret applications of type constructor variables), all preserve λ-cocontinuity.

Because they are typically defined in terms of least fixed points, local λ-pesentability is already the minimal reasonable hypothesis guaranteeing the standard initial algebra semantics for ADTs. What this section ultimately shows, then, is that the assumption of local λ-presentability, which we will presently adopt as our principal hypothesis for interpreting higher-kinded data types, is no stronger than what is already assumed

for ADTs. It is quite amazing that the same class of categories in which ADTs have endofunctor initial algebra semantics provides a natural setting for extending that semantics to higher kinds.

A. Preliminaries

We recall the definition and basic features of locally presentable categories, and verify the properties needed to interpret our syntax. Below, let λ be a regular cardinal.

A category is *small* if its collection of morphisms is a set. It is *locally small* if, for any two objects A and B, the collection of morphisms from A to B is a set. A *small (co)limit* in a category $\mathcal{C}$ is a (co)limit of a diagram $F : \mathcal{A} \to \mathcal{C}$, where $\mathcal{A}$ is a small category; it is *λ-small* if the cardinality of the set of arrows in $\mathcal{A}$ has cardinality less than λ. A category $\mathcal{C}$ is *(co)complete* if it has all small (co)limits.

A poset $\mathcal{D} = (D, \leq)$ is *λ-directed* if every subset of D of cardinality less than λ has an upper bound. When $\mathcal{D}$ is considered as a category, we write $d \in \mathcal{D}$ to indicate that d is an object of $\mathcal{D}$ (i.e., $d \in D$). For $d \in \mathcal{D}$, $d{\uparrow}$ is the poset $\{d' \in D \mid d \leq d'\}$. A *$\lambda$-directed colimit* in $\mathcal{C}$ is a colimit of a diagram $F : \mathcal{D} \to \mathcal{C}$, where $\mathcal{D}$ is a λ-directed poset. A category $\mathcal{C}$ is *λ-cocomplete* if it has all λ-directed colimits. A cocomplete category is one that has all colimits. If $\mathcal{C}$ is cocomplete, then it is λ-cocomplete for all λ.

If $\mathcal{A}$ and $\mathcal{C}$ are (λ-)cocomplete, then the functor $F : \mathcal{A} \to \mathcal{C}$ is *(λ-)cocontinuous* if it preserves (λ-directed) colimits. We write $[\mathcal{A}, \mathcal{C}]_{(\lambda)}$ for the category of (λ-)cocontinuous functors from $\mathcal{A}$ to $\mathcal{C}$, and $\mathcal{C}^{\mathcal{A}}$ for the category of *all* functors from $\mathcal{A}$ to $\mathcal{C}$. Since (co)limits of functors are computed pointwise, $\mathcal{C}^{\mathcal{A}}$ has all limits and colimits that $\mathcal{C}$ has, and (co)limits of (co)continuous functors are again (co)continous. It follows that $[\mathcal{A}, \mathcal{C}]_{(\lambda)}$ is (λ-)cocomplete whenever $\mathcal{C}$ is.

If $\mathcal{A}$ is locally small, then an object A of $\mathcal{A}$ is *λ-presentable* if the functor $\mathsf{Hom}_{\mathcal{A}}(A, -) : \mathcal{A} \to \mathsf{Set}$ preserves λ-directed colimits, i.e., if for every λ-directed poset $\mathcal{D}$ and every functor $F : \mathcal{D} \to \mathcal{C}$, there is a canonical isomorphism $\varinjlim_{d \in \mathcal{D}} \mathsf{Hom}_{\mathcal{A}}(A, Fd) \simeq \mathsf{Hom}_{\mathcal{A}}(A, \varinjlim_{d \in \mathcal{D}} Fd)$. A category $\mathcal{A}$ is *λ-accessible* if it is λ-cocomplete and has a set $\mathcal{A}_0$ of λ-presentable objects such that every object is a λ-directed colimit of objects in $\mathcal{A}_0$; it is *locally λ-presentable* if it is λ-accessible and cocomplete. Of the semantic categories discussed in Section II, Set and models of algebraic theories are locally finitely presentable, ω-CPO and Banach spaces are locally $\aleph_1$-presentable, and Grothendieck toposes are locally λ-presentable with λ depending on the site inducing the Grothendieck topology. Finally, a functor $F : \mathcal{A} \to \mathcal{C}$ is *λ-accessible* if $\mathcal{A}$ and $\mathcal{C}$ are λ-accessible and $F \in [\mathcal{A}, \mathcal{C}]_\lambda$. If $\lambda \leq \mu$ then $\mathcal{A}$ being locally λ-presentable implies $\mathcal{A}$ is locally μ-presentable (see the remark following Theorem 1.20 in [4]), but $\mathcal{A}$ being λ-accessible does not imply that $\mathcal{A}$ is μ-accessible.

If $\mathcal{A}_0$ is an essentially small category we write $\mathcal{A} \simeq \mathsf{Ind}^\lambda(\mathcal{A}_0)$ if $\mathcal{A}$ is a *free cocompletion of $\mathcal{A}_0$ with respect to λ-directed colimits*, i.e., if $\mathcal{A}$ is a λ-cocomplete category and every functor $F : \mathcal{A}_0 \to \mathcal{C}$ with $\mathcal{C}$ λ-cocomplete has a unique (up to natural isomorphism) λ-cocontinuous extension $F^* : \mathcal{A} \to \mathcal{C}$. The next two results are standard (see, e.g., [4]):

Proposition 1. (i) *If $\mathcal{A}$ is λ-accessible then $\mathcal{A} \simeq \mathsf{Ind}^\lambda(\mathcal{A}_0)$ for the essentially small category $\mathcal{A}_0$ of $\mathcal{A}$'s λ-presentable objects.* (ii) *If $\mathcal{A}$ is λ-accessible and $\mathcal{C}$ is λ-cocomplete, then the category $[\mathcal{A}, \mathcal{C}]_\lambda$ is naturally equivalent to the category $\mathcal{C}^{\mathcal{A}_0}$.* (iii) *If $\mathcal{C}$ is locally λ-presentable and $\mathcal{A}_0$ is essentially small, then $\mathcal{C}^{\mathcal{A}_0}$ is locally λ-presentable.*

Also, and crucially, in locally λ-presentable categories, λ-small limits commute with λ-directed colimits [4, Prop.1.59].

To define our semantics we will need to know that the application functor $(F, A) \in [\mathcal{A}, \mathcal{B}]_\lambda \times \mathcal{A} \longmapsto F(A)$ — i.e., the restriction of the evaluation morphism $ev_{\mathcal{A},\mathcal{B}}$ arising from the ccc structure on Cat to λ-accessible F — is λ-accessible. While this fact is used quite frequently, we could not find it recorded anywhere, so we prove it in Lemma 2 below.

To do so, we first recall that if $\mathcal{A}$ is a category, then the *arrow category* $\mathcal{A}^{\to}$ has as objects triples (A, B, f), where A and B are objects of $\mathcal{A}$ and $f \in \mathsf{Hom}_{\mathcal{A}}(A, B)$. The morphisms from (A, B, f) to (A', B', f') in $\mathcal{A}^{\to}$ are pairs $(\alpha, \beta) \in \mathsf{Hom}_{\mathcal{A}}(A, A') \times \mathsf{Hom}_{\mathcal{A}}(B, B')$ such that $\beta \circ f = f' \circ \alpha$. If $D = \{(A_d, B_d, f_d) \mid d \in \mathcal{D}\}$ is a λ-directed diagram in $\mathcal{A}^{\to}$, then the colimit in $\mathcal{A}^{\to}$ of D is (A, B, f), where $A = \varinjlim_{d \in \mathcal{D}} A_d$, $B = \varinjlim_{d \in \mathcal{D}} B_d$ (with colimit maps $b_d : B_d \to B$), and f is the unique map factoring the (A_d)-cocone $(B, \{b_d \circ f_d \mid d \in \mathcal{D}\})$ through the colimit $\varinjlim_{d \in \mathcal{D}} A_d = A$.

Lemma 2. *Let $\mathcal{A}$ and $\mathcal{C}$ be categories with $\mathcal{C}$ λ-cocomplete, and $F = \varinjlim_{d \in \mathcal{D}} F_d$ in $[\mathcal{A}, \mathcal{C}]_\lambda$. Also, let $X = \varinjlim_{d \in \mathcal{D}} X_d$ in $\mathcal{A}$, $f_d : X_d \to Y_d$ for each $d \in \mathcal{D}$, and $f = \varinjlim_{d \in \mathcal{D}} f_d$ in $\mathcal{A}^{\to}$. Then*

(i) $FX = \varinjlim_{d \in \mathcal{D}} F_d X_d$

(ii) $Ff = \varinjlim_{d \in \mathcal{D}} F_d f_d$.

Proof. (i) We compute

$$\begin{aligned} FX &= (\varinjlim_{d \in \mathcal{D}} F_d)X \\ &= \varinjlim_{d \in \mathcal{D}} (F_d X) \\ &= \varinjlim_{d \in \mathcal{D}} (F_d\, (\varinjlim_{d' \in \mathcal{D}} X_{d'})) \\ &= \varinjlim_{d \in \mathcal{D}} \varinjlim_{d' \in \mathcal{D}} F_d X_{d'} \\ &= \varinjlim_{d \in \mathcal{D}} F_d X_d \end{aligned}$$

Here, the penultimate equality holds because each F_d is λ-cocontinuous. The last equality holds because the diagonal set $\{(d, d) \mid d \in \mathcal{D}\}$ is cofinal in $\mathcal{D} \times \mathcal{D}$. Indeed, given $(d_1, d_2) \in \mathcal{D} \times \mathcal{D}$, let $d \geq d_1, d_2$ (by directedness). Then $(d_1, d_2) \leq (d, d)$.

Tracking the colimit calculation above gives that structure maps for $FX = \varinjlim_{d \in \mathcal{D}} F_d X_d$ are $\varphi_d X \circ F_d x_d : F_d X_d \to FX$, where the maps $\varphi_d : F_d \to F$ and $x_d : X_d \to X$ are the colimit structure maps for $F = \varinjlim_{d \in \mathcal{D}} F_d$ and $X = \varinjlim_{d \in \mathcal{D}} X_d$, respectively. This will be used in the next part of the proof.

(ii) We first prove that, for each $d_0 \in \mathcal{D}$, $F_{d_0} f = \varinjlim_{d \in \mathcal{D}} F_{d_0} f_d$. Let $Y = \varinjlim_{d \in \mathcal{D}} Y_d$ and $g_{d_0} = \varinjlim_{d \in \mathcal{D}} F_{d_0} f_d$. By the computation of colimits in arrow categories, if $x_d : X_d \to X$ and $y_d : Y_d \to Y$ are the evident colimit structure maps, then g_{d_0} is the unique

map from $F_{d_0}X = F_{d_0}(\varinjlim_{d\in\mathcal{D}} X_d) = \varinjlim_{d\in\mathcal{D}} F_{d_0}X_d$ to $F_{d_0}Y$ making the following diagram commute for each $d \in \mathcal{D}$:

$$\begin{array}{ccc} F_{d_0}X_d & \xrightarrow{F_{d_0}x_d} & F_{d_0}X \\ {\scriptstyle F_{d_0}f_d}\downarrow & & \downarrow{\scriptstyle g_{d_0}} \\ F_{d_0}Y_d & \xrightarrow{F_{d_0}y_d} & F_{d_0}Y \end{array} \tag{1}$$

Now, replacing g_{d_0} in the above square by $F_{d_0}f$ results in the F_{d_0}-image of the diagram defining f as $\varinjlim_{d\in\mathcal{D}} f_d$, so $F_{d_0}f$ also makes the above diagram commute for each i. Uniqueness of g_{d_0} gives $g_{d_0} = F_{d_0}f$, and thus indeed $F_{d_0}f = \varinjlim_{d\in\mathcal{D}} F_{d_0}f_d$.

Recalling how colimits of functors are defined, we furthermore have that, for each $d \in \mathcal{D}$, $Ff = \varinjlim_{d\in\mathcal{D}} F_d f = \varinjlim_{d\in\mathcal{D}} g_d$. That is, if the maps $\varphi_d : F_d \to F$ are the evident colimit structure maps, then Ff is the unique map from $FX \to FY$ making the following diagram commute for each $d \in \mathcal{D}$:

$$\begin{array}{ccc} F_dX & \xrightarrow{\varphi_d X} & FX \\ {\scriptstyle F_d f}\downarrow & & \downarrow{\scriptstyle Ff} \\ F_dY & \xrightarrow{\varphi_d Y} & FY \end{array} \tag{2}$$

The proof will be finished if we can show that Ff is also the unique map g making the following diagram commute for each d, where the horizontal arrows come from the structure map calculation at the end of the proof of part (i):

$$\begin{array}{ccc} F_dX_d & \xrightarrow{\varphi_d X \circ F_d x_d} & FX \\ {\scriptstyle F_d f_d}\downarrow & & \downarrow{\scriptstyle g} \\ F_dY_d & \xrightarrow{\varphi_d Y \circ F_d y_d} & FY \end{array} \tag{3}$$

But this is immediate by taking $d_0 = d$ and placing the two squares (1) and (2) side by side. Ff clearly makes (3) commute for each d, so uniqueness of g gives that $g = Ff$. □

The restriction of the evaluation functor $ev_{\mathcal{A},\mathcal{C}}$ to $[\mathcal{A},\mathcal{C}]\times\mathcal{A}$ is *not* cocontinuous. If $D = \{0,1\}$ is the discrete subcategory of Set with objects 0 and 1, and, for $d \in \mathcal{D}$, $F_d : \mathsf{Set} \to \mathsf{Set}$ is $F_dX = X + d$ and $A_d : \mathsf{Set}$ is $A_d = d$, then $\varinjlim_{d\in\mathcal{D}} F_dA_d = 2 \not\cong 3 = (\varinjlim_{d\in\mathcal{D}} F_d)(\varinjlim_{d\in\mathcal{D}} A_d)$. Restricting to λ-accessible functors is therefore crucial to our development.

B. Initial algebras of λ-accessible functors

If F is an endofunctor on $\mathcal{A}$ then an *F-algebra* is a pair $(A, f : FA \to A)$. A morphism of F-algebras is a map $h : A \to A'$ in $\mathcal{A}$ such that $h \circ f = f' \circ Th$. Free algebras — and thus initial algebras, since the initial F-algebra is just the free F-algebra on the initial object — of accessible functors on locally presentable categories exist by the Adjoint Functor Theorem (see, e.g., the remark following Corollary 2.75 of [4]). However, we seek an explicit construction that reflects our intuitive understanding of how elements of data types are enumerated. There is a well-known such construction (see, e.g., [3]), but to interpret data types it must actually be λ-cocontinuous in the given functor. Like cocontinuity of functor application, this also is not hard to prove. But since we did not find a proof in the literature, to make this paper self-contained we explicitly present a variant of the well-known construction appropriate to our setting. We prove in Theorem 5 that it is indeed λ-cocontinuous.

For the rest of this section, let $\mathcal{A}$ be locally λ-presentable, let $F : \mathcal{A} \to \mathcal{A}$ be λ-accessible, let $\mathbf{0}$ be the initial object of $\mathcal{A}$, and let $o_A : \mathbf{0} \to A$ be the unique map to a given object A. Intuitively, the initial F-algebra is obtained as the colimit

$$\mathbf{0} \to F\mathbf{0} \to F^2\mathbf{0} \to \cdots F^\omega\mathbf{0} \to F^{\omega+1}\mathbf{0} \to \cdots F^\lambda\mathbf{0} \tag{4}$$

obtained by iterating F transfinitely λ times starting from $\mathbf{0}$. To formally define $F^\lambda\mathbf{0}$ we need to make this colimit precise.

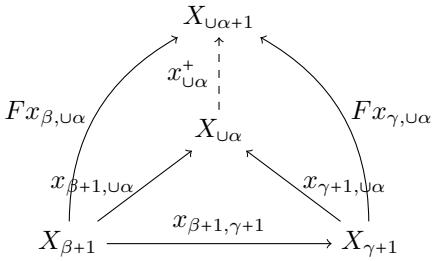
To that end, let Ord be the collection of ordinals, considered as a (large, posetal) category under inclusion. Write $[\beta,\alpha)$ for the (full) subcategory of Ord whose objects are ordinals γ with $\beta \le \gamma < \alpha$, and $\cup\alpha$ for the limit ordinal $\bigcup_{\beta<\alpha}\beta$. We define a diagram $(\{X_\alpha\}_{\alpha\in\mathit{Ord}}, \{x_{\beta,\alpha} : X_\beta \to X_\alpha\}_{\beta\le\alpha})$ by giving the actions of a functor $X : \mathit{Ord} \to \mathcal{A}$ on objects and morphisms by mutual induction together with an auxiliary function x^+_α:

$$\begin{array}{ll} X_0 = \mathbf{0} & x_{\beta,\beta} = id_{X_\beta} \\ X_{\alpha+1} = FX_\alpha & x_{\beta,\alpha+1} = x^+_\alpha \circ x_{\beta,\alpha} \\ X_{\cup\alpha} = \varinjlim_{\beta<\alpha} X_\beta & x_{\beta,\cup\alpha} = \varinjlim_{\gamma\in[\beta,\alpha)} x_{\beta,\gamma} \quad (\text{in } \mathcal{A}^{\to}) \end{array}$$

Here, $x^+_\alpha : X_\alpha \to X_{\alpha+1}$ is defined simultaneously with the above by $x^+_0 = o_{X_1}$, $x^+_{\alpha+1} = Fx^+_\alpha$, and, when $\alpha = \cup\alpha$ is a limit ordinal, taking x^+_α to be the unique map from the colimit X_α to the cocone with vertex $X_{\cup\alpha+1} = FX_{\cup\alpha}$. For all $\beta + 1 < \alpha$, the morphism $\{Fx_{\beta,\cup\alpha} : X_{\beta+1} \to FX_{\cup\alpha}\}_{\beta+1<\alpha}$ is defined to be the unique map satisfying

$$Fx_{\beta,\cup\alpha} = x^+_{\cup\alpha} \circ x_{\beta+1,\cup\alpha} \tag{5}$$

as in the diagram



These three properties are immediate by transfinite induction:

- $x_{\alpha,\alpha+1} = x^+_\alpha$;
- $x_{\beta,\alpha} = x_{\gamma,\alpha} \circ x_{\beta,\gamma}$ whenever $\beta \le \gamma \le \alpha$;
- $Fx_{\beta,\alpha} = x_{\beta+1,\alpha+1}$.

For $\alpha \in \mathit{Ord}$, define $F^\alpha\mathbf{0}$ to be the object X_α given as above.

To calculate F's fixed point, we first observe that, since F is λ-accessible, it preserves λ-directed colimits. In particular, it preserves $\varinjlim_{\alpha<\lambda} X_\alpha$. In literal terms, this means that the canonical map $c : \varinjlim_{\alpha<\lambda} FX_\alpha \to F\left(\varinjlim_{\alpha<\lambda} X_\alpha\right)$ is invertible. But since the cocone structure defining c is the same as that defining x^+_λ, this implies that x^+_λ is invertible. Putting $X := X_\lambda$, $x^- := (x^+_\lambda)^{-1} : FX \to X$ we have

Theorem 3. *(X, x^-) is the initial F-algebra.*

Proof. Fix an arbitrary F-algebra $(A, a : FA \to A)$. We will exhibit a unique F-algebra map from (X, x^-) to (A, a). For existence, we first construct a cocone $(\varphi_\alpha : X_\alpha \to A)_{\alpha<\lambda}$ by

$$\varphi_0 = o_A \qquad \varphi_{\alpha+1} = a \circ F\varphi_\alpha \qquad \varphi_{\cup\alpha} = \varinjlim_{\beta<\gamma} \varphi_\beta$$

where the last colimit is computed in $\mathcal{A}^{\to}$. In other words, $\varphi_{\cup\alpha}$ is the unique map satisfying

$$\forall \beta < \alpha.\ \varphi_\beta = \varphi_{\cup\alpha} \circ x_{\beta,\cup\alpha} \tag{6}$$

To verify the cocone property, we induct on $\beta \le \alpha$ to show that $\beta \le \alpha < \lambda$ implies $\varphi_\alpha \circ x_{\beta,\alpha} = \varphi_\beta$. The base case $\beta \le \beta$ is direct from the definitions; the induction step $\beta \le \alpha{+}1$ uses the fact, proved by induction on β, that $\varphi_{\beta+1} \circ x^+_\beta = \varphi_\beta$; and the limit case $\beta \le \cup\alpha$ is just (6). Now that we have a cocone from $\{X_\alpha\}$ to A, we can use the colimiting property of $X = X_\lambda$ to get a unique map $h : X \to A$ satisfying $\varphi_\alpha = h \circ x_{\alpha,\lambda}$.

We claim that h is an F-algebra map, i.e., that $h \circ x^- = a \circ Fh$. Since $x^- = (x^+_\lambda)^{-1}$, this is equivalent to $h = a \circ Fh \circ x^+_\lambda$. Since the domain of h is a colimit, it suffices to show that the composition of the two sides of this equation with each $x_{\alpha,\lambda}$, for $\alpha < \lambda$, yields the same morphism. Induction on α gives $h \circ x_{\alpha,\lambda} = a \circ Fh \circ x_{\alpha,\lambda+1}$, so h is indeed an F-algebra map.

To prove uniqueness, let $h' : (X, x^-) \to (A, a)$ be another F-algebra morphism, i.e., let $h' : X \to A$ be such that $h' \circ x^- = a \circ Fh'$, or, equivalently, $h' = a \circ Fh' \circ x^+_\lambda$. Since X is a colimit, to show $h = h'$ it suffices to show that $\varphi_\alpha = h' \circ x_{\alpha,\lambda}$. This is accomplished by a straightforward induction on α. □

With Theorem 3 in hand, we define our fixed point functor:

Definition 4. *Let $\mathcal{A}$ be locally λ-presentable. The action of the functor $\boldsymbol{\mu} : [\mathcal{A}, \mathcal{A}]_\lambda \to \mathcal{A}$ on objects is given by $\boldsymbol{\mu}(F) = F^\lambda \mathbf{0}$. Its action on morphisms is given by $\boldsymbol{\mu}(\varphi : F \Rightarrow G) = h$, where in_F and in_G are the structure maps for the initial F- and G-algebras, respectively, and h is the unique F-algebra morphism from $(\boldsymbol{\mu}F, in_F)$ to $(\boldsymbol{\mu}G, in_G \circ \varphi_{\boldsymbol{\mu}G})$ satisfying*

$$\begin{array}{ccc} F\boldsymbol{\mu}F & \xrightarrow{in_F} & \boldsymbol{\mu}F \\ {\scriptstyle Fh}\downarrow & & \downarrow{\scriptstyle !h} \\ F\boldsymbol{\mu}G & \xrightarrow{in_G \circ \varphi_{\boldsymbol{\mu}G}} & \boldsymbol{\mu}G \end{array}$$

The fact that the functor $\boldsymbol{\mu}$ is actually λ-cocontinuous is the special case of the following theorem for $\alpha = \lambda$.

Theorem 5. *If $F = \varinjlim_{d\in\mathcal{D}} F_d$, where $\mathcal{D}$ is λ-directed and each F_d is λ-accessible, then for each $\alpha \in Ord$, $F^\alpha \mathbf{0} = \varinjlim_{d\in\mathcal{D}} F^\alpha_d \mathbf{0}$.*

Proof. By induction on α.

$$\begin{aligned} F^0\mathbf{0} = \mathbf{0} &= \varinjlim_{d\in\mathcal{D}} \mathbf{0} = \varinjlim_{d\in\mathcal{D}} F^0_d \mathbf{0} \\ F^{\alpha+1}\mathbf{0} &= F(F^\alpha \mathbf{0}) \\ &=_{\text{IH}} F\big(\varinjlim_{d\in\mathcal{D}} F^\alpha_d \mathbf{0}\big) \\ &=_{\text{Lemma 2(i)}} \varinjlim_{d\in\mathcal{D}} F_d(F^\alpha_d \mathbf{0}) \\ &= \varinjlim_{d\in\mathcal{D}} F^{\alpha+1}_d \mathbf{0} \end{aligned}$$

$$\begin{aligned} F^{\cup\alpha}\mathbf{0} &= \varinjlim_{\beta<\alpha} F^\beta \mathbf{0} \\ &=_{\text{IH}} \varinjlim_{\beta<\alpha} \varinjlim_{d\in\mathcal{D}} F^\beta_d \mathbf{0} \\ &= \varinjlim_{d\in\mathcal{D}} \varinjlim_{\beta<\alpha} F^\beta_d \mathbf{0} \\ &= \varinjlim_{d\in\mathcal{D}} F^{\cup\alpha}_d \mathbf{0} \end{aligned}$$

□

C. Kan extensions

Definition 6 ([4, Def.2.12])**.** *A regular cardinal λ is* sharply smaller *than a regular cardinal μ, written $\mu \trianglerighteq \lambda$, if each λ-accessible category is μ-accessible.*

If S is a set of cardinals, we write $\mu \trianglerighteq S$ if $\mu \trianglerighteq \lambda$ for all $\lambda \in S$.

Lemma 7. (i) *For any set S of regular cardinals, there exists a regular cardinal μ such that $\mu \trianglerighteq S$.*
(ii) *If $\nu \trianglerighteq \mu \trianglerighteq \lambda$, then $\nu \trianglerighteq \lambda$.*
(iii) *If $\mu \trianglerighteq \lambda$ then $F \in [\mathcal{A}, \mathcal{B}]_\lambda$ implies $F \in [\mathcal{A}, \mathcal{B}]_\mu$.*

Proof. For parts (i) and (ii), see parts (6) and (7) of Example 2.13 in [4]. For (iii), see Remark 2.18(2) in *loc. cit.* □

A functor $F : \mathcal{A} \to \mathcal{B}$ *preserves λ-presentable objects* if FA is λ-presentable in $\mathcal{B}$ whenever A is λ-presentable in $\mathcal{A}$. The next lemma is actually a corollary of Theorem 2.19 of [4]. But since it plays a key role in bounding the cardinal λ in the interpretation of higher-kinded data types in locally λ-presentable categories, we prove it here to track exactly how this cardinal can increase.

Lemma 8. *If $F : \mathcal{A} \to \mathcal{B}$ is λ-accessible, then there exists a $\mu \trianglerighteq \lambda$ such that F is μ-accessible and F preserves μ-presentable objects. Moreover, if $\mathcal{I}$ is a set and $\{F_i : \mathcal{A} \to \mathcal{B}\}_{i\in\mathcal{I}}$ is a family of functors with each F_i λ_i-accessible, then there exists a regular cardinal μ such that $\mu \trianglerighteq \lambda_i$ for each $i \in \mathcal{I}$, and F_i preserves μ-presentable objects and is μ-accessible.*

Proof. For the first claim, let $\mathcal{A}$ and $\mathcal{B}$ be λ-accessible categories generated by the sets $\mathcal{A}_0$ and $\mathcal{B}_0$, respectively, of λ-presentable objects under λ-directed colimits. Let $F : \mathcal{A} \to \mathcal{B}$ be λ-accessible. Since each object of $\mathcal{B}$ is a λ-directed colimit of objects of $\mathcal{B}_0$, for each object X in $\mathcal{A}_0$, we can write $FX = \varinjlim_{i\in\mathcal{I}(X)} B(X, i)$, where each $B(X, i)$ is in $\mathcal{B}_0$, $\mathcal{I}(X)$ is λ-directed, and $|\mathcal{I}(X)|$ is minimal among cardinalities of λ-directed sets of objects in $\mathcal{B}_0$ with colimit FX. (Such a set $\mathcal{I}(X)$ exists by the well-orderedness of cardinals.)

Now, consider the set $\{\lambda\} \cup \{|\mathcal{I}(X)|\}_{X\in\mathcal{A}_0}$. By Lemma 7(i), there exist regular cardinals κ such that $\kappa \trianglerighteq \lambda$ and $\kappa \trianglerighteq |\mathcal{I}(X)|$ for all $X \in \mathcal{A}_0$. Take μ to be the least such cardinal κ. Then Remark 2.15(1) of [4] ensures that each FX is μ-presentable. By the result immediately following Remark 2.15, attributed by Adámek and Rosícky to Makkai and Paré, every μ-presentable object Y of $\mathcal{A}$ can be written as $Y = \varinjlim_{j\in\mathcal{J}} Y_j$, where each $Y_j \in \mathcal{A}_0$, and $\mathcal{J}$ is a λ-directed μ-small set. Since F is λ-cocontinuous, we then have that

$$FY = F\big(\varinjlim_{j\in\mathcal{J}} Y_j\big) = \varinjlim_{j\in\mathcal{J}} FY_j$$

is a μ-small colimit of μ-presentable objects, and is therefore μ-presentable by Proposition 1.16 of [4]. This shows that F

preserves μ-presentable objects. Since F is λ-accessible and $\mu \trianglerighteq \lambda$, Lemma 7(iii) guarantees that F is also μ-accessible.

For the second claim, we first apply the first claim to each F_i individually, obtaining $\{\mu_i\}_{i\in\mathcal{I}}$ such that, for each $i \in \mathcal{I}$, $\mu_i \trianglerighteq \lambda_i$, F_i preserves μ_i-presentable objects, and F_i is μ_i-accessible. Applying Example 2.13(6) of [4] gives a regular cardinal μ such that $\mu \trianglerighteq \mu_i$ for all $i \in \mathcal{I}$. The same argument as above shows that each F_i preserves μ-presentable objects, and Lemma 7(iii) ensures that each F_i is μ-accessible. □

The proof of Lemma 8 gives a recipe for computing the least cardinal μ whose existence it asserts: look at the action of F on each λ-presentable object X, find the least cardinal λ_X needed to write FX as a λ-directed colimit of size bounded by λ_X, and then find the least regular cardinal $\mu \trianglerighteq \bigcup_X \lambda_X \cup \{\lambda\}$.

We can now see that left Kan extensions exist and are easily computable in locally presentable categories. The following characterization [13] can be taken as a definition in this setting:

Theorem 9. *Let $\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$ be locally λ-presentable categories, let $\mathcal{A}_0$, $\mathcal{B}_0$, and $\mathcal{C}_0$ be the sets of λ-presentable objects in $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$, respectively. For functors $F \in [\mathcal{A},\mathcal{C}]_\lambda$ and $K \in [\mathcal{A},\mathcal{B}]_\lambda$, the* left Kan extension of F along K, $\mathsf{Lan}_K F : \mathcal{B} \to \mathcal{C}$, *can be computed by* $(\mathsf{Lan}_K F)Y = \varinjlim_{(P\in\mathcal{A}_0, f:KP\to Y)} FP$.

Here, the colimit defining $\mathsf{Lan}_K F$ exists because $\mathcal{C}$ is cocomplete and the colimit is indexed over a set: there are set-many choices for P because $\mathcal{A}_0$ is small, and set-many choices for f because $\mathcal{B}$ is locally λ-presentable and thus locally small.

Lemma 10. *Let $\mathcal{A},\mathcal{B},\mathcal{C},F$ and K be as in Theorem 9. If K preserves λ-presentable objects, then $\mathsf{Lan}_K F$ is λ-accessible. Moreover, for a fixed K, the functor $\mathsf{Lan}_K F$ is cocontinuous in F, i.e., for an arbitrary diagram $\{F_i\}_{i\in\mathcal{I}}$ in $[\mathcal{A},\mathcal{C}]_\lambda$ we have $\mathsf{Lan}_K\left(\varinjlim_{i\in\mathcal{I}} F_i\right) = \varinjlim_{i\in\mathcal{I}}(\mathsf{Lan}_K F_i)$.*

Proof. The first statement is proved in [2]. The second is immediate since $(\mathsf{Lan}_K -)$ is a left adjoint to the precomposition functor $- \circ K$; for details, see Chapter 6 of [13]. □

V. Semantics of Higher-kinded Data Types

We now show that every expression in $\mathcal{H}$ has an interpretation as an accessible functor on a locally presentable category. Fix a locally λ_0-presentable category $\mathcal{A}$. For $\varphi \subseteq \mathbb{T}$, and for $\lambda \geq \lambda_0$, define $\mathcal{A}_\lambda^{\varphi} := \prod_{\varphi^\kappa\in\varphi} \mathcal{A}_\lambda^\kappa$ where $\mathcal{A}_\lambda^\kappa$ is defined by induction on κ by $\mathcal{A}_\lambda^{\kappa_1\to\cdots\to\kappa_{|\kappa|}\to\star} = [\mathcal{A}_\lambda^{\kappa_1} \times \cdots \times \mathcal{A}_\lambda^{\kappa_{|\kappa|}}, \mathcal{A}]_\lambda$. For $\varphi^\kappa \in \varphi$, and $\lambda \geq \lambda_0$, let $\pi_\varphi : \mathcal{A}_\lambda^{\varphi} \to \mathcal{A}_\lambda^\kappa$ be the obvious projection (eliding the dependency on λ). We show in Theorem 11 that, for every $\mathcal{P} \subseteq \mathbb{T}$ and $E \in \mathcal{H}^{\mathcal{P}}(\kappa)$, there exists a $\lambda \geq \lambda_0$ such that E can be interpreted as a functor $[\![E]\!] \in [\mathcal{A}_\lambda^{\mathcal{P}}, \mathcal{A}_\lambda^\kappa]_\lambda$. In Corollary 12 we give sufficient conditions to ensure that $[\![E]\!] \in [\mathcal{A}_{\lambda_0}^{\mathcal{P}}, \mathcal{A}_{\lambda_0}^\kappa]_{\lambda_0}$.

Now, a naive attempt to prove this by induction runs into the following pitfall. Since the exponentials are contravariant in their first argument, if $\mu \trianglerighteq \lambda$ then $[\mathcal{A}_\lambda^{\mathcal{P}}, \mathcal{A}_\lambda^\kappa]_\lambda$ is a subcategory of $[\mathcal{A}_\lambda^{\mathcal{P}}, \mathcal{A}_\mu^\kappa]_\mu$, but not of $[\mathcal{A}_\mu^{\mathcal{P}}, \mathcal{A}_\mu^\kappa]_\mu$. Yet, in the proof, any time the cardinal increases we need to know that the subexpressions already proved to be λ-cocontinuous remain μ-cocontinuous for the new cardinal μ. We therefore prove a stronger statement: namely, that there always exists $\lambda \geq \lambda_0$ such that for *every* $\mu \trianglerighteq \lambda$, $[\![E]\!] \in [\mathcal{A}_\mu^{\mathcal{P}}, \mathcal{A}_\mu^\kappa]_\mu$. This induction loading is necessitated by the higher-kindedness of $\mathcal{H}$. Another subtlety arises for $\mathcal{L}an$-expressions, in that the functor extended along (represented by the subscript expression) must preserve presentable objects. Fortunately, much of the work needed for treating $\mathcal{L}an$-expressions has been established in the previous section. The main result of this paper is:

Theorem 11. *For every $E \in \mathcal{H}^{\mathcal{P}}(\kappa)$, there exists $\lambda \geq \lambda_0$ such that, for all $\mu \trianglerighteq \lambda$, E induces a functor $[\![E]\!] \in [\mathcal{A}_\mu^{\mathcal{P}}, \mathcal{A}_\mu^\kappa]_\mu$.*

Proof. The exponential adjunction (currying) gives an isomorphism $[\mathcal{A}_\lambda^{\mathcal{P}}, \mathcal{A}_\lambda^\kappa]_\lambda \simeq [\mathcal{A}_\lambda^{\mathcal{P}} \times (\mathcal{A}_\lambda^{\kappa_1} \times \cdots \times \mathcal{A}_\lambda^{\kappa_{|\kappa|}}), \mathcal{A}]_\lambda$. Throughout this proof, we adopt the convention of treating this isomorphism as equality, and using the right-hand side of the isomorphism to construct and manipulate elements of the left-hand side. We will, moreover, put $\mathcal{A}_\lambda^{\bar\kappa} = \mathcal{A}_\lambda^{\kappa_1} \times \cdots \times \mathcal{A}_\lambda^{\kappa_{|\kappa|}}$, and write the right-hand side of the isomorphism as $[\mathcal{A}_\lambda^{\mathcal{P}} \times \mathcal{A}_\lambda^{\bar\kappa}, \mathcal{A}]_\lambda$. Subsequently, all (un)currying will be done implicitly.

Most of the proof will be concerned with proving that certain functors of type $\mathcal{A}_\lambda^{\mathcal{P}} \to \mathcal{A}_\lambda^\kappa$ are λ-cocontinuous. By the convention above, this amounts to showing that corresponding functors of type $\mathcal{A}_\lambda^{\mathcal{P}} \times \mathcal{A}_\lambda^{\bar\kappa} \to \mathcal{A}$ are λ-cocontinuous, i.e., that, for every such functor G, λ-directed $\mathcal{D}$, $\vec{P} = \varinjlim_{d\in\mathcal{D}} \vec{P}_d$ in $\mathcal{A}_\lambda^{\mathcal{P}}$, and $\vec{F} = \varinjlim_{d\in\mathcal{D}} \vec{F}_d$ in $\mathcal{A}_\lambda^{\bar\kappa}$, we have $G\vec{P}\vec{F} = \varinjlim_{d\in\mathcal{D}} G\vec{P}_d\vec{F}_d$. This equation provides a general pattern which will recur throughout the proof, which proceeds by induction on E.

- $\underline{E = \lambda\vec\varphi.\mathbf{0}}$ Define $[\![E]\!]$ to be the constant functor $[\![E]\!]\vec{P}\vec{F} = \mathbf{0}$. Since $\mathbf{0} = \varinjlim_{d\in\mathcal{D}} \mathbf{0}$ for any $\mathcal{D}$, $[\![E]\!]\vec{P}\vec{F}$ is clearly in $[\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar\kappa}, \mathcal{A}]_\mu$ for any $\mu \trianglerighteq \lambda_0$, so take $\lambda = \lambda_0$.

- $\underline{E = \lambda\vec\varphi.\mathbf{1}}$ Define $[\![E]\!]$ to be the constant functor $[\![E]\!]\vec{P}\vec{F} = \mathbf{1}$. Since $\mathbf{1} = \varinjlim_{d\in\mathcal{D}} \mathbf{1}$ for any λ-directed $\mathcal{D}$, $[\![E]\!]\vec{P}\vec{F}$ is clearly in $[\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar\kappa}, \mathcal{A}]_\mu$ for any $\mu \trianglerighteq \lambda_0$, so take $\lambda = \lambda_0$.

- $\underline{E = \lambda\vec\varphi.\psi^\nu(E_1,...,E_{|\nu|})}$ We treat the case when $\psi^\nu \in \mathcal{P}$; the case when $\psi^\nu \in \vec\varphi$ is similar. By the induction hypothesis, for each $j = 1,...,|\nu|$, there exists $\lambda_j \geq \lambda_0$ such that $[\![E_j]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar\kappa}, \mathcal{A}]_\mu$ for all $\mu \trianglerighteq \lambda_j$. Also, for any λ, if $\vec{P} \in \mathcal{A}_\lambda^{\mathcal{P}}$ then $\pi_\psi\vec{P} = P_\psi \in [\mathcal{A}_\lambda^{\bar\nu}, \mathcal{A}]_\lambda$. By Lemma 7(i), there exists $\lambda \trianglerighteq \lambda_j$ for $j = 1,...,|\nu|$. Let $\mu \trianglerighteq \lambda$. Then $\mathcal{A}$ is locally μ-presentable. Define $[\![E]\!]\vec{P}\vec{F} = (\pi_\psi\vec{P})([\![E_1]\!]\vec{P}\vec{F},...,[\![E_{|\nu|}]\!]\vec{P}\vec{F})$. To see that $[\![E]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar\kappa}, \mathcal{A}]_\mu$, let $(\vec{P},\vec{F}) = \varinjlim_{d\in\mathcal{D}} (\vec{P},\vec{F})_d$ for some μ-directed set $\mathcal{D}$. Then $(\vec{P},\vec{F})_d \in \mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar\kappa}$, so that $P_\psi \in [\mathcal{A}_\mu^{\bar\nu}, \mathcal{A}]_\mu$. Then $P_\varphi = \varinjlim_{d\in\mathcal{D}} (P_\varphi)_d$ for each $\varphi \in \mathcal{P}$, $F_i = \varinjlim_{d\in\mathcal{D}} (F_i)_d$ for $i = 1,...,|\kappa|$, and $(P_\psi)_d = (\vec{P}_d)_\psi$. Also let $\vec{X}_d = ([\![E_1]\!]\vec{P}_d\vec{F}_d, \ldots, [\![E_{|\nu|}]\!]\vec{P}_d\vec{F}_d)$ for each $d \in \mathcal{D}$. Then $([\![E_1]\!]\vec{P}\vec{F},...,[\![E_{|\nu|}]\!]\vec{P}\vec{F}) = \varinjlim_{d\in\mathcal{D}} \vec{X}_d$ since each $[\![E_j]\!]$ is μ-cocontinuous. Using Lemma 2(i) to pass from the second to the third line below, we have

$$\begin{aligned}
[\![E]\!]\vec{P}\vec{F} &= P_\psi([\![E_1]\!]\vec{P}\vec{F},...,[\![E_{|\nu|}]\!]\vec{P}\vec{F}) \\
&= (\varinjlim_{d\in\mathcal{D}}(P_\psi)_d)(\varinjlim_{d\in\mathcal{D}}\vec{X}_d) \\
&= \varinjlim_{d\in\mathcal{D}}(P_d)_\psi\vec{X}_d \\
&= \varinjlim_{d\in\mathcal{D}}(\pi_\psi\vec{P}_d)([\![E_1]\!]\vec{P}_d\vec{F}_d,...,[\![E_{|\nu|}]\!]\vec{P}_d\vec{F}_d) \\
&= \varinjlim_{d\in\mathcal{D}}[\![E]\!]\vec{P}_d\vec{F}_d
\end{aligned}$$

• $\underline{E = \lambda\vec{\varphi}.E_1 + E_2}$ The induction hypothesis gives that, for $j \in \{1,2\}$, there exist $\lambda_j \geq \lambda_0$ such that $[\![E_j]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar{\kappa}}, \mathcal{A}]_\mu$ for all $\mu \trianglerighteq \lambda_j$. Let $\lambda \trianglerighteq \lambda_1, \lambda_2$ and $\mu \trianglerighteq \lambda$. Then $\mathcal{A}$ is locally μ-presentable, and $[\![E_1]\!], [\![E_2]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar{\kappa}}, \mathcal{A}]_\mu$. Define $[\![E]\!]\vec{P}\vec{F} = [\![E_1]\!]\vec{P}\vec{F} + [\![E_2]\!]\vec{P}\vec{F}$. To see that $[\![E]\!]$ is μ-cocontinuous, let $(\vec{P},\vec{F}) = \varinjlim_{d\in\mathcal{D}}(\vec{P},\vec{F})_d$ for some μ-directed set $\mathcal{D}$. Using μ-cocontinuity of $[\![E_1]\!]$ and $[\![E_2]\!]$ to pass from the second to the third line below, and commutativity of colimits to go from the third to the fourth, gives

$$\begin{aligned}
[\![E]\!]\vec{P}\vec{F} &= [\![E_1]\!]\vec{P}\vec{F} + [\![E_2]\!]\vec{P}\vec{F} \\
&= [\![E_1]\!](\varinjlim_{d\in\mathcal{D}}(\vec{P},\vec{F})_d) + [\![E_2]\!](\varinjlim_{d\in\mathcal{D}}(\vec{P},\vec{F})_d) \\
&= \varinjlim_{d\in\mathcal{D}}[\![E_1]\!](\vec{P},\vec{F})_d + \varinjlim_{d\in\mathcal{D}}[\![E_2]\!](\vec{P},\vec{F})_d \\
&= \varinjlim_{d\in\mathcal{D}}\big([\![E_1]\!](\vec{P},\vec{F})_d + [\![E_2]\!](\vec{P},\vec{F})_d\big) \\
&= \varinjlim_{d\in\mathcal{D}}[\![E]\!](\vec{P},\vec{F})_d
\end{aligned}$$

• $\underline{E = \lambda\vec{\varphi}.E_1 \times E_2}$ This case is identical to the previous one, except $[\![E]\!]$ is defined by $[\![E]\!]\vec{P}\vec{F} = [\![E_1]\!]\vec{P}\vec{F} \times [\![E_2]\!]\vec{P}\vec{F}$, and $\times$ commutes with the colimits $[\![E_1]\!](\vec{P},\vec{F})_d$ and $[\![E_2]\!](\vec{P},\vec{F})_d$ since λ-small limits commute with λ-directed colimits.

• $\underline{E = \lambda\vec{\varphi}.(\mu\psi^\nu.H)(E_1,...,E_{|\nu|})}$ The induction hypothesis gives that, for each $j = 1,...,n = |\nu|$, there exists $\lambda_j \geq \lambda_0$ such that $[\![E_j]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar{\kappa}}, \mathcal{A}]_\mu$ for all $\mu \trianglerighteq \lambda_j$. Moreover, the exponential adjunction gives a $\lambda_{n+1} \geq \lambda_0$ such that $[\![H]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar{\kappa}} \times \mathcal{A}_\mu^{\nu} \times \mathcal{A}_\mu^{\bar{\nu}}, \mathcal{A}]_\mu$ for $\mu \trianglerighteq \lambda_{n+1}$. Let $\lambda \trianglerighteq \{\lambda_0,\ldots,\lambda_{n+1}\}$, and let $\mu \trianglerighteq \lambda$. Then $\mathcal{A}$ is locally μ-presentable, and $[\![H]\!]$ and $[\![E_j]\!]$ are μ-cocontinuous. Define $[\![E]\!]\vec{P}\vec{F} = \boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}\vec{F}G)([\![E_1]\!]\vec{P}\vec{F},...,[\![E_n]\!]\vec{P}\vec{F})$, using Definition 4 with $\mathcal{A} := \mathcal{A}^\nu$. To see that $[\![E]\!]$ is μ-cocontinuous, let $(\vec{P},\vec{F}) = \varinjlim_{d\in\mathcal{D}}(\vec{P},\vec{F})_d$ for some μ-directed set $\mathcal{D}$. The proof is by case distinction on n. If $n = 0$, then

$$\begin{aligned}
[\![E]\!]\vec{P}\vec{F} &= \boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}\vec{F}G) \\
&= \boldsymbol{\mu}(G \mapsto [\![H]\!](\varinjlim_{d\in\mathcal{D}}\vec{P}_d\vec{F}_dG)) \\
&= \boldsymbol{\mu}(G \mapsto (\varinjlim_{d\in\mathcal{D}}[\![H]\!]\vec{P}_d\vec{F}_dG)) \\
&= \varinjlim_{d\in\mathcal{D}}\boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}_d\vec{F}_dG) \\
&= \varinjlim_{d\in\mathcal{D}}[\![E]\!]P_dF_d
\end{aligned}$$

where Theorem 5 is used to pass from the third to the fourth line. If $n > 0$, then letting $X_d^j = [\![E_j]\!]\vec{P}_d\vec{F}_d$, μ-cocontinuity of $[\![E_j]\!]$ gives that $[\![E_j]\!]\vec{P}\vec{F} = \varinjlim_{d\in\mathcal{D}}X_d^j$ for $j = 1,...,n$. Using Theorem 5 to pass from the first to the second line below, and Lemma 2(i) to pass from the second to the third, we get

$$\begin{aligned}
[\![E]\!]\vec{P}\vec{F} &= \boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}\vec{F}G)([\![E_1]\!]\vec{P}\vec{G},...,[\![E_n]\!]\vec{P}\vec{G}) \\
&= (\varinjlim_{d\in\mathcal{D}}\boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}_d\vec{F}_dG))(B_1,...,B_n) \\
&\qquad \text{where } B_j = \varinjlim_{d\in\mathcal{D}}X_d^j \text{ for } j = 1,...,n \\
&= \varinjlim_{d\in\mathcal{D}}\boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}_d\vec{F}_dG)(X_d^1,...,X_d^n) \\
&= \varinjlim_{d\in\mathcal{D}}[\![E]\!]\vec{P}_d\vec{F}_d
\end{aligned}$$

• $\underline{E = \lambda\vec{\varphi}.(\mathcal{L}an_{\vec{K}}^{\psi}G)\vec{E}}$ The induction hypothesis gives (i) $\lambda_1 \geq \lambda_0$ such that $[\![G]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar{\kappa}} \times \mathcal{A}_\mu^{\psi}, \mathcal{A}]_\mu$ for $\mu \trianglerighteq \lambda_1$; (ii) for each $i = 1,...,n = |\vec{K}|$, $\lambda_i' \geq \lambda_0$ such that $[\![K_i]\!] \in [\mathcal{A}_\mu^{\psi}, \mathcal{A}]_\mu$ for $\mu \trianglerighteq \lambda_i'$; and (iii) for each $j = 1,..,n$, $\lambda_j'' \geq \lambda_0$ such that $[\![E_j]\!] \in [\mathcal{A}_\mu^{\mathcal{P}} \times \mathcal{A}_\mu^{\bar{\kappa}}, \mathcal{A}]_\mu$ for $\mu \trianglerighteq \lambda_j''$. By Lemma 8, there exists a $\lambda \trianglerighteq \{\lambda_0\} \cup \{\lambda_j'\}_{1\leq j\leq n} \cup \{\lambda_j''\}_{1\leq j\leq n}$ such that, for each j, $[\![K_j]\!]$ is λ-accessible and preserves λ-presentable objects. Let $\mu \trianglerighteq \lambda$. Then each $[\![K_j]\!]$ is μ-accessible by Lemma 7(iii), and preserves μ-presentable objects by Remark 2.20 of [4], so the functor $\overrightarrow{[\![K]\!]} : \vec{Y} \mapsto ([\![K_1]\!]\vec{Y},...,[\![K_n]\!]\vec{Y})$ preserves μ-presentable objects as well. Also, $\mathcal{A}$ is locally μ-presentable, and $[\![G]\!]$, $\overrightarrow{[\![K]\!]}$, and each $[\![E_j]\!]$ are μ-cocontinuous. Define $[\![E]\!]\vec{P}\vec{F} = (\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}[\![G]\!]\vec{P}\vec{F})([\![E_1]\!]\vec{P}\vec{F},...,[\![E_n]\!]\vec{P}\vec{F})$.

To see that $[\![E]\!]$ is μ-cocontinuous, let $(\vec{P},\vec{F}) = \varinjlim_{d\in\mathcal{D}}(\vec{P},\vec{F})_d$ for some μ-directed set $\mathcal{D}$, $H_d = [\![G]\!]\vec{P}_d\vec{F}_d$ and $X_d = ([\![E_1]\!]\vec{P}_d\vec{F}_d,\ldots,[\![E_n]\!]\vec{P}_d\vec{F}_d)$. Then $\varinjlim_{d\in\mathcal{D}}H_d = [\![G]\!]\vec{P}\vec{F}$ and $\varinjlim_{d\in\mathcal{D}}X_d = ([\![E_1]\!]\vec{P}\vec{F},\ldots,[\![E_n]\!]\vec{P}\vec{F})$. The second statement of Lemma 10 gives that $\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}[\![G]\!]\vec{P}\vec{F} = \varinjlim_{d\in\mathcal{D}}\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}H_d$, and its first statement gives that each $\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}H_d$ is μ-accessible, and thus μ-cocontinuous. Using Lemma 2(i) to pass from the second to the third line, we have

$$\begin{aligned}
[\![E]\!]\vec{P}\vec{F} &= (\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}[\![G]\!]\vec{P}\vec{F})([\![E_1]\!]\vec{P}\vec{F},\ldots,[\![E_n]\!]\vec{P}\vec{F}) \\
&= (\varinjlim_{d\in\mathcal{D}}\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}H_d)(\varinjlim_{d\in\mathcal{D}}X_d) \\
&= \varinjlim_{d\in\mathcal{D}}(\mathsf{Lan}_{\overrightarrow{[\![K]\!]}}H_d)X_d \\
&= \varinjlim_{d\in\mathcal{D}}[\![E]\!]\vec{P}_d\vec{F}_d
\end{aligned}$$

□

Extracting from the proof of Theorem 11 the interpretation function $[\![-]\!] : E \in \mathcal{H}^{\mathcal{P}}(\kappa) \longmapsto [\![E]\!] \in [\mathcal{A}_\lambda^{\mathcal{P}}, \mathcal{A}_\lambda^{\kappa}]_\lambda$, we have:

$$[\![\lambda\vec{\varphi}.\mathbb{0}]\!]\vec{P}\vec{F} = \mathbf{0}$$

$$[\![\lambda\vec{\varphi}.\mathbb{1}]\!]\vec{P}\vec{F} = \mathbf{1}$$

$$[\![\lambda\vec{\varphi}.\psi^\nu(E_1,...,E_{|\nu|})]\!]\vec{P}\vec{F} = \begin{cases} P_\psi\vec{B} & \text{if } \psi \in \mathcal{P} \\ F_i\vec{B} & \text{if } \psi = \varphi_i \end{cases}$$

$$\text{where } B_j = [\![E_j]\!]\vec{P}\vec{F} \quad \text{for } j = 1,...,|\nu|$$

$$[\![\lambda\vec{\varphi}.E_1 + E_2]\!]\vec{P}\vec{F} = [\![E_1]\!]\vec{P}\vec{F} + [\![E_2]\!]\vec{P}\vec{F}$$

$$[\![\lambda\vec{\varphi}.E_1 \times E_2]\!]\vec{P}\vec{F} = [\![E_1]\!]\vec{P}\vec{F} \times [\![E_2]\!]\vec{P}\vec{F}$$

$$[\![\lambda\vec{\varphi}.(\mu\psi^\nu.H)(E_1,...,E_{|\nu|})]\!]\vec{P}\vec{F} = \boldsymbol{\mu}(G \mapsto [\![H]\!]\vec{P}\vec{F}G)\vec{B}$$

$$\text{where } B_j = [\![E_j]\!]\vec{P}\vec{F} \quad \text{for } j = 1,...,|\nu|$$

$$[\![\lambda\vec{\varphi}.(\mathcal{L}an_{\vec{K}}^{\psi}G)(E_1,\cdots,E_{|\vec{K}|})]\!]\vec{P}\vec{F} = (\mathsf{Lan}_{\vec{L}}[\![G]\!]\vec{P}\vec{F})\vec{B}$$

$$\text{where } L_j = [\![K_j]\!] \text{ and } B_j = [\![E_j]\!]\vec{P}\vec{F} \text{ for } j = 1,...,|\vec{K}|$$

Inspecting the proof of Theorem 11, we see that the only place where the cardinal λ_0 can increase is in the *Lan* case. This observation lets us derive a much tighter bound on λ. A *finitary polynomial* is an expression in $\mathcal{H}^{\varnothing}(\underline{k})$ that is built up using only (type) variables, $\mathbf{0}$, $\mathbf{1}$, $+$, and $\times$. Since $\mathbf{0}$ and $\mathbf{1}$ are λ-presentable for every λ, every finitary polynomial preserves λ-presentable objects for every λ. In particular,

Corollary 12. *Let $\mathcal{A}$ be locally λ_0-presentable. If $E \in \mathcal{H}^{\mathcal{P}}(\kappa)$ is constructed using only finitary polynomials as subscripts to Lan, then $[\![E]\!] \in [\mathcal{A}^{\mathcal{P}}_{\lambda_0}, \mathcal{A}^{\kappa}_{\lambda_0}]_{\lambda_0}$. In particular, when $\lambda_0 = \omega$, E's data elements can be effectively enumerated.*

VI. Examples

A. ADTs We have already seen that the natural number and list data types can be represented in $\mathcal{H}$, and similarly for all other ADTs, since their functorial expression representations are easily derived from the functors whose fixed points define them. Moreover, our semantics interprets ADTs as usual.

B. Nested types We have already seen that `PSeq` can be represented in $\mathcal{H}$ by $\mathtt{PSeq} := \lambda\alpha.\big(\mu\varphi^1.\lambda\beta.\beta + \varphi(\beta \times \beta)\big)\,\alpha$. Another nested type is that of λ-terms with variables of type `a`:

```
data Lam a where
  Var :: a → Lam a
  App :: Lam a → Lam a → Lam a
  Abs :: Lam (a + 1) → Lam a
```

The type constructor `Lam` can be represented in $\mathcal{H}$ by

$$\mathtt{Lam} := \lambda\alpha.\big(\mu\varphi^1.\lambda\beta.\beta + \varphi\beta \times \varphi\beta + \varphi(\beta{+}1)\big)\,\alpha$$

The higher-order functor interpreting the μ-subexpression is $H_{\mathsf{Lam}}\,FX = X + FX \times FX + F(X{+}1)$. We can enumerate data elements into the type `Lam a` by iterating H_{Lam} starting at the functor $K_{\mathbf{0}}X = \mathbf{0}$. Putting $F_k = (H_{\mathsf{Lam}})^k K_{\mathbf{0}}$, we compute the first few successive approximants $(F_k)_{k\geq 0}$ to $\mathsf{Lam}\,X$:

X	F_0X	F_1X	F_2X	F_3X
0	0	$0 = 0+0+0$	$1 = 0+0^2+1$	$5 = 0+1^2+4$
1	0	$1 = 1+0+0$	$4 = 1+1^2+2$	$26 = 1+4^2+9$
2	0	$2 = 2+0+0$	$9 = 2+2^2+3$	$99 = 2+9^2+16$
3	0	$3 = 3+0+0$	$16 = 3+3^2+4$	$284 = 3+16^2+25$
4	0	$4 = 4+0+0$	$25 = 4+4^2+5$	$665 = 4+25^2+36$
X	0	$X = X+0+0$	$(X{+}1)^2 = X{+}X^2{+}(X{+}1)$	$X{+}(X{+}1)^4{+}(X{+}2)^2$

The above decompositions of the F_kX directly correspond to how data elements of $\mathsf{Lam}\,X$ are constructed. If $X \simeq 2 = \{\bot, \top\}$, e.g., then $F_1(X)$ has the two elements $\mathsf{Var}\,\top$ and $\mathsf{Var}\,\bot$, while $F_2(X)$ has nine elements, namely, $\mathsf{Var}\,\top$, $\mathsf{Var}\,\bot$, $\mathsf{App}(\mathsf{Var}\,\bot)(\mathsf{Var}\,\bot)$, $\mathsf{App}(\mathsf{Var}\,\bot)(\mathsf{Var}\,\top)$, $\mathsf{App}(\mathsf{Var}\,\top)(\mathsf{Var}\,\bot)$, $\mathsf{App}(\mathsf{Var}\,\top)(\mathsf{Var}\,\top)$, $\mathsf{Abs}(\mathsf{Var}(\mathsf{inL}\,\bot))$, $\mathsf{Abs}(\mathsf{Var}(\mathsf{inL}\,\top))$, and $\mathsf{Abs}(\mathsf{Var}(\mathsf{inR}\,()))$. Here, Var, App, and Abs are the injections corresponding to the data constructors `Var`, `App`, and `Abs`.

C. Truly nested types We have seen above that `Bush` can be represented in $\mathcal{H}$ by $\mathtt{Bush} := \lambda\alpha.\big(\mu\varphi^1.\lambda\beta.\mathbb{1} + \beta \times \varphi(\varphi\beta)\big)\,\alpha$. The higher-order functor interpreting the μ-subexpression is $H_{\mathtt{Bush}}FX = 1 + X \times F(FX)$. The first approximants are

X	F_0X	F_1X	F_2X	F_3X
0	1	1	$1 = 1+0\cdot F_1 1$	$1 = 1+0\cdot F_2 1$
1	1	2	$4 = 1+1\cdot F_1 2 = 1+1{\cdot}3$	$26 = 1+1\cdot F_2 4 = 1+1{\cdot}25$
2	1	3	$9 = 1+2\cdot F_1 3 = 1+2{\cdot}4$	$201 = 1+2\cdot F_2 9 = 1+2{\cdot}10$
3	1	4	$16 = 1+3\cdot F_1 4 = 1+3{\cdot}5$	$868 = 1+3\cdot F_2 16 = 1+3{\cdot}289$
4	1	5	$25 = 1+4\cdot F_1 5 = 1+4{\cdot}6$	$2705 = 1+4\cdot F_2 25 = 1+4{\cdot}676$
X	1	$X+1$	$(X+1)^2 = 1+X\cdot F_1(X+1)$	$1+X\cdot F_2(X+1)^2$
F_kX	1	$X+2$	$((X+1)^2+1)^2$	$= 1+X\cdot((X+1)^2+1)^2$

The following nested type from [8] extends `Lam` with a constructor

$$\mathtt{Sub} :: \mathtt{Lam}\,(\mathtt{Lam}\,\mathtt{a}) \to \mathtt{Lam}\,\mathtt{a}$$

for explicit substitutions: The type constructor of the resulting data type can be represented in $\mathcal{H}$ by

$$\mathtt{LamES} := \lambda\alpha.\big(\mu\varphi^1.\lambda\beta.\beta + \varphi\beta \times \varphi\beta + \varphi(\beta{+}1) + \varphi(\varphi\beta)\big)\,\alpha$$

Putting $F_k = (H_{\mathtt{LamES}})^k K_{\mathbf{0}}$, where $H_{\mathtt{LamES}}FX = X + FX^2 + F(X+1) + F(FX)$, the first approximants are

X	F_0X	F_1X	F_2X	F_3X
0	0	0	$1 = 0+0^2+1+0$	$0+1^2+5+5$
1	0	1	$5 = 1+1^2+2+1$	$1+5^2+11+41$
2	0	2	$11 = 2+2^2+3+2$	$2+11^2+19+F^2 11$
3	0	3	$19 = 3+3^2+4+3$	$3+19^2+29+F_2 19$
4	0	4	$29 = 4+4^2+5+4$	$4+29^2+41+F_2 29$
5	0	5	$41 = 5+5^2+6+5$	$5+41^2+F_2 6+F_2 41$
X	0	X	$X{+}X^2{+}(X{+}1){+}X$	$2(X^2{+}3X{+}1){+}4X^2{+}15X{+}9$

The second approximant for $X = 2$ gives all the data elements for `Lam`, as well as $\mathsf{Sub}(\mathsf{Var}(\mathsf{Var}\,\bot))$ and $\mathsf{Sub}(\mathsf{Var}(\mathsf{Var}\,\top))$, where Sub is the injection corresponding to `Sub`.

D. GADTs We saw above that `Seq` can be represented in $\mathcal{H}$ by

$$\begin{aligned}\mathtt{Seq} := \lambda\alpha.\big(\mu\varphi^1.\lambda\beta.\beta &+ \big(\mathit{Lan}^{\gamma_1,\gamma_2}_{\gamma_1\times\gamma_2}(\varphi\gamma_1 \times \varphi\gamma_2)\big)\,\beta \\ &+ \big(\mathit{Lan}^{\gamma}_{\mathtt{Nat}\to\gamma}(\mathtt{Nat} \to \varphi\gamma)\big)\,\beta + \varphi(\mathtt{Nat} \to \beta) \times \mathtt{Nat}\big)\,\alpha\end{aligned}$$

where $\mathtt{Nat} = \mu\alpha.\mathbf{1} + \alpha$ and $\mathtt{Nat} \to D$ abbreviates $(\mathit{Lan}^{\beta}_{\mathtt{Nat}}\mathbf{1})D$. But the approximants here do *not* converge in ω steps. Indeed, $NX = \mathbb{N} \to X$ is not ω-cocontinuous: in Set, we have the ω-directed colimit $[0] \subseteq [1] \subseteq [2] \subseteq \cdots \subseteq \mathbb{N}$ for $[n] = \{0, \ldots, n{-}1\}$. Now, $id : \mathbb{N} \to \mathbb{N}$ is not in $N[n]$ for any n since the range of any morphism in $N[n]$ is finite whereas that of $id : \mathbb{N} \to \mathbb{N}$ is infinite, so the above colimit cannot be preserved by N.

Nevertheless, `Seq` has a well-defined semantics in our framework. That the functor N is $\aleph_1$-accessible follows from the first statement in Lemma 10 after noting that the remark after the proof of Lemma 8 ensures that the constantly `Nat`-valued functor preserves $\aleph_1$-presentable objects. The proof of Theorem 11 then shows that `Seq` induces a λ-presentable functor for some regular cardinal λ. In fact, this λ will be greater than $\aleph_1$, because the functor $N\,X = \mathtt{Nat} \to X$ appears in the subscript of the Kan extension, and thus λ must be taken to be large enough that N preserves λ-presentable objects. Since $\mathtt{Nat} \to \mathtt{Nat}$ is uncountable, N does not preserve countable objects, so λ must be strictly greater than $\aleph_1$.

We allow vectors in the subscript of *Lan* to handle, e.g.,

```
data G a b where
  c :: a → G a a
  d :: G a b → G b e → G e a → G (a, b) (b, e)
```

This data type can be represented in $\mathcal{H}$ by

$$\begin{aligned}\mathtt{G} := \mu\varphi^2.\lambda\alpha.\lambda\beta.\ &\Big(\mathit{Lan}^{\xi}_{(\xi,\xi)}\mathbb{1}\Big)\,(\alpha,\beta) \\ &+ \Big(\mathit{Lan}^{\xi,\upsilon,\zeta}_{(\xi\times\upsilon,\upsilon\times\zeta)}\varphi\xi\upsilon \times \varphi\upsilon\zeta \times \varphi\zeta\xi\Big)\,(\alpha,\beta)\end{aligned}$$

but cannot be represented using just unary *Lan*s since the two indices in the return type of `d` depend on one another.

E. Higher-kinded examples

• ***Fixed points*** The fixed point operator $\mathtt{Y} :: (\star \to \star) \to \star$ can be represented in $\mathcal{H}$ by $\mathtt{Y} := \lambda f^{\underline{1}}.\mu\alpha.f\alpha$, for which $[\![\mathtt{Y}]\!] =$

$[\![\mu\upsilon^{\underline{1}\to\underline{0}}.\lambda f^{\underline{1}}.f(\upsilon f)]\!]$. If the functor F is *strict* in the sense that $F\mathbf{0} \simeq \mathbf{0}$, then the carrier $\boldsymbol{\mu}F$ of the initial F-algebra will obviously be $\mathbf{0}$, too. This will happen whenever F has no data constructors with non-recursive types. In this case we can add a new data constructor with a non-recursive type (e.g., `Nil` for lists). The fixed point of the functor underlying the resulting data type will then be computable using $\mathtt{Y^1} = \lambda f^{\underline{1}}.\mu\alpha.\mathbf{1} + f\alpha$, for which $[\![\mathtt{Y^1}]\!]F = [\![\mathtt{Y}]\!](S \circ F)$ for the lifting monad $SX = X + 1$. Our initial algebra semantics proceeds by transfinite iteration from $\mathbf{0}$. This also gives a construction of the *free F-algebras* since, for $\mathtt{Y'} :: (\star \to \star) \to \star \to \star$ defined by $\mathtt{Y'} := \lambda f^{\underline{1}}.\lambda\alpha.\mu\beta.f(\beta{+}\alpha)$, we have $[\![\mathtt{Y'}]\!]\,F\,X = [\![\mathtt{Y}]\!](F\circ(+X))$.

• ***Joint initial algebras*** The above fixed points are instances of a more general pattern. We can define a functional $\mathtt{Mix} :: (\star \to \star) \to (\star \to \star) \to \star$ that computes a *joint initial algebra* of any F and G by $[\![\mathtt{Mix}]\!]\,F\,G$, where $\mathtt{Mix} = \lambda f^{\underline{1}}.\lambda g^{\underline{1}}.\mu\alpha.f\alpha + g\alpha$.

• ***Rose trees*** The data type of generic rose trees

```
data GRose f a where
  b :: a → f (GRose f a) → GRose f a
```

has underlying functor $HGFX = a \times F(GFX)$. `GRose` can be represented in $\mathcal{H}$ by $\lambda f^{\star\to\star}.\ \lambda\beta^{\star}.(\mu\psi^{(\star\to\star)\to(\star\to\star)}.\ \lambda\varphi^{\star\to\star}.\ \lambda\alpha^{\star}.\ \alpha \times \varphi(\psi\varphi\alpha))f\beta$. But while `GRose` obviously has a properly higher kind, it can also be represented using only type-level recursion by $\mathtt{GRose} := \lambda f^{\star\to\star}\lambda\alpha.\mu\beta.\alpha \times f\beta$.

• ***Truly higher-order data types*** A data type that genuinely requires recursion beyond the second order is

```
data Hf where
  HNil :: Hf
  HCons :: f (Hf) → H (f . f) → Hf
```

which is represented in $\mathcal{H}$ by $\Phi := \mu\varphi^{\underline{1}\to\underline{0}}.\lambda f^{\underline{1}}.\mathbf{1} + f(\varphi f) \times \varphi(f \circ f)$. We can similarly represent

```
data Hf a where
  HLeaf :: f a → Hf a
  HNode :: f (Hf a) → H (f . f) a → Hf a
```

by $\Psi := \mu\psi^{\underline{1}\to\underline{1}}.\lambda f^{\underline{1}}\lambda\alpha.f\alpha + f(\psi f\alpha) \times \psi(f \circ f)\alpha$. We invite the reader to try to find other higher-kinded examples like this, and to entertain their possible applications.

VII. Conclusion, Related Work, and Future Work

We have defined a large class of higher-kinded data types and provided a robust categorical framework to compute their semantics. As our examples illustrate, the breadth of data types that can be defined is quite considerable.

One obvious generalization of the present work is to extend our semantics to account for coinductive types. Locally presentable categories are also a convenient setting for studying coinduction. However, the construction of final coalgebras is more subtle than that of initial algebras, and extra care will have to be taken to ensure that interaction of coinduction with other type constructors does not produce unwanted side effects, and that the accessibility cardinals can still be bounded.

We may also want to incorporate dependent types. Here we may find rich interaction with the semantics of type theory, as well as the possibility of linking semantic complexities of data types with the proof-theoretic strength of the type system as a logic.

It would be especially interesting to explore which fragments of System F can be accommodated in the locally presentable setting. Coquand [5] presents a full model of System Fω using *categories of embeddings*. These are similar in spirit to locally presentable categories, but have the significant restriction that all maps be monomorphisms. (Also, the resulting model is not parametric.) As shown by Reynolds, full polymorphism cannot be interpreted in an arbitrary finitely presentable category (since these include Set), but perhaps smaller fragments of System F can be so interpreted.

We will ultimately want to give introduction and elimination rules, such as the following, so that Lan becomes a well-behaved type constructor from the syntactic standpoint:

$$\frac{x : F\vec{X} \quad f : K\vec{X} \to A}{\mathit{Lan}I(x,f) : (\mathit{Lan}_K F)A} \qquad \frac{\eta_{\vec{X}} : F\vec{X} \to G(K\vec{X}) \qquad \mathit{Gmap}_{X,Y} : (X \to Y) \to GX \to GY}{\mathit{Lan}\mathcal{E}(\eta, \mathit{Gmap}) : (\mathit{Lan}_K F)A \to GA}$$

$$\mathit{Lan}\mathcal{E}(\eta, \mathit{Gmap})(\mathit{Lan}I(x,f)) \longrightarrow \mathit{Gmap}(f)\eta(x)$$

But to get confluence and subject reduction, we will need η and Gmap to be natural and functorial. Alternatively, we can try to use parametricity to get these properties for free.

Related Work: While there are many treatments of GADTs — e.g., as initial algebras of dependent polynomial functors [7] and as indexed containers [12] — and some treatments of higher-kinded types, in the literature, few consider their semantics at all. All those that do give semantics are restricted in some fundamental way. For example, neither Johann and Ghani [10] nor Hamana and Fiore [9] consider nested GADTs. Moreover, Hamana and Fiore give semantics only in Set for the GADTs they do consider.

Versions of some of the results we prove here are "known" in folklore. For example, Martin and Gibbons [11] outline a semantics for nested types in the same spirit as ours. Their proposal is not quite right, however, because it asserts that every nested type can be interpreted as the least fixed point of an ω-cocontinuous functor whenever the category over which it is defined is ω-cocomplete and has all finite products and coproducts. This is not enough, however: those products must also commute with directed colimits. This is indeed the case when the underlying category is locally ω-presentable, in which case the functor is not just ω-cocontinuous, but actually ω-accessible, and thus its initial algebra can be computed in at most ω steps, i.e., its initial algebra semantics is effectively computable. Furthermore, locally ω-presentable categories are closed under exponentials, thus enabling generalization to functor categories. The same holds for any regular cardinal λ. As far as we know there is no treatment in the literature of higher-kinded data types that is as general and computationally effective as the one we give here.

Supported by National Science Foundation CCF-1713389.

REFERENCES

[1] http://ncatlab.org/nlab/show/transfinite+construction+of+free+algebras.

[2] https://mathoverflow.net/questions/112137/when-do-kan-extensions-preserve-limits-colimits.

[3] J. Adámek. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae*, 015:589–602, 1974.

[4] J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994.

[5] T. Coquand. Categories of embeddings. *Theoretical Computer Science*, 68:221–237, 1989.

[6] P. Gabriel and F. Ulmer. *Lokal Praesentierbare Kategorien*. Springer, 1971.

[7] N. Gambino and M. Hyland. Well-founded trees and dependent polynomial functors. In *TYPES*, pages 210–225, 2003.

[8] N. Ghani, T. Uustalu, and M. Hamana. Explicit substitutions and higher-order syntax. *Higher-order and Symbolic Computation*, 19:263–282, 2006.

[9] M. Hamama and M. Fiore. A foundation for GADTs and inductive families: Dependent polynomial functor approach. In *Workshop on Generic Programming*, pages 59–70, 2011.

[10] P. Johann and N. Ghani. Foundations for structured programming with GADTs. In *Principles of Programming Languages*, pages 297–308, 2008.

[11] C. Martin and J. Gibbons. On the semantics of nested datatypes. *Information Processing Letters*, 80(5):233–238, 2001.

[12] P. Morris and T. Altenkirch. Indexed containers. In *Logic in Computer Science*, pages 227–285, 2009.

[13] E. Riehl. *Category Theory in Context*. Dover, 2016.