Free Theorems for Nested Types

ANONYMOUS AUTHOR(S)

This paper considers parametricity and its consequent free theorems for nested data types. Rather than representing nested types via their Church encodings in a higher-kinded extension of System F, we adopt a functional programming perspective and design a polymorphic calculus that provides primitives for constructing a robust class of nested types directly as fixpoints. At the term level, our calculus supports pattern matching, map functions, and stylized recursion via folds for all such types. Our main technical contribution is the construction of a parametric model for this calculus. This is both delicate and challenging. To establish a suitable Identity Extension Lemma for our calculus, extra care must be taken in the formation of fixpoint types; all ADTs and all (truly) nested types appearing in the literature can nevertheless be expressed in our calculus. In addition, to ensure the existence of semantic fixpoints interpreting nested types, cocontinuity conditions on the functors interpreting all of the types in our calculus must be threaded throughout our model construction. Overall, this paper provides a blueprint for deriving principled parametricity results for practical programming languages, such as Haskell and Agda, that directly construct nested types as fixpoints rather than representing them via Church encodings. It also shows how parametric models can derive useful free theorems for programs over nested types. Like their counterparts for ADTs, these free theorems provide a principled alternative to proving program properties via induction, e.g., using the deep induction rules for nested types recently developed by Johann and Polonsky. Our approach is illustrated with several examples.

1 INTRODUCTION

Suppose we want to prove some property of programs over an algebraic data type (ADT) such as that of lists, coded in Agda as

```
data List (A : Set) : Set where
  nil : List A
  cons : A → List A → List A
```

A natural approach to the problem uses structural induction on the input data structure in question. This requires knowing not just the definition of the ADT of which the input data structure is an instance, but also the program text for the functions involved in the properties to be proved. For example, to prove by induction that mapping a polymorphic function over a list and then reversing the resulting list is the same as reversing the original list and then mapping the function over the result, we unwind the (recursive) definitions of the `reverse` and `map` functions over lists according to the inductive structure of the input list. Such data-driven induction proofs over ADTs are routine enough to be included in, say, an undergraduate functional programming course.

An alternative approach to proving results like the above `map-reverse` property for lists is to use parametricity, a formalization of extensional type-uniformity in polymorphic languages. Parametricity captures the intuition that a polymorphic program must act uniformly on all of its possible type instantiations; it is formalized as the requirement that every polymorphic program preserves all relations between any pair of types that it is instantiated with. Parametricity was originally put forth by Reynolds [Reynolds 1983] for System F [Girard et al. 1989], the formal calculus at the core of all polymorphic functional languages. It was later popularized as Wadler's "theorems for free" [Wadler 1989] because it allows the deduction of many properties of programs in such languages solely from their types, i.e., with no knowledge whatsoever of the text of the programs involved. To get interesting free theorems, Wadler's calculus included built-in ADTs; indeed, most of the free theorems in [Wadler 1989] are consequences of naturality for polymorphic list-processing functions. However, the techniques of [Wadler 1989] show that parametricity can

2020. 2475-1421/2020/1-ART1 $15.00
https://doi.org/

also be used to derive naturality properties for non-list ADTs, as well as results, like correctness of program optimizations like *short cut fusion* [Gill et al. 1993], that go beyond simple naturality.

This paper is about parametricity and free theorems for a polymorphic calculus with explicit syntax not just for ADTs, but for the generalization of ADTs known as *nested types* [Bird and Meertens 1998], as well. An ADT defines a *family of inductive data types*, one for each input type. For example, the `List` data type definition above defines a collection of data types `List A`, `List B`, `List (A × B)`, `List (List A)`, etc., each independent of all the others. By contrast, a nested type is an *inductive family of data types* that is defined over, or is defined mutually recursively with, (other) such data types. Since the structures of the data type at one input type can depend on those at other input types, the entire family of types must be defined at once. Examples of nested types include, trivially, ordinary ADTs, such as list and tree types; simple nested types, such as the data type

```
data PTree (A : Set) : Set where
  pleaf : A → PTree A
  pnode : PTree (A × A) → PTree A
```

of perfect trees, whose recursive occurrences never appear below other type constructors; "deep" nested types [Johann and Polonsky 2020], such as the data type

```
data Forest (A : Set) : Set where
  fempty : Forest A
  fnode : A → PTree (Forest A) → Forest A
```

of perfect forests, whose recursive occurrences appear below type constructors for other nested types; and truly nested types[1], such as the data type

```
data Bush (A : Set) : Set where
  bnil : Bush A
  bcons : A → Bush (Bush A) → Bush A
```

of bushes (also called *bootstrapped heaps* in [Okasaki 1999]), whose recursive occurrences appear below their own type constructors.

Suppose we now want to prove properties of functions over nested types. We might, for example, want to prove a `map-reverse` property for the functions on perfect trees in Figure 1, or for those on bushes[2] in Figure 2. A few well-chosen examples quickly convince us that such a property should indeed hold for perfect trees, and, drawing inspiration from the situation for ADTs, we easily construct a proof by induction on the input perfect tree. To formally establish this result, we could even prove it in Coq or Agda: each of these provers actually generates an induction rule for perfect trees, and the rule it generates gives the expected result because proving properties of perfect trees requires only that we induct over the top-level perfect tree in the recursive position. That is, any data internal to the input tree is left untouched.

Unfortunately, it is nowhere near as clear that analogous intuitive or formal inductive arguments can be made for the `map-reverse` property for bushes. Indeed, a proof by induction on the input bush must recursively induct over the bushes that are internal to the top-level bush in the recursive position. This is sufficiently delicate that no induction rule for bushes or other truly nested types was known until very recently, when *deep induction* [Johann and Polonsky 2020] was developed as a way to induct over *all* of the structured data present in an input. Deep induction thus not only gave the first principled and practically useful structural induction rules for bushes and other truly nested types, but has also opened the way for incorporating automatic generation of such rules for (truly) nested data types — and, eventually, even GADTs — into modern proof assistants.

[1] Nested types that are defined over themselves are known as *truly nested types*.

[2] To define the `foldBush` and `mapBush` functions in Figure 2 it is necessary to turn off Agda's termination checker.

```
reversePTree : ∀{A : Set} → PTree A → PTree A
reversePTree {A} = foldPTree {A} {PTree}
                     pleaf
                     (λp → pnode (mapPTree swap p))

foldPTree : ∀{A : Set} → {F : Set → Set} →
              ({B : Set} → B → F B) →
              ({B : Set} → F (B × B) → F B) →
              PTree A → F A
foldPTree n c (pleaf x) = n x
foldPTree n c (pnode p) = c (foldPTree n c p)

mapPTree : ∀{A B : Set} → (A → B) → PTree A → PTree B
mapPTree f (pleaf x) = pleaf (f x)
mapPTree f (pnode p) =
           pnode (mapPTree (λp → (f(π₁ p), f(π₂ p))) p)

swap : ∀{A : Set} → (A × A) → (A × A)
swap (x, y) = (y, x)
```

Fig. 1. `reversePTree` and auxiliary functions in Agda

```
reverseBush : ∀{A : Set} → Bush A → Bush A
reverseBush {A} = foldBush {A} {Bush} bnil balg

foldBush : ∀{A : Set} → {F : Set → Set} →
        ({B : Set} → F B) →
        ({B : Set} → B → F (F B) → F B) →
        Bush A → F A
foldBush bn bc bnil = bn
foldBush bn bc (bcons x bbx) =
        bc x (foldBush bn bc (mapBush (foldBush bn bc) bbx))

mapBush : ∀{A B : Set} → (A → B) → Bush A → Bush B
mapBush _ bnil = bnil
mapBush f (bcons x bbx) = bcons (f x) (mapBush (mapBush f) bbx)

balg : ∀{B : Set} → B → Bush (Bush B) → Bush B
balg x bnil = bcons x bnil
balg x (bcons bnil bbbx) = bcons x (bcons bnil bbbx)
balg x (bcons (bcons y bbx) bbbx) =
        bcons y (bcons (bcons x bbx) bbbx)
```

Fig. 2. `reverseBush` and auxiliary functions in Agda

Of course it is great to know that we *can*, at last, prove properties of programs over (truly) nested types by induction. But recalling that parametricity often provides a viable alternative to inductive proofs for ADTs, we might naturally ask:

Can we derive properties of functions over (truly) nested types from parametricity?

This paper answers the above question in the affirmative by constructing a parametric model for a polymorphic calculus providing primitives for *constructing* nested types directly via type-level recursion — rather than representing them indirectly by Church encodings as in, e.g., System F.

We introduce our calculus in Section 2. At the type level, it is the level-2-truncation of the higher-kinded calculus from [Johann and Polonsky 2019], augmented with a primitive type of natural transformations. To construct nested types, it carefully builds up type expressions not just from standard type variables, but also from type constructor variables of various arities, and includes an explicit μ-construct for type-level recursion with respect to these variables. The class of nested types it constructs is very robust and includes all (truly) nested types known from the literature. In Section 3 we give set and relational interpretations for these types. As is standard when modeling parametricity, types are interpreted as functors from environments interpreting their type variable contexts to sets or relations, as appropriate. But to ensure that these functors satisfy the cocontinuity properties needed for the semantic fixpoints interpreting μ-types to exist, set environments must map each k-ary type constructor variable to an appropriately cocontinuous k-ary functor on sets, relation environments must map each k-ary type constructor variable to an appropriately cocontinuous k-ary relation transformer, and these cocontinuity conditions must be threaded throughout the type interpretations in such a way that the resulting model is guaranteed to satisfy an appropriate Identity Extension Lemma (Theorem 22). Properly propagating the cocontinuity conditions turns out to be both subtle and challenging, and Section 4, where it is done, is where the bulk of the work in constructing our model lies. At the term level, our calculus includes primitive constructs for the actions on morphisms of the functors interpreting types, initial algebras for fixpoints of these functors, and structured recursion over elements of these initial algebras (i.e., map, in, and fold constructs, respectively). While our calculus does not support general recursion at the term level, it is strongly normalizing, so does perhaps edge us toward the kind of provably total practical programming language proposed at the end of [Wadler 1989]. In Section 5, we give set and relational interpretations for the terms of our calculus. As usual in parametric models, terms are interpreted as natural transformations from interpretations of the term contexts in which they are

formed to the interpretations of their types, and these must cohere in what is essentially a fibrational way. Immediately from the definitions of our interpretations we prove in Section 5.2 a scheme deriving free theorems that are consequences of naturality of functions that are polymorphic over nested types. This scheme is very general, is parameterized over both the data type and the type of the polymorphic function at hand, and has each of the above `map-reverse` theorems as instances. The relationship between naturality and parametricity has long been of interest, and our inclusion of a primitive type of natural transformations allows us to clearly delineate those free theorems that are consequences of naturality, and may thus hold even in non-parametric models, from those that require the full power of parametricity. In Section 5.4 we prove that our model satisfies an Abstraction Theorem (Theorem 28), and we derive several of this latter kind of free theorem from it in Section 6. Specifically, we state and prove (non-)inhabitation results in Sections 6.1 and 6.2, a free theorem for the type of a filter function on generalized rose trees in Section 6.3, the correctness of short cut fusion for lists in Section 6.4, and its generalization to nested types in Section 6.5.

It is well-known that there are no set-based parametric models of System F [Reynolds 1984]. So it is worth noting that our calculus does not contain all System F types, and is not impredicative, so a set-based parametric model is not *a priori* precluded. In fact, local finite presentability of Set makes it an obvious choice for interpreting nested types. We return to this point at several places below.

There is a long line of work on categorical models of parametricity for System F; see, e.g., [Bainbridge et al. 1990; Birkedal and Møgelberg 2005; Dunphy and Reddy 2004; Ghani et al. 2015; Hasegawa 1994; Jacobs 1999; Ma and Reynolds 1992; Robinson and Rosolini 1994]. To our knowledge, all categorical models that treat (algebraic) data types do so via their Church encodings, verifying in the just-constructed parametric model that each Church encoding is interpreted as the least fixpoint of the (first-order) functor interpreting the type constructor from which it was constructed. The present paper draws on this rich tradition of categorical models of parametricity for System F, but treats nested types as well as ADTs, and is the first to treat these types by direct construction via primitives rather than by Church encodings. This requires that we design our calculus in such a way that its types all have interpretations as appropriately cocontinuous functors, so that the existence of the fixpoints by which nested types are to be interpreted is ensured.

Like our calculus, Pitts' PolyPCF [Pitts 1998, 2000] includes primitives for constructing data types directly. Only list types are added in [Pitts 2000], but other polynomial ADTs are easily included as in [Pitts 1998]. Part of Pitts' motivation is to show that ADTs and their Church encodings have the same operational behavior in his calculus. We cannot even ask this question about our calculus, which cannot express Church encodings of even simple ADTs such as list, or pair, or sum types, since these are not functorial; nevertheless, we do prove the correctness of short cut fusion for nested types, whose operational analogue is valid in the parametric model Pitts constructs and is the means by which he proves his equivalence result. It would be interesting to know what operational analogues of functoriality and cocontinuity are needed to extend Pitts' parametricity results from a calculus with primitives for constructing data types modeled as fixpoints of first-order functors to one that also provides such primitives for data types modeled as fixpoints of higher-order functors.

We are not the first to consider parametricity at higher kinds. Atkey [Atkey 2012] constructs a parametric model for full System F_ω, but within the impredicative Calculus of Inductive Constructions rather than in a semantic category. His construction is in some ways similar to ours, but it represents data types using Church encodings rather than constructing them via primitives. However, the *fmap* functions associated to Atkey's functors must be *given* together with their underlying type constructors, and he need not impose cocontinuity conditions on his model to ensure that fixpoints of his functors exist. Unfortunately, Atkey does not indicate which type constructors support the *fmap* functions needed for them to be functors, but we suspect spelling this out explicitly would result in a full higher-kinded extension of the calculus presented here.

2 THE CALCULUS

2.1 Types

For each $k \geq 0$, we assume countable sets $\mathbb{T}^k$ of *type constructor variables of arity k* and $\mathbb{F}^k$ of *functorial variables of arity k*, all mutually disjoint. The sets of all type constructor variables and functorial variables are $\mathbb{T} = \bigcup_{k\geq 0} \mathbb{T}^k$ and $\mathbb{F} = \bigcup_{k\geq 0} \mathbb{F}^k$, respectively, and a *type variable* is any element of $\mathbb{T} \cup \mathbb{F}$. We use lower case Greek letters for type variables, writing ϕ^k to indicate that $\phi \in \mathbb{T}^k \cup \mathbb{F}^k$, and omitting the arity indicator k when convenient, unimportant, or clear from context. We reserve letters from the beginning of the alphabet to denote type variables of arity 0, i.e., elements of $\mathbb{T}^0 \cup \mathbb{F}^0$. We write $\overline{\zeta}$ for either a set $\{\zeta_1, ..., \zeta_n\}$ of type constructor variables or a set of functorial variables when the cardinality n of the set is unimportant or clear from context. If P is a set of type variables we write $P, \overline{\phi}$ for $P \cup \overline{\phi}$ when $P \cap \overline{\phi} = \emptyset$. We omit the vector notation for a singleton set, thus writing ϕ, instead of $\overline{\phi}$, for $\{\phi\}$.

Definition 1. *Let V be a finite subset of $\mathbb{T}$, P be a finite subset of $\mathbb{F}$, $\overline{\alpha}$ be a finite subset of $\mathbb{F}^0$ disjoint from P, and $\phi^k \in \mathbb{F}^k \setminus P$. The set $\mathcal{F}^P(V)$ of* functorial expressions *over P and V are given by*

$$\begin{aligned}\mathcal{F}^P(V) \ ::= \quad & \mathbb{0} \mid \mathbb{1} \mid \mathsf{Nat}^{\overline{\alpha}}\, \mathcal{F}^{\overline{\alpha}}(V)\, \mathcal{F}^{\overline{\alpha}}(V) \mid P\,\overline{\mathcal{F}^P(V)} \mid V\,\overline{\mathcal{F}^P(V)} \mid \mathcal{F}^P(V) + \mathcal{F}^P(V) \\ & \mid \mathcal{F}^P(V) \times \mathcal{F}^P(V) \mid \left(\mu\phi^{\,k}.\lambda\alpha_1 ... \alpha_k.\mathcal{F}^{P,\alpha_1,\ldots,\alpha_k,\phi}(V)\right)\overline{\mathcal{F}^P(V)}\end{aligned}$$

A *type* over P and V is any element of $\mathcal{F}^P(V)$.

The notation for types entails that an application $FF_1...F_k$ is allowed only when F is a type variable of arity k, or F is a subexpression of the form $\mu\phi^k.\lambda\alpha_1...\alpha_k.F'$. Moreover, if F has arity k then F must be applied to exactly k arguments. Accordingly, an overbar indicates a sequence of subexpressions whose length matches the arity of the type applied to it. The fact that types are always in *η-long normal form* avoids having to consider β-conversion at the level of types. In a subexpression $\mathsf{Nat}^{\overline{\alpha}} F\, G$, the Nat operator binds all occurrences of the variables in $\overline{\alpha}$ in F and G. Similarly, in a subexpression $\mu\phi^k.\lambda\overline{\alpha}.F$, the μ operator binds all occurrences of the variable ϕ, and the λ operator binds all occurrences of the variables in $\overline{\alpha}$, in the body F.

A *type constructor context* is a finite set Γ of type constructor variables, and a *functorial context* is a finite set Φ of functorial variables. In Definition 2, a judgment of the form $\Gamma; \Phi \vdash F$ indicates that the type F is intended to be functorial in the variables in Φ but not necessarily in those in Γ.

Definition 2. *The formation rules for the set $\mathcal{F} \subseteq \bigcup_{V\subseteq\mathbb{T}, P\subseteq\mathbb{F}} \mathcal{F}^P(V)$ of* well-formed types *are*

$$\frac{}{\Gamma; \Phi \vdash \mathbb{0}} \qquad \frac{}{\Gamma; \Phi \vdash \mathbb{1}}$$

$$\frac{\Gamma; \overline{\alpha^0} \vdash F \qquad \Gamma; \overline{\alpha^0} \vdash G}{\Gamma; \emptyset \vdash \mathsf{Nat}^{\overline{\alpha^0}} F\, G}$$

$$\frac{\phi^k \in \Gamma \cup \Phi \qquad \overline{\Gamma; \Phi \vdash F}}{\Gamma; \Phi \vdash \phi^k\,\overline{F}}$$

$$\frac{\Gamma; \overline{\gamma^0}, \overline{\alpha^0}, \phi^k \vdash F \qquad \overline{\Gamma; \Phi, \overline{\gamma^0} \vdash G}}{\Gamma; \Phi, \overline{\gamma^0} \vdash (\mu\phi^k.\lambda\overline{\alpha^0}.\, F)\,\overline{G}}$$

$$\frac{\Gamma; \Phi \vdash F \qquad \Gamma; \Phi \vdash G}{\Gamma; \Phi \vdash F + G} \qquad \frac{\Gamma; \Phi \vdash F \qquad \Gamma; \Phi \vdash G}{\Gamma; \Phi \vdash F \times G}$$

If F is a closed type we may write $\vdash F$, rather than $\emptyset; \emptyset \vdash F$, for the judgment that it is well-formed. Definition 2 ensures that the expected weakening rules for well-formed types hold, although weakening does not change the contexts in which types can be formed. If $\Gamma; \emptyset \vdash F$ and $\Gamma; \emptyset \vdash G$, then

our rules allow formation of the type $\Gamma;\emptyset \vdash \mathsf{Nat}^{\emptyset} F\, G$. Since $\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}} F\, G$ represents a natural transformation in $\overline{\alpha}$ from F to G, the type $\Gamma;\emptyset \vdash \mathsf{Nat}^{\emptyset} F\, G$ represents the arrow type $\Gamma \vdash F \to G$ in our calculus. The type $\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\, \mathbb{1}\, F$ similarly represents the $\forall$-type $\Gamma;\emptyset \vdash \forall \overline{\alpha}.F$. Note, however, that not all System F types are representable in our calculus.

Definition 2 allows the formation of all of the (closed) nested types from the introduction:

$$\begin{array}{lcl} \mathit{List}\,\alpha & = & \mu\beta.\, \mathbb{1} + \alpha \times \beta \;=\; (\mu\phi.\lambda\beta.\, \mathbb{1} + \beta \times \phi\beta)\,\alpha \\ \mathit{PTree}\,\alpha & = & (\mu\phi.\lambda\beta.\, \beta + \phi\,(\beta \times \beta))\,\alpha \\ \mathit{Forest}\,\alpha & = & (\mu\phi.\lambda\beta.\, \mathbb{1} + \beta \times \mathit{PTree}\,(\phi\beta))\,\alpha \\ \mathit{Bush}\,\alpha & = & (\mu\phi.\lambda\beta.\, \mathbb{1} + \beta \times \phi\,(\phi\beta))\,\alpha \end{array}$$

Each of these types can be considered either functorial in α or not, according to whether α is in the functorial or non-functorial context in which it is formed. In particular, if $\emptyset;\alpha \vdash \mathit{List}\,\alpha$, then $\vdash \mathsf{Nat}^{\alpha}\,\mathbb{1}\,(\mathit{List}\,\alpha)$ is well-formed; if $\alpha;\emptyset \vdash \mathsf{List}\;\alpha$, then it is not. Similarly, if $\mathit{Tree}\,\alpha\,\gamma = \mu\,\beta.\,\alpha+\beta\times\gamma\times\beta$, then $\gamma;\emptyset \vdash \mathsf{Nat}^{\alpha}(\mathit{List}\,\alpha)\,(\mathit{Tree}\,\alpha\,\gamma)$ is well-formed, but $\emptyset;\gamma \vdash \mathsf{Nat}^{\alpha}(\mathit{List}\,\alpha)\,(\mathit{Tree}\,\alpha\,\gamma)$ is not. Although types can be functorial in variables of arity greater than 0, the body F of a type $(\mu\phi.\lambda\overline{\alpha}.F)\,\overline{G}$ cannot. For example, the type $\mathit{GRose}\,\phi\,\alpha = \mu\beta.\alpha\times\phi\beta$ can be functorial or not in α, but cannot be functorial in ϕ. Moreover, $\mathit{GRose}\,\phi\,\alpha$ cannot be the (co)domain of a Nat type representing a natural transformation functorial in ϕ, and the type $\mu\phi.\lambda\alpha.\mathit{GRose}\,\phi\,\alpha$ is not well-formed in our system. These restrictions ensure that the set and relational interpretations of types in Section 3 are functors. In Section 7 we discuss how more expressive types might be accommodated.

Definition 2 allows well-formed types to be functorial in no variables. Functorial variables can also be demoted to non-functorial status. The proof is by induction on the structure of F.

Lemma 3. *If $\Gamma;\Phi,\phi^k \vdash F$, then $\Gamma,\psi^k;\Phi \vdash F[\phi^k :== \psi^k]$ is also derivable. Here, $F[\phi :== \psi]$ is the textual replacement of ϕ in F, meaning that all occurrences of $\phi\overline{\sigma}$ in F become $\psi\overline{\sigma}$.*

In addition to textual replacement, we also have a proper substitution operation on types. If F is a type over P and V, if P and V contain only type variables of arity 0, and if $k = 0$ for every occurrence of ϕ^k bound by μ in F, then we say that F is *first-order*; otherwise we say that F is *second-order*. Substitution for first-order types is the usual capture-avoiding textual substitution. We write $F[\alpha := \sigma]$ for the result of substituting σ for α in F, and $F[\alpha_1 := F_1, \ldots, \alpha_k := F_k]$, or $F[\overline{\alpha := F}]$ when convenient, for $F[\alpha_1 := F_1][\alpha_2 := F_2, \ldots, \alpha_k := F_k]$. Substitution for second-order types is defined below, where we adopt a similar notational convention for vectors of types. Note that it is not correct to substitute along non-functorial variables.

Definition 4. *If $\Gamma;\Phi,\phi^k \vdash H$ and if $\Gamma;\Phi,\overline{\alpha} \vdash F$ with $|\overline{\alpha}| = k$, then $\Gamma;\Phi \vdash H[\phi :=_{\overline{\alpha}} F]$. Similarly, if $\Gamma,\phi^k;\Phi \vdash H$, and if $\Gamma;\overline{\psi},\overline{\alpha} \vdash F$ with $|\overline{\alpha}| = k$ and $\Phi \cap \overline{\psi} = \emptyset$, then $\Gamma,\overline{\psi}';\Phi \vdash H[\phi :=_{\overline{\alpha}} F[\overline{\psi :== \psi'}]]$. Here, the operation $(\cdot)[\phi :=_{\overline{\alpha}} F]$ of* second-order type substitution along $\overline{\alpha}$ *is defined by:*

$$\begin{array}{lcl} \mathbb{0}[\phi :=_{\overline{\alpha}} F] & = & \mathbb{0} \\ \mathbb{1}[\phi :=_{\overline{\alpha}} F] & = & \mathbb{1} \\ (\mathsf{Nat}^{\overline{\beta}}\, G\, K)[\phi :=_{\overline{\alpha}} F] & = & \mathsf{Nat}^{\overline{\beta}}\,(G[\phi :=_{\overline{\alpha}} F])\,(K[\phi :=_{\overline{\alpha}} F]) \\ (\psi\overline{G})[\phi :=_{\overline{\alpha}} F] & = & \begin{cases} \psi\,\overline{G[\phi :=_{\overline{\alpha}} F]} & \text{if } \psi \neq \phi \\ F[\overline{\alpha := G[\phi :=_{\overline{\alpha}} F]}] & \text{if } \psi = \phi \end{cases} \\ (G + K)[\phi :=_{\overline{\alpha}} F] & = & G[\phi :=_{\overline{\alpha}} F] + K[\phi :=_{\overline{\alpha}} F] \\ (G \times K)[\phi :=_{\overline{\alpha}} F] & = & G[\phi :=_{\overline{\alpha}} F] \times K[\phi :=_{\overline{\alpha}} F] \\ ((\mu\psi.\lambda\overline{\beta}.\, G)\overline{K})[\phi :=_{\overline{\alpha}} F] & = & (\mu\psi.\lambda\overline{\beta}.\, G[\phi :=_{\overline{\alpha}} F])\,\overline{K[\phi :=_{\overline{\alpha}} F]} \end{array}$$

We note that $(\cdot)[\phi^0 :=_{\emptyset} F]$ coincides with first-order substitution. We also omit $\overline{\alpha}$ when convenient.

2.2 Terms

We now define our term calculus. To do so we assume an infinite set $\mathcal{V}$ of term variables disjoint from $\mathbb{T}$ and $\mathbb{F}$. If Γ is a type constructor context and Φ is a functorial context, then a *term context for* Γ *and* Φ is a finite set of bindings of the form $x : F$, where $x \in \mathcal{V}$ and $\Gamma; \Phi \vdash F$. We adopt the same conventions for denoting disjoint unions and for vectors in term contexts as for type constructor contexts and functorial contexts.

Definition 5. *Let* Δ *be a term context for* Γ *and* Φ. *The formation rules for the set of* well-formed terms over Δ *are*

$$\frac{\Gamma; \Phi \vdash F}{\Gamma; \Phi \mid \Delta, x : F \vdash x : F} \qquad \frac{\Gamma; \Phi \mid \Delta \vdash t : \mathbb{0} \qquad \Gamma; \Phi \vdash F}{\Gamma; \Phi \mid \Delta \vdash \bot_F t : F} \qquad \frac{}{\Gamma; \Phi \mid \Delta \vdash \top : \mathbb{1}}$$

$$\frac{\Gamma; \Phi \mid \Delta \vdash s : F}{\Gamma; \Phi \mid \Delta \vdash \mathsf{inl}\, s : F + G} \qquad \frac{\Gamma; \Phi \mid \Delta \vdash t : G}{\Gamma; \Phi \mid \Delta \vdash \mathsf{inr}\, t : F + G}$$

$$\frac{\Gamma; \Phi \vdash F, G \qquad \Gamma; \Phi \mid \Delta \vdash t : F + G \qquad \Gamma; \Phi \mid \Delta, x : F \vdash l : K \qquad \Gamma; \Phi \mid \Delta, y : G \vdash r : K}{\Gamma; \Phi \mid \Delta \vdash \mathsf{case}\, t\, \mathsf{of}\, \{x \mapsto l;\, y \mapsto r\} : K}$$

$$\frac{\Gamma; \Phi \mid \Delta \vdash s : F \qquad \Gamma; \Phi \mid \Delta \vdash t : G}{\Gamma; \Phi \mid \Delta \vdash (s, t) : F \times G} \qquad \frac{\Gamma; \Phi \mid \Delta \vdash t : F \times G}{\Gamma; \Phi \mid \Delta \vdash \pi_1 t : F} \qquad \frac{\Gamma; \Phi \mid \Delta \vdash t : F \times G}{\Gamma; \Phi \mid \Delta \vdash \pi_2 t : G}$$

$$\frac{\Gamma; \overline{\alpha} \vdash F \qquad \Gamma; \overline{\alpha} \vdash G \qquad \Gamma; \overline{\alpha} \mid \Delta, x : F \vdash t : G}{\Gamma; \emptyset \mid \Delta \vdash L_{\overline{\alpha}} x.t : \mathsf{Nat}^{\overline{\alpha}}\, F\, G}$$

$$\frac{\overline{\Gamma; \Phi \vdash K} \qquad \Gamma; \emptyset \mid \Delta \vdash t : \mathsf{Nat}^{\overline{\alpha}}\, F\, G \qquad \Gamma; \Phi \mid \Delta \vdash s : F[\overline{\alpha := K}]}{\Gamma; \Phi \mid \Delta \vdash t_{\overline{K}} s : G[\overline{\alpha := K}]}$$

$$\frac{\Gamma; \overline{\phi}, \overline{\gamma} \vdash H \qquad \overline{\Gamma; \overline{\beta}, \overline{\gamma} \vdash F} \qquad \overline{\Gamma; \overline{\beta}, \overline{\gamma} \vdash G}}{\Gamma; \emptyset \mid \emptyset \vdash \mathsf{map}_H^{\overline{F}, \overline{G}} : \mathsf{Nat}^{\emptyset}\, (\overline{\mathsf{Nat}^{\overline{\beta}, \overline{\gamma}}\, F\, G})\, (\mathsf{Nat}^{\overline{\gamma}}\, H[\overline{\phi :=_{\overline{\beta}} F}]\, H[\overline{\phi :=_{\overline{\beta}} G}])}$$

$$\frac{\Gamma; \phi, \overline{\alpha}, \overline{\gamma} \vdash H}{\Gamma; \emptyset \mid \emptyset \vdash \mathsf{in}_H : \mathsf{Nat}^{\overline{\beta}, \overline{\gamma}} H[\phi :=_{\overline{\beta}} (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}][\overline{\alpha := \beta}]\, (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}}$$

$$\frac{\Gamma; \phi, \overline{\alpha}, \overline{\gamma} \vdash H \qquad \Gamma; \overline{\beta}, \overline{\gamma} \vdash F}{\Gamma; \emptyset \mid \emptyset \vdash \mathsf{fold}_H^F : \mathsf{Nat}^{\emptyset}\, (\mathsf{Nat}^{\overline{\beta}, \overline{\gamma}}\, H[\phi :=_{\overline{\beta}} F][\overline{\alpha := \beta}]\, F)\, (\mathsf{Nat}^{\overline{\beta}, \overline{\gamma}}\, (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}\, F)}$$

In the rule for $L_{\overline{\alpha}} x.t$, the L operator binds all occurrences of the type variables in $\overline{\alpha}$ in the type of the term variable x and in the body t, as well as all occurrences of x in t. In the rule for $t_{\overline{K}} s$ there is one functorial expression in $\overline{K}$ for every functorial variable in $\overline{\alpha}$. In the rule for $\mathsf{map}_H^{\overline{F}, \overline{G}}$ there is one functorial expression F and one functorial expression G for each functorial variable in $\overline{\phi}$. Moreover, for each ϕ^k in $\overline{\phi}$ the number of functorial variables in $\overline{\beta}$ in the judgments for its corresponding functorial expresssions F and G is k. In the rules for in_H and fold_H^F, the functorial variables in $\overline{\beta}$ are fresh with respect to H, and there is one β for every α. (Recall from above that, in order for the types of in_H and fold_H^F to be well-formed, the length of α must equal the arity of ϕ.)

Substitution for terms is the obvious extension of the usual capture-avoiding textual substitution, and Definition 5 ensures that the expected weakening rules for well-formed terms hold.

Using Definition 5 we can represent the `reversePTree` function from Figure 1 in our calculus as

$$\vdash (\mathsf{fold}^{PTree\,\alpha}_{\beta+\phi(\beta\times\beta)})_{\emptyset}\; s : \mathsf{Nat}^{\alpha}(PTree\,\alpha)\,(PTree\,\alpha)$$

where

$$\begin{array}{lcl}
\mathsf{fold}^{PTree\,\alpha}_{\beta+\phi(\beta\times\beta)} & : & \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\alpha}(\alpha + PTree\,(\alpha\times\alpha))\,(PTree\,\alpha))\,(\mathsf{Nat}^{\alpha}(PTree\,\alpha)\,(PTree\,\alpha)) \\
\mathsf{in}_{\beta+\phi(\beta\times\beta)} & : & \mathsf{Nat}^{\alpha}(\alpha + PTree\,(\alpha\times\alpha))\,(PTree\,\alpha) \\
\mathsf{map}^{\alpha\times\alpha,\alpha\times\alpha}_{PTree\,\alpha} & : & \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\alpha}(\alpha\times\alpha)\,(\alpha\times\alpha))\,(\mathsf{Nat}^{\alpha}(PTree\,(\alpha\times\alpha))\,(PTree\,(\alpha\times\alpha)))
\end{array}$$

and *pleaf*,*pnode*, *swap*, and s are the terms

$$\begin{array}{l}
\vdash \mathsf{in}_{\beta+\phi(\beta\times\beta)} \circ (L_{\alpha}\; x.\,\mathsf{inl}\,x) \;:\; \mathsf{Nat}^{\alpha}\;\alpha\,(PTree\,\alpha) \\
\vdash \mathsf{in}_{\beta+\phi(\beta\times\beta)} \circ (L_{\alpha}\; x.\,\mathsf{inr}\,x) \;:\; \mathsf{Nat}^{\alpha}\;PTree(\alpha\times\alpha)\,(PTree\,\alpha) \\
\vdash L_{\alpha}p.\,(\pi_2 p, \pi_1 p) : \mathsf{Nat}^{\alpha}(\alpha\times\alpha)\,(\alpha\times\alpha) \\
\vdash L_{\alpha}t.\,\mathsf{case}\;t\;\mathsf{of}\;\{b \mapsto \mathit{pleaf}\;b;\; t' \mapsto \mathit{pnode}\,(((\mathsf{map}^{\alpha\times\alpha,\alpha\times\alpha}_{PTree\,\alpha})_{\emptyset}\;\mathit{swap})_{\alpha}\;t')\} : \mathsf{Nat}^{\alpha}(\alpha + PTree\,(\alpha\times\alpha))\;PTree\,\alpha
\end{array}$$

respectively. We can similarly represent the `reverseBush` function from Figure 2 as

$$\vdash (\mathsf{fold}^{Bush\,\alpha}_{\mathbb{1}+\beta\times\phi(\phi\beta)})_{\emptyset}\;\mathit{balg} : \mathsf{Nat}^{\alpha}(Bush\,\alpha)\,(Bush\,\alpha)$$

where

$$\vdash \mathsf{fold}^{Bush\,\alpha}_{\mathbb{1}+\beta\times\phi(\phi\beta)} : \mathsf{Nat}^{\emptyset}\,(\mathsf{Nat}^{\alpha}\,(\mathbb{1} + \alpha\times Bush\,(Bush\,\alpha)))\,(Bush\,\alpha))\,(\mathsf{Nat}^{\alpha}\,(Bush\,\alpha)\,(Bush\,\alpha))$$

and *bnil*, *bcons*, $\mathsf{in}^{-1}_{\mathbb{1}+\beta\times\phi(\phi\beta)}$, *balg*, and *consalg* are the terms

$$\begin{array}{l}
\vdash \mathsf{in}_{\mathbb{1}+\beta\times\phi(\phi\beta)} \circ (L_{\alpha}\; x.\,\mathsf{inl}\,x) \;:\; \mathsf{Nat}^{\alpha}\;\mathbb{1}\;(Bush\,\alpha) \\
\vdash \mathsf{in}_{\mathbb{1}+\beta\times\phi(\phi\beta)} \circ (L_{\alpha}\; x.\,\mathsf{inr}\,x) \;:\; \mathsf{Nat}^{\alpha}\;(\alpha\times Bush\,(Bush\,\alpha))\;(Bush\,\alpha) \\
\vdash (\mathsf{fold}^{(\mathbb{1}+\beta\times\phi(\phi\beta))[\phi:=Bush\,\alpha]}_{\mathbb{1}+\beta\times\phi(\phi\beta)})_{\emptyset}\;((\mathsf{map}^{(\mathbb{1}+\beta\times\phi(\phi\beta))[\phi:=Bush\,\alpha][\beta:=\alpha],Bush\,\alpha}_{\mathbb{1}+\beta\times\phi(\phi\beta)})_{\emptyset}\;\mathsf{in}_{\mathbb{1}+\beta\times\phi(\phi\beta)}) \\
\quad : \mathsf{Nat}^{\alpha}\;(Bush\,\alpha)\;(\mathbb{1} + \alpha\times Bush\,(Bush\,\alpha)) \\
\vdash L_{\alpha}\;s.\mathsf{case}\;s\;\mathsf{of}\;\{* \mapsto \mathit{bnil}_{\alpha}*;\; (a, bba) \mapsto \mathit{consalg}_{\alpha}(a, bba)\} : \mathsf{Nat}^{\alpha}\;(\mathbb{1} + \alpha\times Bush(Bush\,\alpha))\,(Bush\,\alpha) \\
\vdash L_{\alpha}\;(a, bba).\mathsf{case}\;((\mathsf{in}^{-1}_{\mathbb{1}+\beta\times\phi(\phi\beta)})_{Bush\,\alpha}\;bba)\;\mathsf{of}\;\{ \\
\qquad\qquad * \mapsto \mathit{bcons}_{\alpha}(a, \mathit{bnil}_{Bush\,\alpha}*); \\
\qquad\qquad (ba, bbba) \mapsto \mathsf{case}\;((\mathsf{in}^{-1}_{\mathbb{1}+\beta\times\phi(\phi\beta)})_{\alpha}\;ba)\;\mathsf{of}\;\{ \\
\qquad\qquad\qquad\qquad * \mapsto \mathit{bcons}_{\alpha}(a, \mathit{bcons}_{Bush\,\alpha}(\mathit{bnil}_{\alpha}*, bbba)); \\
\qquad\qquad\qquad\qquad (a', bba') \mapsto \mathit{bcons}_{\alpha}(a', \mathit{bcons}_{Bush\,\alpha}(\mathit{bcons}_{\alpha}(a, bba'), bbba))\}\} \\
\quad : \mathsf{Nat}^{\alpha}\;(\alpha\times Bush(Bush\,\alpha))\,(Bush\,\alpha)
\end{array}$$

respectively.

Our rule for forming Nat-types means that our calculus cannot express types of recursive functions, such as a concatenation function for perfect trees, that take as inputs a nested type and an argument of another type, both of which are parameterized over the same variable α. For the same reason, our calculus cannot express, e.g., the type of a zip function for bushes. The fundamental issue is that recursion is expressible only via folds, and a fold over a nested type must take a natural transformation as its argument and return a natural transformation as its result. In fact, even for ADTs there is a difference between which folds over them we can type when ADTs are viewed as ADTs — i.e., as fixpoints of types interpreted as first-order functors — and which folds we can type when ADTs are viewed as proper nested types — i.e., as fixpoints of types that are interpreted as higher-order functors — because the arity of the recursion variable ϕ in the fixpoint in the return type of fold must equal the number of variables bound by its Nat construct. For ADTs this number can be 0, making it possible to write types such as $\alpha; \emptyset \vdash \mathsf{Nat}^{\emptyset}\;(\mathsf{List}\;\alpha)\,(\mathsf{Nat}^{\emptyset}\alpha\,(\mathsf{List}\,(\mu\beta.1 + \beta)))$, all

of whose argument types are parameterized over the same type α, which are not possible for nested types even when they are semantically equivalent to ADTs. Our calculus can, however, express types of recursive functions that take multiple nested types as arguments, provided those nested types are parameterized over disjoint sets of type variables and the return type of the function is parameterized over only the variables occurring in the type of its final argument. Interestingly, even some recursive functions of a single proper nested type — e.g., a `reverseBush` function that is a true involution — cannot be expressed as folds because the algebra arguments needed to define them are again recursive functions with types of the same problematic forms. Allowing extra functorial variables in the codomains of Nat-types would recover typeability of, say, zip for bushes, but such extended Nat-types need not have semantics as ω-cocontinuous functors in Set, as will be required in Section 3. For example, the type $\emptyset; \gamma \vdash \mathsf{Nat}^{\emptyset}\,(\mu\beta.\mathbb{1} + \beta)\,\gamma$ has no such semantics. To remedy this, we would need to construct our models from semantic categories that are locally λ-presentable for $\lambda > \omega$. Adding more expressive recursion combinators could help, too. But since the expressivity issue for folds for nested types long pre-dates, and is somewhat orthogonal to, the issue of parametricity for languages that construct nested types as fixpoints, we do not consider it further in this paper other than to identify, in Section 7, some ideas for addressing it in future work.

The presence of the "extra" functorial variables in $\overline{\gamma}$ in the rules for $\mathsf{map}_H^{\overline{F},\overline{G}}$, in_H, and fold_H^F merit special mention. They allow us to map or fold polymorphic functions over nested types. Consider, for example, the polymorphic function *flatten* : $\mathsf{Nat}^{\beta}(\mathit{PTree}\,\beta)\,(\mathit{List}\,\beta)$ that uniformly maps perfect trees to lists. Even in the absence of extra variables the instance of map required to map each non-functorial monomorphic instantiation of *flatten* over a list of perfect trees is well-typed:

$$\frac{\Gamma; \alpha \vdash \mathit{List}\,\alpha \qquad \Gamma; \emptyset \vdash \mathit{PTree}\,F \qquad \Gamma; \emptyset \vdash \mathit{List}\,G}{\Gamma; \emptyset \mid \emptyset \vdash \mathsf{map}_{\mathit{List}\,\alpha}^{\mathit{PTree}\,F,\,\mathit{List}\,G} : \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\emptyset}\,(\mathit{PTree}\,F)\,(\mathit{List}\,G))\,(\mathsf{Nat}^{\emptyset}\,(\mathit{List}\,(\mathit{PTree}\,F))\,(\mathit{List}\,(\mathit{List}\,G)))}$$

But in the absence of $\overline{\gamma}$, the instance

$$\Gamma; \emptyset \mid \emptyset \vdash \mathsf{map}_{\mathit{List}\,\alpha}^{\mathit{PTree}\,\beta,\,\mathit{List}\,\beta} : \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\beta}(\mathit{PTree}\,\beta)\,(\mathit{List}\,\beta))\,(\mathsf{Nat}^{\beta}\,(\mathit{List}\,(\mathit{PTree}\,\beta))\,(\mathit{List}\,(\mathit{List}\,\beta)))$$

required to map the *polymorphic flatten* function over a list of perfect trees is not: in that setting the functorial contexts for F and G in the rule for $\mathsf{map}_H^{F,G}$ would have to be empty, but because the polymorphic *flatten* function is natural in some variable, say δ, it cannot possibly have a type of the form $\mathsf{Nat}^{\emptyset}\,F\,G$ that would be required for it to be the function input to map. Since untypeability of this instance of map is unsatisfactory in a polymorphic calculus, where we naturally expect to be able to manipulate entire polymorphic functions rather than just their monomorphic instances, we use the "extra" variables in $\overline{\gamma}$ to remedy the situation. Specifically, Definition 5 ensures that the instance of map needed to map the polymorphic *flatten* function is typeable as follows:

$$\frac{\Gamma; \alpha, \beta \vdash \mathit{List}\,\alpha \qquad \Gamma; \beta \vdash \mathit{PTree}\,\beta \qquad \Gamma; \beta \vdash \mathit{List}\,\beta}{\Gamma; \emptyset \mid \emptyset \vdash \mathsf{map}_{\mathit{List}\,\alpha}^{F,G} : \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\beta}(\mathit{PTree}\,\beta)\,(\mathit{List}\,\beta))\,(\mathsf{Nat}^{\beta}\,(\mathit{List}\,(\mathit{PTree}\,\beta))\,(\mathit{List}\,(\mathit{List}\,\beta)))}$$

Similar remarks explain the appearance of $\overline{\gamma}$ in the typing rules for in and fold.

3 INTERPRETING TYPES

We denote the category of sets and functions by Set. The category Rel has as its objects triples (A, B, R) where R is a relation between the objects A and B in Set, i.e., a subset of $A \times B$, and has as its morphisms from (A, B, R) to (A', B', R') pairs $(f : A \to A', g : B \to B')$ of morphisms in Set such that $(f\,a, g\,b) \in R'$ whenever $(a, b) \in R$. We write $R : \mathsf{Rel}(A, B)$ in place of (A, B, R) when convenient. If $R : \mathsf{Rel}(A, B)$ we write $\pi_1 R$ and $\pi_2 R$ for the *domain* A of R and the *codomain* B of R, respectively. If A : Set, then we write $\mathsf{Eq}_A = (A, A, \{(x, x) \mid x \in A\})$ for the *equality relation* on A.

The key idea underlying Reynolds' parametricity is to give each type $F(\alpha)$ with one free variable α both an *object interpretation* F_0 taking sets to sets and a *relational interpretation* F_1 taking relations $R : \mathsf{Rel}(A, B)$ to relations $F_1(R) : \mathsf{Rel}(F_0(A), F_0(B))$, and to interpret each term $t(\alpha, x) : F(\alpha)$ with one free term variable $x : G(\alpha)$ as a map t_0 associating to each set A a function $t_0(A) : G_0(A) \to F_0(A)$. These interpretations are to be given inductively on the structures of F and t in such a way that they imply two fundamental theorems. The first is an *Identity Extension Lemma*, which states that $F_1(\mathsf{Eq}_A) = \mathsf{Eq}_{F_0(A)}$, and is the essential property that makes a model relationally parametric rather than just induced by a logical relation. The second is an *Abstraction Theorem*, which states that, for any $R : \mathsf{Rel}(A, B)$, $(t_0(A), t_0(B))$ is a morphism in Rel from $(G_0(A), G_0(B), G_1(R))$ to $(F_0(A), F_0(B), F_1(R))$. The Identity Extension Lemma is similar to the Abstraction Theorem except that it holds for *all* elements of a type's interpretation, not just those that are interpretations of terms. Similar theorems are expected to hold for types and terms with any number of free variables.

The key to proving the Identity Extension Lemma (Theorem 22) in our setting is a familiar "cutting down" of the interpretations of universally quantified types to include only the "parametric" elements; the relevant types in our calculus are the Nat-types. This cutting down requires, as usual, that the set interpretations of our types are defined simultaneously with their relational interpretations. We give the set interpretations for our types in Section 3.1 and give their relational interpretations in Section 3.2. While the set interpretations are relatively straightforward, their relation interpretations are less so, mainly because of the cocontinuity conditions required to ensure that they are well-defined. We develop these conditions in Section 3.2, which separates Definitions 7 and 16 in space, but otherwise has no impact on the fact that they are given by mutual induction.

3.1 Interpreting Types as Sets

We interpret types in our calculus as ω-cocontinuous functors on locally finitely presentable categories [Adámek and Rosický 1994]. Since functor categories of locally finitely presentable categories are again locally finitely presentable, this ensures that the fixpoints interpreting μ-types in Set and Rel exist, and thus that both the set and relational interpretations of all of the types in Definition 2 are well-defined [Johann and Polonsky 2019]. To bootstrap this process, we interpret type variables as ω-cocontinuous functors in Definitions 6 and 14. If $\mathcal{C}$ and $\mathcal{D}$ are locally finitely presentable categories, we write $[\mathcal{C}, \mathcal{D}]$ for the category of ω-cocontinuous functors from $\mathcal{C}$ to $\mathcal{D}$.

Definition 6. *A* set environment *maps each type variable in* $\mathbb{T}^k \cup \mathbb{F}^k$ *to an element of* $[\mathsf{Set}^k, \mathsf{Set}]$. *A morphism* $f : \rho \to \rho'$ *for set environments* ρ *and* ρ' *with* $\rho|_{\mathbb{T}} = \rho'|_{\mathbb{T}}$ *maps each type constructor variable* $\psi^k \in \mathbb{T}$ *to the identity natural transformation on* $\rho\psi^k = \rho'\psi^k$ *and each functorial variable* $\phi^k \in \mathbb{F}$ *to a natural transformation from the* k*-ary functor* $\rho\phi^k$ *on* Set *to the* k*-ary functor* $\rho'\phi^k$ *on* Set. *Composition of morphisms on set environments is given componentwise, with the identity morphism mapping each set environment to itself. This gives a category of set environments and morphisms between them, which we denote* SetEnv.

When convenient we identify a functor in $[\mathsf{Set}^0, \mathsf{Set}]$ with its value on $*$ and consider a set environment to map a type variable of arity 0 to a set. If $\overline{\alpha} = \{\alpha_1, ..., \alpha_k\}$ and $\overline{A} = \{A_1, ..., A_k\}$, then we write $\rho[\overline{\alpha := A}]$ for the set environment ρ' such that $\rho'\alpha_i = A_i$ for $i = 1, ..., k$ and $\rho'\alpha = \rho\alpha$ if $\alpha \notin \{\alpha_1, ..., \alpha_k\}$. If ρ is a set environment we write Eq_ρ for the relation environment (see Definition 14) such that $\mathsf{Eq}_\rho v = \mathsf{Eq}_{\rho v}$ for every type variable v. The relational interpretations appearing in the second clause of Definition 7 are given in full in Definition 16.

Definition 7. *The* set interpretation $[\![\cdot]\!]^{\mathrm{Set}} : \mathcal{F} \to [\mathsf{SetEnv}, \mathsf{Set}]$ *is defined by*

$$
\begin{aligned}
[\![\Gamma; \Phi \vdash \mathbb{0}]\!]^{\mathrm{Set}}\rho &= 0\\
[\![\Gamma; \Phi \vdash \mathbb{1}]\!]^{\mathrm{Set}}\rho &= 1\\
[\![\Gamma; \emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathrm{Set}}\rho &= \{\eta : \lambda\overline{A}.\, [\![\Gamma; \overline{\alpha} \vdash F]\!]^{\mathrm{Set}}\rho[\overline{\alpha := A}] \Rightarrow \lambda\overline{A}.\, [\![\Gamma; \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}\rho[\overline{\alpha := A}]\\
&\qquad \mid \forall \overline{A}, \overline{B} : \mathsf{Set}.\forall \overline{R : \mathsf{Rel}(A, B)}.\\
&\qquad\quad (\eta_{\overline{A}}, \eta_{\overline{B}}) : [\![\Gamma; \overline{\alpha} \vdash F]\!]^{\mathrm{Rel}}\mathsf{Eq}_{\rho}[\overline{\alpha := R}] \to [\![\Gamma; \overline{\alpha} \vdash G]\!]^{\mathrm{Rel}}\mathsf{Eq}_{\rho}[\overline{\alpha := R}]\}\\
[\![\Gamma; \Phi \vdash \phi\overline{F}]\!]^{\mathrm{Set}}\rho &= (\rho\phi)\ \overline{[\![\Gamma; \Phi \vdash F]\!]^{\mathrm{Set}}\rho}\\
[\![\Gamma; \Phi \vdash F + G]\!]^{\mathrm{Set}}\rho &= [\![\Gamma; \Phi \vdash F]\!]^{\mathrm{Set}}\rho + [\![\Gamma; \Phi \vdash G]\!]^{\mathrm{Set}}\rho\\
[\![\Gamma; \Phi \vdash F \times G]\!]^{\mathrm{Set}}\rho &= [\![\Gamma; \Phi \vdash F]\!]^{\mathrm{Set}}\rho \times [\![\Gamma; \Phi \vdash G]\!]^{\mathrm{Set}}\rho\\
[\![\Gamma; \Phi, \overline{\gamma} \vdash (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}]\!]^{\mathrm{Set}}\rho &= (\mu T^{\mathrm{Set}}_{H,\rho})\overline{[\![\Gamma; \Phi, \overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho}\\
&\quad \textit{where } T^{\mathrm{Set}}_{H,\rho}\, F = \lambda\overline{A}.[\![\Gamma; \overline{\gamma}, \phi, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}}\rho[\phi := F][\overline{\alpha := A}]\\
&\quad \textit{and } T^{\mathrm{Set}}_{H,\rho}\, \eta = \lambda\overline{A}.[\![\Gamma; \overline{\gamma}, \phi, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}}\, id_{\rho}[\phi := \eta][\overline{\alpha := id_A}]
\end{aligned}
$$

If $\rho \in \mathsf{SetEnv}$ and $\vdash F$ then we write $[\![\vdash F]\!]^{\mathrm{Set}}$ instead of $[\![\vdash F]\!]^{\mathrm{Set}}\rho$ since the environment is immaterial. The third clause of Definition 7 does indeed define a set: local finite presentability of Set and ω-cocontinuity of $[\![\Gamma; \overline{\alpha} \vdash F]\!]^{\mathrm{Set}}\rho$ ensure that $\{\eta : [\![\Gamma; \overline{\alpha} \vdash F]\!]^{\mathrm{Set}}\rho \Rightarrow [\![\Gamma; \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}\rho\}$ (which contains $[\![\Gamma; \emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathrm{Set}}\rho$) is a subset of $\left\{(([\![\Gamma; \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}\rho[\overline{\alpha := S}])^{([\![\Gamma; \overline{\alpha} \vdash F]\!]^{\mathrm{Set}}\rho[\overline{\alpha := S}])} \;\middle|\; \overline{S} = (S_1, \ldots, S_{|\overline{\alpha}|}),\ \text{and}\ S_i \text{ is a finite set for } i = 1, \ldots, |\overline{\alpha}|\right\}$. There are countably many choices of tuples $\overline{S}$, each of which gives rise to a morphism from $[\![\Gamma; \overline{\alpha} \vdash F]\!]^{\mathrm{Set}}\rho[\overline{\alpha := S}]$ to $[\![\Gamma; \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}\rho[\overline{\alpha := S}]$. But there are only Set-many choices of morphisms between any two objects since Set is locally small. We note that $[\![\Gamma; \emptyset \vdash \mathsf{Nat}^{\overline{\alpha}} F\, G]\!]^{\mathrm{Set}}$ is ω-cocontinuous because it is a constant functor (and that allowing functorial variables not in $\overline{\alpha}$ in G fails here, as shown in Section 2.2). Interpretations of Nat-types ensure that $[\![\Gamma \vdash F \to G]\!]^{\mathrm{Set}}$ and $[\![\Gamma \vdash \forall\overline{\alpha}.F]\!]^{\mathrm{Set}}$ are as expected in any parametric model.

To make sense of the last clause in Definition 7, we need to know that, for each $\rho \in \mathsf{SetEnv}$, $T^{\mathrm{Set}}_{H,\rho}$ is an ω-cocontinuous endofunctor on $[\mathsf{Set}^k, \mathsf{Set}]$, and thus admits a fixpoint. Since $T^{\mathrm{Set}}_{H,\rho}$ is defined in terms of $[\![\Gamma; \overline{\gamma}, \phi, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}}$, this means that interpretations of types must be such functors, which in turn means that the actions of set interpretations of types on objects and on morphisms in SetEnv are intertwined. Fortunately, we know from [Johann and Polonsky 2019] that, for every $\Gamma; \overline{\alpha} \vdash G$, $[\![\Gamma; \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}$ is actually in $[\mathsf{Set}^k, \mathsf{Set}]$ where $k = |\overline{\alpha}|$. This means that for each $[\![\Gamma; \overline{\gamma}, \phi^k, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}}$, the corresponding operator T^{Set}_H can be extended to a *functor* from SetEnv to $[[\mathsf{Set}^k, \mathsf{Set}], [\mathsf{Set}^k, \mathsf{Set}]]$. The action of T^{Set}_H on an object $\rho \in \mathsf{SetEnv}$ is given by the higher-order functor $T^{\mathrm{Set}}_{H,\rho}$, whose actions on objects (functors in $[\mathsf{Set}^k, \mathsf{Set}]$) and morphisms (natural transformations between such functors) are given in Definition 7. Its action on a morphism $f : \rho \to \rho'$ is the higher-order natural transformation $T^{\mathrm{Set}}_{H,f} : T^{\mathrm{Set}}_{H,\rho} \to T^{\mathrm{Set}}_{H,\rho'}$ whose action on $F : [\mathsf{Set}^k, \mathsf{Set}]$ is the natural transformation $T^{\mathrm{Set}}_{H,f}\, F : T^{\mathrm{Set}}_{H,\rho}\, F \to T^{\mathrm{Set}}_{H,\rho'}\, F$ whose component at $\overline{A}$ is $(T^{\mathrm{Set}}_{H,f}\, F)_{\overline{A}} = [\![\Gamma; \overline{\gamma}, \phi, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}} f[\phi := id_F][\overline{\alpha := id_A}]$. The next definition uses the functor T^{Set}_H to define the actions of functors interpreting types on morphisms between set environments.

Definition 8. *Let $f : \rho \to \rho'$ be a morphism between set environments ρ and ρ' (so that $\rho|_{\mathbb{T}} = \rho'|_{\mathbb{T}}$). The action $[\![\Gamma; \Phi \vdash F]\!]^{\mathrm{Set}} f$ of $[\![\Gamma; \Phi \vdash F]\!]^{\mathrm{Set}}$ on the morphism f is given as follows:*

- *If $\Gamma; \Phi \vdash \mathbb{0}$ then $[\![\Gamma; \Phi \vdash \mathbb{0}]\!]^{\mathrm{Set}} f = id_0$*
- *If $\Gamma; \Phi \vdash \mathbb{1}$ then $[\![\Gamma; \Phi \vdash \mathbb{1}]\!]^{\mathrm{Set}} f = id_1$*

- *If* $\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\ F\ G$ *then* $[\![\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\ F\ G]\!]^{\mathrm{Set}} f = id_{[\![\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\ F\ G]\!]^{\mathrm{Set}}\rho}$
- *If* $\Gamma;\Phi \vdash \phi\overline{F}$ *then* $[\![\Gamma;\Phi \vdash \phi\overline{F}]\!]^{\mathrm{Set}} f : [\![\Gamma;\Phi \vdash \phi\overline{F}]\!]^{\mathrm{Set}}\rho \to [\![\Gamma;\Phi \vdash \phi\overline{F}]\!]^{\mathrm{Set}}\rho' = (\rho\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho} \to (\rho'\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho'}$ *is defined by* $[\![\Gamma;\Phi \vdash \phi\overline{F}]\!]^{\mathrm{Set}} f = (f\phi)_{\overline{[\![\Gamma;\Phi\vdash F]\!]^{\mathrm{Set}}\rho'}} \circ (\rho\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}} f} = (\rho'\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}} f} \circ (f\phi)_{\overline{[\![\Gamma;\Phi\vdash F]\!]^{\mathrm{Set}}\rho}}$. *The latter equality holds because* $\rho\phi$ *and* $\rho'\phi$ *are functors and* $f\phi : \rho\phi \to \rho'\phi$ *is a natural transformation, so the following naturality square commutes:*

$$\begin{array}{ccc} (\rho\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho} & \xrightarrow{(f\phi)_{\overline{[\![\Gamma;\Phi\vdash F]\!]^{\mathrm{Set}}\rho}}} & (\rho'\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho} \\ {\scriptstyle (\rho\phi)\overline{[\![\Gamma;\Phi\vdash F]\!]^{\mathrm{Set}} f}}\downarrow & & \downarrow{\scriptstyle (\rho'\phi)\overline{[\![\Gamma;\Phi\vdash F]\!]^{\mathrm{Set}} f}} \\ (\rho\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho'} & \xrightarrow{(f\phi)_{\overline{[\![\Gamma;\Phi\vdash F]\!]^{\mathrm{Set}}\rho'}}} & (\rho'\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho'} \end{array} \tag{1}$$

- *If* $\Gamma;\Phi \vdash F+G$ *then* $[\![\Gamma;\Phi \vdash F+G]\!]^{\mathrm{Set}} f$ *is defined by* $[\![\Gamma;\Phi \vdash F+G]\!]^{\mathrm{Set}} f(\mathsf{inl}\ x) = \mathsf{inl}\,([\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}} f x)$ *and* $[\![\Gamma;\Phi \vdash F+G]\!]^{\mathrm{Set}} f(\mathsf{inr}\ y) = \mathsf{inr}\,([\![\Gamma;\Phi \vdash G]\!]^{\mathrm{Set}} f y)$
- *If* $\Gamma;\Phi \vdash F \times G$ *then* $[\![\Gamma;\Phi \vdash F \times G]\!]^{\mathrm{Set}} f = [\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}} f \times [\![\Gamma;\Phi \vdash G]\!]^{\mathrm{Set}} f$
- *If* $\Gamma;\Phi,\overline{\gamma} \vdash (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}$ *then*

$$\begin{array}{rcl} [\![\Gamma;\Phi,\overline{\gamma} \vdash (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}]\!]^{\mathrm{Set}} f & : & [\![\Gamma;\Phi,\overline{\gamma} \vdash (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}]\!]^{\mathrm{Set}}\rho \to [\![\Gamma;\Phi,\overline{\gamma} \vdash (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}]\!]^{\mathrm{Set}}\rho' \\ & = & (\mu T^{\mathrm{Set}}_{H,\rho})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho} \to (\mu T^{\mathrm{Set}}_{H,\rho'})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho'} \end{array}$$

is defined by

$$\begin{array}{rcl} & & (\mu T^{\mathrm{Set}}_{H,f})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho'} \circ (\mu T^{\mathrm{Set}}_{H,\rho})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}} f} \\ = & & (\mu T^{\mathrm{Set}}_{H,\rho'})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}} f} \circ (\mu T^{\mathrm{Set}}_{H,f})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho} \end{array}$$

The latter equality holds because $\mu T^{\mathrm{Set}}_{H,\rho}$ *and* $\mu T^{\mathrm{Set}}_{H,\rho'}$ *are functors and* $\mu T^{\mathrm{Set}}_{H,f} : \mu T^{\mathrm{Set}}_{H,\rho} \to \mu T^{\mathrm{Set}}_{H,\rho'}$ *is a natural transformation, so the following naturality square commutes:*

$$\begin{array}{ccc} (\mu T^{\mathrm{Set}}_{H,\rho})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho} & \xrightarrow{(\mu T^{\mathrm{Set}}_{H,f})_{\overline{[\![\Gamma;\Phi,\overline{\gamma}\vdash G]\!]^{\mathrm{Set}}\rho}}} & (\mu T^{\mathrm{Set}}_{H,\rho'})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho} \\ {\scriptstyle (\mu T^{\mathrm{Set}}_{H,\rho})\overline{[\![\Gamma;\Phi,\overline{\gamma}\vdash G]\!]^{\mathrm{Set}} f}}\downarrow & & \downarrow{\scriptstyle (\mu T^{\mathrm{Set}}_{H,\rho'})\overline{[\![\Gamma;\Phi,\overline{\gamma}\vdash G]\!]^{\mathrm{Set}} f}} \\ (\mu T^{\mathrm{Set}}_{H,\rho})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho'} & \xrightarrow{(\mu T^{\mathrm{Set}}_{H,f})_{\overline{[\![\Gamma;\Phi,\overline{\gamma}\vdash G]\!]^{\mathrm{Set}}\rho'}}} & (\mu T^{\mathrm{Set}}_{H,\rho'})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathrm{Set}}\rho'} \end{array} \tag{2}$$

Definitions 7 and 8 respect weakening, i.e., ensure that a type and its weakenings have the same set interpretations.

3.2 Interpreting Types as Relations

Definition 9. *A* k*-ary relation transformer* F *is a triple* (F^1, F^2, F^*)*, where* $F^1, F^2 : [\mathrm{Set}^k, \mathrm{Set}]$ *are functors,* $F^* : [\mathrm{Rel}^k, \mathrm{Rel}]$ *is a functor, if* $R_1 : \mathrm{Rel}(A_1, B_1), \ldots, R_k : \mathrm{Rel}(A_k, B_k)$*, then* $F^*\overline{R} : \mathrm{Rel}(F^1\overline{A}, F^2\overline{B})$*, and if* $(\alpha_1, \beta_1) \in \mathrm{Hom}_{\mathrm{Rel}}(R_1, S_1), \ldots, (\alpha_k, \beta_k) \in \mathrm{Hom}_{\mathrm{Rel}}(R_k, S_k)$ *then* $F^*\overline{(\alpha, \beta)} = (F^1\overline{\alpha}, F^2\overline{\beta})$*. We define* $F\overline{R}$ *to be* $F^*\overline{R}$ *and* $F\overline{(\alpha, \beta)}$ *to be* $F^*\overline{(\alpha, \beta)}$*.*

The last clause of Definition 9 expands to: if $\overline{(a, b) \in R}$ implies $\overline{(\alpha\, a, \beta\, b) \in S}$ then $(c, d) \in F^*\overline{R}$ implies $(F^1\overline{\alpha}\, c, F^2\overline{\beta}\, d) \in F^*\overline{S}$. When convenient we identify a 0-ary relation transformer (A, B, R) with $R : \mathrm{Rel}(A, B)$. We may also write $\pi_1 F$ for F^1 and $\pi_2 F$ for F^2. We extend these conventions to relation environments, introduced in Definition 14 below, in the obvious way.

Definition 10. *The category* RT_k *of* k*-ary relation transformers is given by the following data:*

- *An object of* RT_k *is a relation transformer.*

- *A morphism* $\delta : (G^1, G^2, G^*) \to (H^1, H^2, H^*)$ *in* RT_k *is a pair of natural transformations* (δ^1, δ^2) *where* $\delta^1 : G^1 \to H^1$, $\delta^2 : G^2 \to H^2$ *such that, for all* $\overline{R : \mathsf{Rel}(A, B)}$, *if* $(x, y) \in G^*\overline{R}$ *then* $(\delta^1_{\overline{A}} x, \delta^2_{\overline{B}} y) \in H^*\overline{R}$.
- *Identity morphisms and composition are inherited from the category of functors on* Set.

Definition 11. *An endofunctor* H *on* RT_k *is a triple* $H = (H^1, H^2, H^*)$, *where*

- H^1 *and* H^2 *are functors from* $[\mathsf{Set}^k, \mathsf{Set}]$ *to* $[\mathsf{Set}^k, \mathsf{Set}]$
- H^* *is a functor from* RT_k *to* $[\mathsf{Rel}^k, \mathsf{Rel}]$
- *for all* $\overline{R : \mathsf{Rel}(A, B)}$, $\pi_1((H^*(\delta^1, \delta^2))_{\overline{R}}) = (H^1\delta^1)_{\overline{A}}$ *and* $\pi_2((H^*(\delta^1, \delta^2))_{\overline{R}}) = (H^2\delta^2)_{\overline{B}}$
- *The action of* H *on objects is given by* $H(F^1, F^2, F^*) = (H^1F^1,\ H^2F^2,\ H^*(F^1, F^2, F^*))$
- *The action of* H *on morphisms is given by* $H(\delta^1, \delta^2) = (H^1\delta^1, H^2\delta^2)$ *for* $(\delta^1, \delta^2) : (F^1, F^2, F^*) \to (G^1, G^2, G^*)$

Since the results of applying an endofunctor H to k-ary relation transformers and morphisms between them must again be k-ary relation transformers and morphisms between them, respectively, Definition 11 implicitly requires that the following three conditions hold: *i*) if $R_1 : \mathsf{Rel}(A_1, B_1), \ldots, R_k : \mathsf{Rel}(A_k, B_k)$, then $H^*(F^1, F^2, F^*)\overline{R} : \mathsf{Rel}(H^1F^1\overline{A}, H^2F^2\overline{B})$; *ii*) if $(\alpha_1, \beta_1) \in \mathsf{Hom}_{\mathsf{Rel}}(R_1, S_1), \ldots, (\alpha_k, \beta_k) \in \mathsf{Hom}_{\mathsf{Rel}}(R_k, S_k)$, then $H^*(F^1, F^2, F^*)\,\overline{(\alpha, \beta)} = (H^1F^1\overline{\alpha}, H^2F^2\overline{\beta})$; and $(\delta^1, \delta^2) : (F^1, F^2, F^*) \to (G^1, G^2, G^*)$ and $R_1 : \mathsf{Rel}(A_1, B_1), \ldots, R_k : \mathsf{Rel}(A_k, B_k)$, then $((H^1\delta^1)_{\overline{A}} x, (H^2\delta^2)_{\overline{B}} y) \in H^*(G^1, G^2, G^*)\overline{R}$ whenever $(x, y) \in H^*(F^1, F^2, F^*)\overline{R}$. Note, however, that this last condition is automatically satisfied because it is implied by the third bullet point of Definition 11.

Definition 12. *If* H *and* K *are endofunctors on* RT_k, *then a* natural transformation $\sigma : H \to K$ *is a pair* $\sigma = (\sigma^1, \sigma^2)$, *where* $\sigma^1 : H^1 \to K^1$ *and* $\sigma^2 : H^2 \to K^2$ *are natural transformations between endofunctors on* $[\mathsf{Set}^k, \mathsf{Set}]$ *and the component of* σ *at* $F \in RT_k$ *is given by* $\sigma_F = (\sigma^1_{F^1}, \sigma^2_{F^2})$.

Definition 12 entails that $\sigma^i_{F^i}$ must be natural in $F^i : [\mathsf{Set}^k, \mathsf{Set}]$, and, for every F, both $(\sigma^1_{F^1})_{\overline{A}}$ and $(\sigma^2_{F^2})_{\overline{A}}$ must be natural in $\overline{A}$. Moreover, since the results of applying σ to k-ary relation transformers must be morphisms of k-ary relation transformers, Definition 12 implicitly requires that $(\sigma_F)_{\overline{R}} = ((\sigma^1_{F^1})_{\overline{A}}, (\sigma^2_{F^2})_{\overline{B}})$ is a morphism in Rel for any k-tuple of relations $\overline{R : \mathsf{Rel}(A, B)}$, i.e., that if $(x, y) \in H^*F\overline{R}$, then $((\sigma^1_{F^1})_{\overline{A}} x, (\sigma^2_{F^2})_{\overline{B}} y) \in K^*F\overline{R}$.

Critically, we can compute ω-directed colimits in RT_k: it is not hard to see that if $\mathcal{D}$ is an ω-directed set then $\varinjlim_{d \in \mathcal{D}} (F^1_d, F^2_d, F^*_d) = (\varinjlim_{d \in \mathcal{D}} F^1_d, \varinjlim_{d \in \mathcal{D}} F^2_d, \varinjlim_{d \in \mathcal{D}} F^*_d)$. We define an endofunctor $T = (T^1, T^2, T^*)$ on RT_k to be *ω-cocontinuous* if T^1 and T^2 are ω-cocontinuous endofunctors on $[\mathsf{Set}^k, \mathsf{Set}]$ and T^* is an ω-cocontinuous functor from RT_k to $[\mathsf{Rel}^k, \mathsf{Rel}]$, i.e., is in $[RT_k, [\mathsf{Rel}^k, \mathsf{Rel}]]$. Now, for any k, any $A : \mathsf{Set}$, and any $R : \mathsf{Rel}(A, B)$, let K^{Set}_A be the constantly A-valued functor from Set^k to Set and K^{Rel}_R be the constantly R-valued functor from Rel^k to Rel. Also let 0 denote either the initial object of Set or the initial object of Rel, as appropriate. Observing that, for every k, K^{Set}_0 is initial in $[\mathsf{Set}^k, \mathsf{Set}]$, and K^{Rel}_0 is initial in $[\mathsf{Rel}^k, \mathsf{Rel}]$, we have that, for each k, $K_0 = (K^{\mathsf{Set}}_0, K^{\mathsf{Set}}_0, K^{\mathsf{Rel}}_0)$ is initial in RT_k. Thus, if $T = (T^1, T^2, T^*) : RT_k \to RT_k$ is an endofunctor on RT_k then we can define the relation transformer μT to be $\varinjlim_{n \in \mathbb{N}} T^n K_0$. It is not hard to see that μT is given explicitly as

$$\mu T = (\mu T^1, \mu T^2, \varinjlim_{n \in \mathbb{N}} (T^n K_0)^*) \tag{3}$$

and that, as our notation suggests, it really is a fixpoint for T if T is ω-cocontinuous:

Lemma 13. *For any* $T : [RT_k, RT_k]$, $\mu T \cong T(\mu T)$.

The isomorphism is given by the morphisms $(in_1, in_2) : T(\mu T) \to \mu T$ and $(in_1^{-1}, in_2^{-1}) : \mu T \to T(\mu T)$ in RT_k. The latter is always a morphism in RT_k, but the former need not be if T is not ω-cocontinuous.

It is worth noting that the third component in Equation (3) is the colimit in $[\mathsf{Rel}^k, \mathsf{Rel}]$ of third components of relation transformers, rather than a fixpoint of an endofunctor on $[\mathsf{Rel}^k, \mathsf{Rel}]$. That there is an asymmetry between the first two components of μT and its third reflects the important conceptual observation that the third component of an endofunctor on RT_k need not be a functor on all of $[\mathsf{Rel}^k, \mathsf{Rel}]$. In particular, although we can define $T_{H,\rho}\, F$ for a relation transformer F in Definition 16 below, it is not clear how we could define it for $F : [\mathsf{Rel}^k, \mathsf{Rel}]$.

Definition 14. *A* relation environment *maps each type variable in $\mathbb{T}^k \cup \mathbb{F}^k$ to a k-ary relation transformer. A morphism $f : \rho \to \rho'$ between relation environments ρ and ρ' with $\rho|_{\mathbb{T}} = \rho'|_{\mathbb{T}}$ maps each type constructor variable $\psi^k \in \mathbb{T}$ to the identity morphism on $\rho\psi^k = \rho'\psi^k$ and each functorial variable $\phi^k \in \mathbb{F}$ to a morphism from the k-ary relation transformer $\rho\phi$ to the k-ary relation transformer $\rho'\phi$. Composition of morphisms on relation environments is given componentwise, with the identity morphism mapping each relation environment to itself. This gives a category of relation environments and morphisms between them, which we denote* RelEnv.

When convenient we identify a 0-ary relation transformer with the relation (transformer) that is its codomain and consider a relation environment to map a type variable of arity 0 to a relation. We write $\rho[\overline{\alpha := R}]$ for the relation environment ρ' such that $\rho'\alpha_i = R_i$ for $i = 1, \ldots, k$ and $\rho'\alpha = \rho\alpha$ if $\alpha \notin \{\alpha_1, \ldots, \alpha_k\}$. If ρ is a relation environment, we write $\pi_1\rho$ and $\pi_2\rho$ for the set environments mapping each type variable ϕ to the functors $(\rho\phi)^1$ and $(\rho\phi)^2$, respectively.

We define, for each k, the notion of an ω-cocontinuous functor from RelEnv to RT_k:

Definition 15. *A functor $H : [\mathsf{RelEnv}, RT_k]$ is a triple $H = (H^1, H^2, H^*)$, where*

- *H^1 and H^2 are objects in $[\mathsf{SetEnv}, [\mathsf{Set}^k, \mathsf{Set}]]$*
- *H^* is a an object in $[\mathsf{RelEnv}, [\mathsf{Rel}^k, \mathsf{Rel}]]$*
- *for all $\overline{R : \mathsf{Rel}(A, B)}$ and morphisms f in RelEnv, $\pi_1(H^* f\, \overline{R}) = H^1(\pi_1 f)\, \overline{A}$ and $\pi_2(H^* f\, \overline{R}) = H^2(\pi_2 f)\, \overline{B}$*
- *The action of H on ρ in RelEnv is given by $H\rho = (H^1(\pi_1\rho),\ H^2(\pi_2\rho),\ H^*\rho)$*
- *The action of H on morphisms $f : \rho \to \rho'$ in RelEnv is given by $Hf = (H^1(\pi_1 f), H^2(\pi_2 f))$*

Spelling out the last two bullet points above gives the following analogues of the three conditions immediately following Definition 11: *i*) if $R_1 : \mathsf{Rel}(A_1, B_1), \ldots, R_k : \mathsf{Rel}(A_k, B_k)$, then $H^*\rho\, \overline{R} : \mathsf{Rel}(H^1(\pi_1\rho)\, \overline{A}, H^2(\pi_2\rho)\, \overline{B})$; *ii*) if $(\alpha_1, \beta_1) \in \mathsf{Hom}_{\mathsf{Rel}}(R_1, S_1), \ldots, (\alpha_k, \beta_k) \in \mathsf{Hom}_{\mathsf{Rel}}(R_k, S_k)$, then $H^*\rho\, \overline{(\alpha, \beta)} = (H^1(\pi_1\rho)\, \overline{\alpha}, H^2(\pi_2\rho)\, \overline{\beta})$; and *iii*) if $f : \rho \to \rho'$ and $R_1 : \mathsf{Rel}(A_1, B_1), \ldots, R_k : \mathsf{Rel}(A_k, B_k)$, then $(H^1(\pi_1 f)\, \overline{A}\, x, H^2(\pi_2 f)\, \overline{B}\, y) \in H^*\rho'\, \overline{R}$ whenever $(x, y) \in H^*\rho\, \overline{R}$. As before, the last condition is automatically satisfied because it is implied by the third bullet point of Definition 15.

Considering RelEnv as a product $\Pi_{\phi^k \in \mathbb{T} \cup \mathbb{F}} RT_k$, we extend the computation of ω-directed colimits in RT_k to compute colimits in RelEnv componentwise. We similarly extend the notion of an ω-cocontinuous endofunctor on RT_k componentwise to give a notion of ω-cocontinuity for functors from RelEnv to RT_k. Recalling from the start of this subsection that Definition 16 is given mutually inductively with Definition 7 we can, at last, define:

Definition 16. *The* relational interpretation $[\![\cdot]\!]^{\mathsf{Rel}} : \mathcal{F} \to [\mathsf{RelEnv}, \mathsf{Rel}]$ *is defined by*

$$[\![\Gamma; \Phi \vdash \mathbb{0}]\!]^{\mathsf{Rel}}\rho = 0$$

$$[\![\Gamma; \Phi \vdash \mathbb{1}]\!]^{\mathsf{Rel}}\rho = 1$$

$$\begin{aligned}
[\![\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathsf{Rel}}\rho &= \{\eta : \lambda\overline{R}.\,[\![\Gamma;\overline{\alpha} \vdash F]\!]^{\mathsf{Rel}}\rho[\overline{\alpha := R}] \Rightarrow \lambda\overline{R}.\,[\![\Gamma;\overline{\alpha} \vdash G]\!]^{\mathsf{Rel}}\rho[\overline{\alpha := R}]\} \\
&= \{(t,t') \in [\![\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathsf{Set}}(\pi_1\rho) \times [\![\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathsf{Set}}(\pi_2\rho) \mid \\
&\qquad \forall R_1 : \mathsf{Rel}(A_1,B_1) \ldots R_k : \mathsf{Rel}(A_k,B_k). \\
&\qquad\quad (t_{\overline{A}}, t'_{\overline{B}}) \in ([\![\Gamma;\overline{\alpha} \vdash G]\!]^{\mathsf{Rel}}\rho[\overline{\alpha := R}])^{[\![\Gamma;\overline{\alpha}\vdash F]\!]^{\mathsf{Rel}}\rho[\overline{\alpha:=R}]}\} \\
[\![\Gamma;\Phi \vdash \phi\overline{F}]\!]^{\mathsf{Rel}}\rho &= (\rho\phi)\overline{[\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}\rho} \\
[\![\Gamma;\Phi \vdash F+G]\!]^{\mathsf{Rel}}\rho &= [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}\rho + [\![\Gamma;\Phi \vdash G]\!]^{\mathsf{Rel}}\rho \\
[\![\Gamma;\Phi \vdash F\times G]\!]^{\mathsf{Rel}}\rho &= [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}\rho \times [\![\Gamma;\Phi \vdash G]\!]^{\mathsf{Rel}}\rho \\
[\![\Gamma;\Phi,\overline{\gamma} \vdash (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}]\!]^{\mathsf{Rel}}\rho &= (\mu T_{H,\rho})\overline{[\![\Gamma;\Phi,\overline{\gamma} \vdash G]\!]^{\mathsf{Rel}}\rho} \\
\textit{where } T_{H,\rho} &= (T^{\mathsf{Set}}_{H,\pi_1\rho}, T^{\mathsf{Set}}_{H,\pi_2\rho}, T^{\mathsf{Rel}}_{H,\rho}) \\
\textit{and } T^{\mathsf{Rel}}_{H,\rho}\, F &= \lambda\overline{R}.[\![\Gamma;\overline{\gamma},\phi,\overline{\alpha} \vdash H]\!]^{\mathsf{Rel}}\rho[\phi := F][\overline{\alpha := R}] \\
\textit{and } T^{\mathsf{Rel}}_{H,\rho}\,\delta &= \lambda\overline{R}.[\![\Gamma;\overline{\gamma},\phi,\overline{\alpha} \vdash H]\!]^{\mathsf{Rel}}id_\rho[\phi := \delta][\overline{\alpha := id_{\overline{R}}}]
\end{aligned}$$

The interpretations in Definitions 16 and 17 below respect weakening, and also ensure that $[\![\Gamma \vdash F \to G]\!]^{\mathsf{Rel}}$ and $[\![\Gamma \vdash \forall\overline{\alpha}.F]\!]^{\mathsf{Rel}}$ are as expected in any parametric model. As for set interpretations, $[\![\Gamma;\emptyset \vdash \mathsf{Nat}^{\overline{\alpha}} F\, G]\!]^{\mathsf{Rel}}$ is ω-cocontinuous because it is a constant functor. If $\rho \in \mathsf{RelEnv}$ and $\vdash F$, then we write $[\![\vdash F]\!]^{\mathsf{Rel}}$ instead of $[\![\vdash F]\!]^{\mathsf{Rel}}\rho$. For the last clause in Definition 16 to be well-defined we need $T_{H,\rho}$ to be an ω-cocontinuous endofunctor on RT so that, by Lemma 13, it admits a fixpoint. Since $T_{H,\rho}$ is defined in terms of $[\![\Gamma;\overline{\gamma},\phi^k,\overline{\alpha} \vdash H]\!]^{\mathsf{Rel}}$, this means that relational interpretations of types must be ω-cocontinuous functors from RelEnv to RT_0, which in turn entails that the actions of relational interpretations of types on objects and on morphisms in RelEnv are intertwined. As for set interpretations, we know from [Johann and Polonsky 2019] that, for every $\Gamma;\overline{\alpha} \vdash F$, $[\![\Gamma;\overline{\alpha} \vdash F]\!]^{\mathsf{Rel}}$ is actually in $[\mathsf{Rel}^k, \mathsf{Rel}]$ where $k = |\overline{\alpha}|$. We first define the actions of each of these functors on morphisms between environments in Definition 17, and then argue that the functors given by Definitions 16 and 17 are well-defined and have the required properties. To do this, we extend T_H to a *functor* from RelEnv to $[[\mathsf{Rel}^k, \mathsf{Rel}], [\mathsf{Rel}^k, \mathsf{Rel}]]$. Its action on an object $\rho \in \mathsf{RelEnv}$ is given by the higher-order functor $T_{H,\rho}$ whose actions on objects and morphisms are given in Definition 17. Its action on a morphism $f : \rho \to \rho'$ is the higher-order natural transformation $T_{H,f} : T_{H,\rho} \to T_{H,\rho'}$ whose action on any $F : [\mathsf{Rel}^k, \mathsf{Rel}]$ is the natural transformation $T_{H,f}\, F : T_{H,\rho}\, F \to T_{H,\rho'}\, F$ whose component at $\overline{R}$ is $(T_{H,f}\, F)_{\overline{R}} = [\![\Gamma;\overline{\gamma},\phi,\overline{\alpha} \vdash H]\!]^{\mathsf{Rel}} f[\phi := id_F][\overline{\alpha := id_R}]$. The next definition uses T_H to define the actions of functors interpreting types on morphisms between relation environments.

Definition 17. *Let $f : \rho \to \rho'$ for relation environments ρ and ρ' (so that $\rho|_{\mathbb{T}} = \rho'|_{\mathbb{T}}$). The action $[\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}} f$ of $[\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}$ on the morphism f is given exactly as in Definition 8, except that all interpretations are relational interpretations and all occurrences of $T^{\mathsf{Set}}_{H,f}$ are replaced by $T_{H,f}$.*

To see that the functors given by Definitions 16 and 17 are well-defined we must show that, for every H, $T_{H,\rho}\, F$ is a relation transformer for any relation transformer F, and that $T_{H,f}\, F : T_{H,\rho}\, F \to T_{H,\rho'}\, F$ is a morphism of relation transformers for every relation transformer F and every morphism $f : \rho \to \rho'$ in RelEnv. This is an immediate consequence of

Lemma 18. *For every $\Gamma;\Phi \vdash F$, $[\![\Gamma;\Phi \vdash F]\!] = ([\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Set}}, [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Set}}, [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}) \in [\mathsf{RelEnv}, RT_0]$.*

The proof is a straightforward induction on the structure of F, using an appropriate result from [Johann and Polonsky 2019] to deduce ω-cocontinuity of $[\![\Gamma;\Phi \vdash F]\!]$ in each case, together with Lemma 13 and Equation 3 for μ-types.

We can also prove by simultaneous induction that our interpretations of types interact well with demotion of functorial variables. Indeed, we have that, if $\rho, \rho' : \mathsf{SetEnv}$, $f : \rho \to \rho'$, $\rho\phi = \rho\psi = \rho'\phi = \rho'\psi$, $f\phi = f\psi = id_{\rho\phi}$, $\Gamma; \Phi, \phi^k \vdash F$, $\Gamma; \Phi, \overline{\alpha} \vdash G$, $\Gamma; \Phi, \alpha_1 ... \alpha_k \vdash H$, and $\overline{\Gamma}; \overline{\Phi} \vdash \overline{K}$, then

$$[\![\Gamma; \Phi, \phi \vdash F]\!]^{\mathrm{Set}}\rho = [\![\Gamma, \psi; \Phi \vdash F[\phi :== \psi]]\!]^{\mathrm{Set}}\rho \tag{4}$$

$$[\![\Gamma; \Phi, \phi \vdash F]\!]^{\mathrm{Set}}f = [\![\Gamma, \psi; \Phi \vdash F[\phi :== \psi]]\!]^{\mathrm{Set}}f \tag{5}$$

$$[\![\Gamma; \Phi \vdash G[\overline{\alpha := K}]]\!]^{\mathrm{Set}}\rho = [\![\Gamma; \Phi, \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}\rho[\overline{\alpha := [\![\Gamma; \Phi \vdash K]\!]^{\mathrm{Set}}\rho}] \tag{6}$$

$$[\![\Gamma; \Phi \vdash G[\overline{\alpha := K}]]\!]^{\mathrm{Set}}f = [\![\Gamma; \Phi, \overline{\alpha} \vdash G]\!]^{\mathrm{Set}}f[\overline{\alpha := [\![\Gamma; \Phi \vdash K]\!]^{\mathrm{Set}}f}] \tag{7}$$

$$[\![\Gamma; \Phi \vdash F[\phi := H]]\!]^{\mathrm{Set}}\rho = [\![\Gamma; \Phi, \phi \vdash F]\!]^{\mathrm{Set}}\rho[\phi := \lambda\overline{A}.\, [\![\Gamma; \Phi, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}}\rho[\overline{\alpha := A}]] \tag{8}$$

$$[\![\Gamma; \Phi \vdash F[\phi := H]]\!]^{\mathrm{Set}}f = [\![\Gamma; \Phi, \phi \vdash F]\!]^{\mathrm{Set}}f[\phi := \lambda\overline{A}.\, [\![\Gamma; \Phi, \overline{\alpha} \vdash H]\!]^{\mathrm{Set}}f[\overline{\alpha := id_{\overline{A}}}]] \tag{9}$$

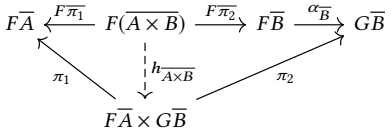
Identities analogous to (4) through (9) hold for relational interpretations as well.

4 THE IDENTITY EXTENSION LEMMA

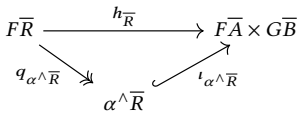
In most treatments of parametricity, equality relations on sets are taken as *given* — either directly as diagonal relations, or perhaps via reflexive graphs if kinds are also being tracked — and the graph relations used to validate existence of initial algebras are defined in terms of them. We take a different approach here, giving a categorical definition of graph relations for morphisms (i.e., natural transformations) between functors and *constructing* equality relations as particular graph relations. Our definitions specialize to the usual ones for the graph relation for a morphism between sets and the equality relation for a set. In light of its novelty, we spell out our construction in detail.

The standard definition of the graph for a morphism $f : A \to B$ in Set is the relation $\langle f \rangle : \mathsf{Rel}(A, B)$ defined by $(x, y) \in \langle f \rangle$ iff $fx = y$. This definition naturally generalizes to associate to each natural transformation between k-ary functors on Set a k-ary relation transformer as follows:

Definition 19. *If $F, G : \mathrm{Set}^k \to \mathrm{Set}$ and $\alpha : F \to G$ is a natural transformation, then the functor $\langle\alpha\rangle^* : \mathsf{Rel}^k \to \mathsf{Rel}$ is defined as follows. Given $R_1 : \mathsf{Rel}(A_1, B_1), ..., R_k : \mathsf{Rel}(A_k, B_k)$, let $\iota_{R_i} : R_i \hookrightarrow A_i \times B_i$, for $i = 1, ..., k$, be the inclusion of R_i as a subset of $A_i \times B_i$, let $h_{\overline{A \times B}}$ be the unique morphism making the diagram*



commute, and let $h_{\overline{R}} : F\overline{R} \to F\overline{A} \times G\overline{B}$ be $h_{\overline{A \times B}} \circ F\overline{\iota_R}$. Further, let $\alpha^\wedge\overline{R}$ be the subobject through which $h_{\overline{R}}$ is factorized by the mono-epi factorization system in Set*, as shown in the following diagram:*

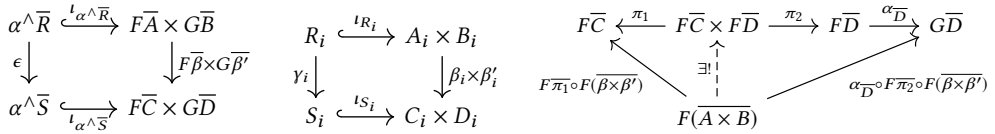


Then $\alpha^\wedge\overline{R} : \mathsf{Rel}(F\overline{A}, G\overline{B})$ by construction, so the action of $\langle\alpha\rangle^*$ on objects can be given by $\langle\alpha\rangle^*(\overline{A, B, R}) = (F\overline{A}, G\overline{B}, \iota_{\alpha^\wedge\overline{R}}\alpha^\wedge\overline{R})$. Its action on morphisms is given by $\langle\alpha\rangle^*(\overline{\beta, \beta'}) = (F\overline{\beta}, G\overline{\beta}')$.

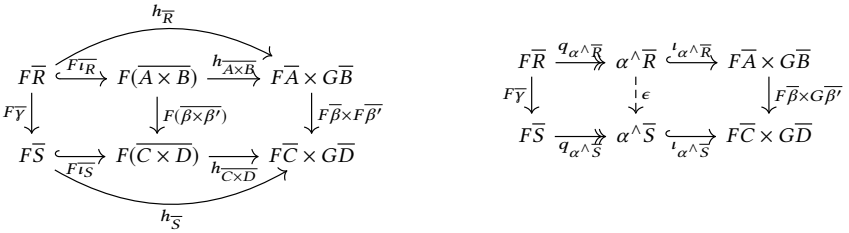
The data in Definition 19 yield the *graph relation transformer for α*, denoted $\langle\alpha\rangle = (F, G, \langle\alpha\rangle^*)$.

Lemma 20. *If $F, G : [\mathrm{Set}^k, \mathrm{Set}]$, and if $\alpha : F \to G$ is a natural transformation, then $\langle\alpha\rangle$ is in RT_k.*

Proof. Clearly, $\langle\alpha\rangle^*$ is ω-cocontinuous, so $\langle\alpha\rangle^* : [\mathrm{Rel}^k, \mathrm{Rel}]$. Now, let $\overline{R : \mathrm{Rel}(A,B)}$, $\overline{S : \mathrm{Rel}(C,D)}$, and $\overline{(\beta,\beta') : R \to S}$. We want to show that there exists a morphism $\epsilon : \alpha^\wedge\overline{R} \to \alpha^\wedge\overline{S}$ such that the diagram on the left below commutes. Since $\overline{(\beta,\beta') : R \to S}$, there exist $\overline{\gamma : R \to S}$ such that each diagram in the middle commutes. Moreover, since both $h_{\overline{C\times D}} \circ F(\overline{\beta\times\beta'})$ and $(F\overline{\beta}\times G\overline{\beta'}) \circ h_{\overline{A\times B}}$ make the diagram on the right commute, they must be equal.



We therefore get that the right-hand square in the diagram on the left below commutes, and thus that the entire diagram does as well. Finally, by the left-lifting property of $q_{F^\wedge\overline{R}}$ with respect to $\iota_{F^\wedge\overline{S}}$ given by the epi-mono factorization system, there exists an ϵ such that the diagram on the right below commutes.



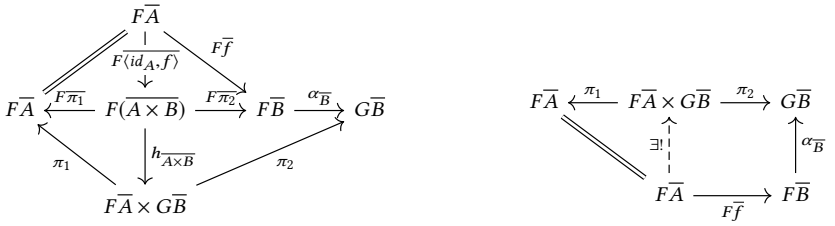
□

If $f : A \to B$ is a morphism in Set then the definition of the graph relation transformer $\langle f\rangle$ for f as a natural transformation between 0-ary functors A and B coincides with its standard definition. Graph relation transformers are thus a reasonable extension of graph relations to functors.

To prove the IEL, we will need to know that the equality relation transformer preserves equality relations; see Equation 10 below. This will follow from the next lemma, which shows how to compute the action of a graph relation transformer on any graph relation.

Lemma 21. *If $\alpha : F \to G$ is a morphism in $[\mathrm{Set}^k, \mathrm{Set}]$ and $f_1 : A_1 \to B_1, \ldots, f_k : A_k \to B_k$, then $\langle\alpha\rangle^*\langle\overline{f}\rangle = \langle G\overline{f} \circ \alpha_{\overline{A}}\rangle = \langle\alpha_{\overline{B}} \circ F\overline{f}\rangle$.*

Proof. Since $h_{\overline{A\times B}}$ is the unique morphism making the bottom triangle of the diagram on the left below commute, and since $h_{\langle\overline{f}\rangle} = h_{\overline{A\times B}} \circ F\,\overline{\iota_{\langle f\rangle}} = h_{\overline{A\times B}} \circ F\overline{\langle id_A, f\rangle}$, the universal property of the product depicted in the diagram on the right gives $h_{\langle\overline{f}\rangle} = \langle id_{F\overline{A}}, \alpha_{\overline{B}} \circ F\overline{f}\rangle : F\overline{A} \to F\overline{A}\times G\overline{B}$.



Moreover, $\langle id_{F\overline{A}}, \alpha_{\overline{B}} \circ F\overline{f}\rangle$ is a monomorphism in Set because $id_{F\overline{A}}$ is, so its epi-mono factorization gives $\iota_{\alpha^\wedge\langle\overline{f}\rangle} = \langle id_{F\overline{A}}, \alpha_{\overline{B}} \circ F\overline{f}\rangle$, and thus $\alpha^\wedge\langle\overline{f}\rangle = F\overline{A}$. Then $\iota_{\alpha^\wedge\langle\overline{f}\rangle}\alpha^\wedge\langle\overline{f}\rangle = \langle id_{F\overline{A}}, \alpha_{\overline{B}} \circ F\overline{f}\rangle(F\overline{A}) = \langle\alpha_{\overline{B}} \circ F\overline{f}\rangle^*$, so that $\langle\alpha\rangle^*\langle\overline{f}\rangle = (F\overline{A}, G\overline{B}, \iota_{\alpha^\wedge\langle\overline{f}\rangle}\alpha^\wedge\langle\overline{f}\rangle) = (F\overline{A}, G\overline{B}, \langle\alpha_{\overline{B}} \circ F\overline{f}\rangle^*) = \langle\alpha_{\overline{B}} \circ F\overline{f}\rangle$. Finally, $\alpha_{\overline{B}} \circ F\overline{f} = G\overline{f} \circ \alpha_{\overline{A}}$ by naturality of α. □

The *equality relation transformer* on $F : [\mathsf{Set}^k, \mathsf{Set}]$ is defined to be $\mathsf{Eq}_F = \langle id_F \rangle$. Specifically, $\mathsf{Eq}_F = (F, F, \mathsf{Eq}_F^*)$ with $\mathsf{Eq}_F^* = \langle id_F \rangle^*$, and Lemma 21 indeed ensures that

$$\mathsf{Eq}_F^* \overline{\mathsf{Eq}_A} = \langle id_F \rangle^* \langle id_{\overline{A}} \rangle = \langle F id_{\overline{A}} \circ (id_F)_{\overline{A}} \rangle = \langle id_{F\overline{A}} \circ id_{F\overline{A}} \rangle = \langle id_{F\overline{A}} \rangle = \mathsf{Eq}_{F\overline{A}} \tag{10}$$

for all $\overline{A : \mathsf{Set}}$. Graph relation transformers in general, and equality relation transformers in particular, extend to relation environments in the obvious ways. Indeed, if $\rho, \rho' : \mathsf{SetEnv}$ and $f : \rho \to \rho'$, then the *graph relation environment* $\langle f \rangle$ is defined pointwise by $\langle f \rangle \phi = \langle f \phi \rangle$ for every ϕ, which entails that $\pi_1 \langle f \rangle = \rho$ and $\pi_2 \langle f \rangle = \rho'$. In particular, the *equality relation environment* Eq_ρ is defined to be $\langle id_\rho \rangle$, which entails that $\mathsf{Eq}_\rho \phi = \mathsf{Eq}_{\rho\phi}$ for every ϕ. With these definitions in hand, we can state and prove both an Identity Extension Lemma and a Graph Lemma for our calculus.

Theorem 22 (IEL). *If* $\rho : \mathsf{SetEnv}$ *and* $\Gamma; \Phi \vdash F$ *then* $[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \mathsf{Eq}_\rho = \mathsf{Eq}_{[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho}$.

The proof is by induction on the structure of F. Only the Nat, application, and fixpoint cases are non-routine. The latter two use Lemma 21. The fixpoint case also uses the observation that, for every $n \in \mathbb{N}$, the following intermediate results can be proved by simultaneous induction with Theorem 22: for any H, ρ, A, and any subformula J of H, both $T^n_{H, \mathsf{Eq}_\rho} K_0 \, \overline{\mathsf{Eq}_A} = (\mathsf{Eq}_{(T^{\mathsf{Set}}_{H,\rho})^n K_0})^* \overline{\mathsf{Eq}_A}$ and

$$\begin{aligned} & [\![\Gamma; \Phi, \phi, \overline{\alpha} \vdash J]\!]^{\mathsf{Rel}} \mathsf{Eq}_\rho [\phi := T^n_{H, \mathsf{Eq}_\rho} K_0] \overline{[\alpha := \mathsf{Eq}_A]} \\ = \quad & [\![\Gamma; \Phi, \phi, \overline{\alpha} \vdash J]\!]^{\mathsf{Rel}} \mathsf{Eq}_\rho [\phi := \mathsf{Eq}_{(T^{\mathsf{Set}}_{H,\rho})^n K_0}] \overline{[\alpha := \mathsf{Eq}_A]} \end{aligned}$$

hold. With these results in hand, the proof follows easily. It is given in detail in the accompanying anonymous supplementary material. As noted there, if functorial variables of arity greater than 0 were allowed to appear in the bodies of μ-types, then the IEL would fail.

Lemma 23 (Graph Lemma). *If* $\rho, \rho' : \mathsf{SetEnv}$, $f : \rho \to \rho'$, *and* $\Gamma; \Phi \vdash F$, *then* $\langle [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \rangle = [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$.

Proof. Applying Lemma 18 to the morphisms $(f, id_{\rho'}) : \langle f \rangle \to \mathsf{Eq}_{\rho'}$ and $(id_\rho, f) : \mathsf{Eq}_\rho \to \langle f \rangle$ of relation environments gives that $([\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} id_{\rho'}) = [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} (f, id_{\rho'}) : [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle \to [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \mathsf{Eq}_{\rho'}$ and $([\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} id_\rho, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f) = [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} (id_\rho, f) : [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \mathsf{Eq}_\rho \to [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$. Expanding the first equation gives that if $(x, y) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$ then $([\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} id_{\rho'} \, y) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \mathsf{Eq}_{\rho'}$. So $[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} id_{\rho'} \, y = id_{[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho'} \, y = y$ and $[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \mathsf{Eq}_{\rho'} = \mathsf{Eq}_{[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho'}$, and if $(x, y) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$ then $([\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x, y) \in \mathsf{Eq}_{[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho'}$, i.e., $[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x = y$, i.e., $(x, y) \in \langle [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \rangle$. So, we have that $[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle \subseteq \langle [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \rangle$. Expanding the second equation gives that if $x \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho$ then $([\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} id_\rho \, x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$. Then $[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} id_\rho \, x = id_{[\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho} x = x$, so for any $x \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho$ we have that $(x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$. Moreover, $x \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho$ if and only if $(x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x) \in \langle [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \rangle$ and, if $x \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} \rho$ then $(x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$, so if $(x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x) \in \langle [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \rangle$ then $(x, [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \, x) \in [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$, i.e., $\langle [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Set}} f \rangle \subseteq [\![\Gamma; \Phi \vdash F]\!]^{\mathsf{Rel}} \langle f \rangle$. □

5 INTERPRETING TERMS

If $\Delta = x_1 : F_1, \ldots, x_n : F_n$ is a term context for Γ and Φ, then the interpretations $[\![\Gamma; \Phi \vdash \Delta]\!]^{\mathsf{Set}}$ and $[\![\Gamma; \Phi \vdash \Delta]\!]^{\mathsf{Rel}}$ are defined by

$$\begin{aligned} [\![\Gamma; \Phi \vdash \Delta]\!]^{\mathsf{Set}} &= [\![\Gamma; \Phi \vdash F_1]\!]^{\mathsf{Set}} \times \ldots \times [\![\Gamma; \Phi \vdash F_n]\!]^{\mathsf{Set}} \\ [\![\Gamma; \Phi \vdash \Delta]\!]^{\mathsf{Rel}} &= [\![\Gamma; \Phi \vdash F_1]\!]^{\mathsf{Rel}} \times \ldots \times [\![\Gamma; \Phi \vdash F_n]\!]^{\mathsf{Rel}} \end{aligned}$$

$$
\begin{array}{lcl}
[\![\Gamma;\Phi \mid \Delta, x : F \vdash x : F]\!]^{\mathsf{D}}\rho & = & \pi_{|\Delta|+1} \\
[\![\Gamma;\emptyset \mid \Delta \vdash L_{\overline{\alpha}} x.t : \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathsf{D}}\rho & = & \mathrm{curry}([\![\Gamma;\overline{\alpha} \mid \Delta, x : F \vdash t : G]\!]^{\mathsf{D}}\rho[\overline{\alpha := \_}]) \\
[\![\Gamma;\Phi \mid \Delta \vdash t_{\overline{K}} s : G[\overline{\alpha := K}]]\!]^{\mathsf{D}}\rho & = & \mathrm{eval} \circ \langle \lambda d.\, ([\![\Gamma;\emptyset \mid \Delta \vdash t : \mathsf{Nat}^{\overline{\alpha}}\, F\, G]\!]^{\mathsf{D}}\rho\ d)_{\overline{[\![\Gamma;\Phi \vdash K]\!]^{\mathsf{D}}\rho}}, \\
 & & \quad [\![\Gamma;\Phi \mid \Delta \vdash s : F[\overline{\alpha := K}]]\!]^{\mathsf{D}}\rho \rangle \\
 & & \\
[\![\Gamma;\Phi \mid \Delta \vdash \bot_F t : F]\!]^{\mathsf{D}}\rho & = & !^{0}_{[\![\Gamma;\Phi \vdash F]\!]^{\mathsf{D}}\rho} \circ [\![\Gamma;\Phi \mid \Delta \vdash t : \mathbb{0}]\!]^{\mathsf{D}}\rho, \text{ where} \\
 & & \quad !^{0}_{[\![\Gamma;\Phi \vdash F]\!]^{\mathsf{D}}\rho} \text{ is the unique morphism from } 0 \\
 & & \quad \text{to } [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{D}}\rho \\
[\![\Gamma;\Phi \mid \Delta \vdash \top : \mathbb{1}]\!]^{\mathsf{D}}\rho & = & !^{[\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{D}}\rho}_{1}, \text{ where } !^{[\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{D}}\rho}_{1} \\
 & & \quad \text{is the unique morphism from } [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{D}}\rho \text{ to } 1 \\
[\![\Gamma;\Phi \mid \Delta \vdash (s,t) : F \times G]\!]^{\mathsf{D}}\rho & = & [\![\Gamma;\Phi \mid \Delta \vdash s : F]\!]^{\mathsf{D}}\rho \times [\![\Gamma;\Phi \mid \Delta \vdash t : G]\!]^{\mathsf{D}}\rho \\
[\![\Gamma;\Phi \mid \Delta \vdash \pi_1 t : F]\!]^{\mathsf{D}}\rho & = & \pi_1 \circ [\![\Gamma;\Phi \mid \Delta \vdash t : F \times G]\!]^{\mathsf{D}}\rho \\
[\![\Gamma;\Phi \mid \Delta \vdash \pi_2 t : G]\!]^{\mathsf{D}}\rho & = & \pi_2 \circ [\![\Gamma;\Phi \mid \Delta \vdash t : F \times G]\!]^{\mathsf{D}}\rho \\
[\![\Gamma;\Phi \mid \Delta \vdash \mathsf{case}\ t\ \mathsf{of}\ \{x \mapsto l;\ y \mapsto r\} : K]\!]^{\mathsf{D}}\rho & = & \mathrm{eval} \circ \langle \mathrm{curry}\,[[\![\Gamma;\Phi \mid \Delta, x : F \vdash l : K]\!]^{\mathsf{D}}\rho, \\
 & & \quad [\![\Gamma;\Phi \mid \Delta, y : G \vdash r : K]\!]^{\mathsf{D}}\rho], \\
 & & \quad [\![\Gamma;\Phi \mid \Delta \vdash t : F + G]\!]^{\mathsf{D}}\rho \rangle \\
[\![\Gamma;\Phi \mid \Delta \vdash \mathsf{inl}\, s : F + G]\!]^{\mathsf{D}}\rho & = & \mathsf{inl} \circ [\![\Gamma;\Phi \mid \Delta \vdash s : F]\!]^{\mathsf{D}}\rho \\
[\![\Gamma;\Phi \mid \Delta \vdash \mathsf{inr}\, t : F + G]\!]^{\mathsf{D}}\rho & = & \mathsf{inr} \circ [\![\Gamma;\Phi \mid \Delta \vdash t : G]\!]^{\mathsf{D}}\rho \\
[\![\Gamma;\emptyset \mid \emptyset \vdash \mathsf{map}_H^{\overline{F},\overline{G}} : \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\overline{\beta},\overline{\gamma}}\, F\, G) & = & \lambda d\, \overline{\eta}\, \overline{C}.\, [\![\Gamma;\overline{\phi},\overline{\gamma} \vdash H]\!]^{\mathsf{D}}\, id_{\rho[\overline{\gamma:=C}]}[\overline{\phi := \lambda \overline{B}.\eta_{\overline{B}\,\overline{C}}}] \\
\quad (\mathsf{Nat}^{\overline{\gamma}}\, H[\overline{\phi :=_{\overline{\beta}} F}]\, H[\overline{\phi :=_{\overline{\beta}} G}])]\!]^{\mathsf{D}}\rho & & \\
[\![\Gamma;\emptyset \mid \emptyset \vdash \mathsf{in}_H : Nat^{\overline{\beta},\overline{\gamma}}\, H[\phi := (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}][\overline{\alpha := \beta}] & = & \lambda d\, \overline{B}\, \overline{C}.\, (in_{T^X_{H,\rho[\overline{\gamma:=C}]}})_{\overline{B}} \quad \text{where } X \text{ is Set when} \\
\quad (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]\!]^{\mathsf{D}}\rho & & \quad \mathsf{D} = \mathsf{Set} \text{ and not present when } \mathsf{D} = \mathsf{Rel} \\
[\![\Gamma;\emptyset \mid \emptyset \vdash \mathrm{fold}_H^F : \mathsf{Nat}^{\emptyset}\ (\mathsf{Nat}^{\overline{\beta},\overline{\gamma}}\, H[\phi :=_{\overline{\beta}} F][\overline{\alpha := \beta}]\, F) & = & \lambda d\, \eta\, \overline{B}\, \overline{C}.\, (fold_{T^X_{H,\rho[\overline{\gamma:=C}]}}\, (\lambda \overline{A}.\, \eta_{\overline{A}\,\overline{C}}))_{\overline{B}} \\
\quad (\mathsf{Nat}^{\overline{\beta},\overline{\gamma}}\, (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}\ F)]\!]^{\mathsf{D}}\rho & & \quad \text{where } X \text{ is as above}
\end{array}
$$

Fig. 3. Term semantics

Every well-formed term $\Gamma;\Phi \mid \Delta \vdash t : F$ then has, for every $\rho \in \mathsf{SetEnv}$, set interpretations $[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathrm{Set}}\rho$ as natural transformations from $[\![\Gamma;\Phi \vdash \Delta]\!]^{\mathrm{Set}}\rho$ to $[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Set}}\rho$, and, for every $\rho \in \mathsf{RelEnv}$, relational interpretations $[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathrm{Rel}}\rho$ as natural transformations from $[\![\Gamma;\Phi \vdash \Delta]\!]^{\mathrm{Rel}}\rho$ to $[\![\Gamma;\Phi \vdash F]\!]^{\mathrm{Rel}}\rho$. These are given in Definition 24.

Definition 24. *If ρ is a set (resp., relation) environment and $\Gamma;\Phi \mid \Delta \vdash t : F$ then $[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathrm{Set}}\rho$ (resp., $[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathrm{Rel}}\rho$) is defined as in Figure 3, where* D *is either* Set *or* Rel *as appropriate.*

5.1 Basic Properties of Term Interpretations

The interpretations in Definition 24 respect weakening, i.e., a term and its weakenings all have the same set and relational interpretations. Specifically, for any $\rho \in \mathsf{SetEnv}$, $[\![\Gamma;\Phi \mid \Delta, x : F \vdash t : G]\!]^{\mathrm{Set}}\rho = ([\![\Gamma;\Phi \mid \Delta \vdash t : G]\!]^{\mathrm{Set}}\rho) \circ \pi_{\Delta}$, where π_{Δ} is the projection $[\![\Gamma;\Phi \vdash \Delta, x : F]\!]^{\mathrm{Set}} \to [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathrm{Set}}$. A similar result holds for relational interpretations.

The return type for the semantic fold is $[\![\Gamma;\overline{\beta},\overline{\gamma} \vdash F]\!]^{\mathsf{D}}\rho[\overline{\gamma := C}][\overline{\beta := B}]$. This interpretation gives $[\![\Gamma;\emptyset \mid \Delta \vdash \lambda x.t : F \to G]\!]^{\mathsf{D}}\rho = \mathrm{curry}([\![\Gamma;\emptyset \mid \Delta, x : F \vdash t : G]\!]^{\mathsf{D}}\rho)$ and $[\![\Gamma;\emptyset \mid \Delta \vdash st : G]\!]^{\mathsf{D}}\rho = \mathrm{eval} \circ \langle [\![\Gamma;\emptyset \mid \Delta \vdash s : F \to G]\!]^{\mathsf{D}}\rho, [\![\Gamma;\emptyset \mid \Delta \vdash t : F]\!]^{\mathsf{D}}\rho \rangle$, so it specializes to the standard interpretations for System F terms. If t is closed, i.e., if $\emptyset;\emptyset \mid \emptyset \vdash t : F$, then we write $[\![\vdash t : F]\!]^{\mathsf{D}}$ instead of $[\![\emptyset;\emptyset \mid \emptyset \vdash t : F]\!]^{\mathsf{D}}$. Moreover, if $\Gamma, \alpha;\Phi \mid \Delta \vdash t : F$ and $\Gamma;\Phi, \alpha \mid \Delta \vdash t' : F$ and $\Gamma;\Phi \vdash G$ then

- $[\![\Gamma;\Phi \mid \Delta[\alpha := G] \vdash t[\alpha := G] : F[\alpha := G]]\!]^{\mathrm{Set}}\rho = [\![\Gamma, \alpha;\Phi \mid \Delta \vdash t : F]\!]^{\mathrm{Set}}\rho[\alpha := [\![\Gamma;\Phi \vdash G]\!]^{\mathrm{Set}}\rho]$
- $[\![\Gamma;\Phi \mid \Delta[\alpha := G] \vdash t'[\alpha := G] : F[\alpha := G]]\!]^{\mathrm{Set}}\rho = [\![\Gamma;\Phi, \alpha \mid \Delta \vdash t' : F]\!]^{\mathrm{Set}}\rho[\alpha := [\![\Gamma;\Phi \vdash G]\!]^{\mathrm{Set}}\rho]$

and if $\Gamma;\Phi \mid \Delta, x : G \vdash t : F$ and $\Gamma;\Phi \mid \Delta \vdash s : G$ then

- $\lambda A.\, [\![\Gamma;\Phi \mid \Delta \vdash t[x := s] : F]\!]^{\mathrm{Set}} \rho\, A = \lambda A.\, [\![\Gamma;\Phi \mid \Delta, x : G \vdash t : F]\!]^{\mathrm{Set}} \rho\, (A, [\![\Gamma;\Phi \mid \Delta \vdash s : G]\!]^{\mathrm{Set}} \rho\, A)$

Direct calculation reveals that the set interpretations of terms also satisfy

- $[\![\Gamma;\Phi \mid \Delta \vdash (L_{\overline{\alpha}} x.t)_{\overline{K}} s]\!]^{\mathrm{Set}} = [\![\Gamma;\Phi \mid \Delta \vdash t[\overline{\alpha := K}][x := s]]\!]^{\mathrm{Set}}$

Term extensionality with respect to types and terms — i.e., $[\![\Gamma;\Phi \vdash (L_\alpha x.t)_\alpha \top : F]\!]^{\mathrm{Set}} = [\![\Gamma;\Phi \vdash t : F]\!]^{\mathrm{Set}}$ and $[\![\Gamma;\Phi \vdash (L_\alpha x.t)_\alpha x : F]\!]^{\mathrm{Set}} = [\![\Gamma;\Phi \vdash t : F]\!]^{\mathrm{Set}}$ — follow (when both sides of the equations are defined). Similar properties hold for relational interpretations.

5.2 Properties of Terms of Nat-Type

Define, for $\Gamma;\overline{\alpha} \vdash F$, the term id_F to be $\Gamma;\emptyset \mid \emptyset \vdash L_{\overline{\alpha}} x.x : \mathrm{Nat}^{\overline{\alpha}} F\, F$ and, for terms $\Gamma;\emptyset \mid \Delta \vdash t : \mathrm{Nat}^{\overline{\alpha}} F\, G$ and $\Gamma;\emptyset \mid \Delta \vdash s : \mathrm{Nat}^{\overline{\alpha}} G\, H$, the *composition* $s \circ t$ of t and s to be $\Gamma;\emptyset \mid \Delta \vdash L_{\overline{\alpha}} x.s_{\overline{\alpha}}(t_{\overline{\alpha}} x) : \mathrm{Nat}^{\overline{\alpha}} F\, H$. Then $[\![\Gamma;\emptyset \mid \emptyset \vdash id_F : \mathrm{Nat}^{\overline{\alpha}} F\, F]\!]^{\mathrm{Set}} \rho * = id_{\lambda \overline{A}.\, [\![\Gamma;\overline{\alpha} \vdash F]\!]^{\mathrm{Set}} \rho[\overline{\alpha := A}]}$ for any set environment ρ, and $[\![\Gamma;\emptyset \mid \Delta \vdash s \circ t : \mathrm{Nat}^{\overline{\alpha}} F\, H]\!]^{\mathrm{Set}} = [\![\Gamma;\emptyset \mid \Delta \vdash s : \mathrm{Nat}^{\overline{\alpha}} G\, H]\!]^{\mathrm{Set}} \circ [\![\Gamma;\emptyset \mid \Delta \vdash t : \mathrm{Nat}^{\overline{\alpha}} F\, G]\!]^{\mathrm{Set}}$. Straightforward calculation shows that terms of Nat type behave as natural transformations with respect to their source and target functorial types, i.e., we have

Theorem 25. *If* $\Gamma;\emptyset \mid \Delta \vdash s : \mathrm{Nat}^{\overline{\alpha},\overline{\gamma}} F\, G$ *and* $\overline{\Gamma;\emptyset \mid \Delta \vdash t : \mathrm{Nat}^{\overline{\gamma}} K\, H}$, *then*

$$\begin{aligned} &[\![\Gamma;\emptyset \mid \Delta \vdash ((\mathrm{map}_G^{\overline{K},\overline{H}})_\emptyset\, \overline{t}) \circ (L_{\overline{\gamma}} z.s_{\overline{K},\overline{\gamma}} z) : \mathrm{Nat}^{\overline{\gamma}} F[\overline{\alpha := K}]\, G[\overline{\alpha := H}]]\!]^{\mathrm{Set}} \\ &= [\![\Gamma;\emptyset \mid \Delta \vdash (L_{\overline{\gamma}} z.s_{\overline{H},\overline{\gamma}} z) \circ ((\mathrm{map}_F^{\overline{K},\overline{H}})_\emptyset\, \overline{t}) : \mathrm{Nat}^{\overline{\gamma}} F[\overline{\alpha := K}]\, G[\overline{\alpha := H}]]\!]^{\mathrm{Set}} \end{aligned}$$

When $s = \mathrm{in}_H$, and $\overline{\Gamma;\emptyset \mid t : \mathrm{Nat}^{\overline{\gamma}} F\, G}$, and $\xi = \mathrm{Nat}^{\overline{\gamma}} H[\phi := (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}][\overline{\alpha := F}]\, (\mu\phi.\lambda\overline{\alpha}.H)\overline{G}$, Theorem 25 specializes to

$$\begin{aligned} &[\![\Gamma;\emptyset \mid \Delta \vdash ((\mathrm{map}^{\overline{F},\overline{G}}_{(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}})_\emptyset\, \overline{t}) \circ (L_{\overline{\gamma}} z.(\mathrm{in}_H)_{\overline{F},\overline{\gamma}} z) : \xi]\!]^{\mathrm{Set}} \\ &= [\![\Gamma;\emptyset \mid \Delta \vdash (L_{\overline{\gamma}} z.(\mathrm{in}_H)_{\overline{G},\overline{\gamma}} z) \circ ((\mathrm{map}^{\overline{F},\overline{G}}_{H[\phi:=(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]})_\emptyset\, \overline{t}) : \xi]\!]^{\mathrm{Set}} \end{aligned} \tag{11}$$

Theorem 25 gives a family of free theorems that are consequences of naturality, and thus do not require the full power of parametricity. Other specific instances include the three `map`/`reverse` theorems from Section 1. The `map`/`reverse` theorem for bushes, e.g., is

$$\begin{aligned} &[\![\emptyset;\emptyset \mid y : \mathrm{Nat}^{\emptyset} F\, G \vdash ((\mathrm{map}^{F,G}_{Bush\,\alpha} P)_\emptyset\, y) \circ (L_\emptyset z.reverseBush_F\, z) : \mathrm{Nat}^{\emptyset}(Bush\, F)\, (Bush\, G)]\!]^{\mathrm{Set}} \\ &= [\![\emptyset;\emptyset \mid y : \mathrm{Nat}^{\emptyset} F\, G \vdash (L_\emptyset z.reverseBush_G\, z) \circ ((\mathrm{map}^{F,G}_{Bush\,\alpha})_\emptyset\, y) : \mathrm{Nat}^{\emptyset}(Bush\, F)\, (Bush\, G)]\!]^{\mathrm{Set}} \end{aligned}$$

for any term $\vdash reverseBush : \mathrm{Nat}^{\alpha}(Bush\,\alpha)\,(Bush\,\alpha)$. We can also prove a theorem for the data type $\emptyset;\alpha,\gamma \vdash Tree\,\alpha\,\gamma$ from Section 2 that is similar but only maps along some of its functorial variables: if $\vdash reverseTree : \mathrm{Nat}^{\alpha,\gamma}(Tree\,\alpha\,\gamma)\,(Tree\,\alpha\,\gamma)$ then

$$\begin{aligned} &[\![\emptyset;\emptyset \mid y : \mathrm{Nat}^{\gamma} F\, G \vdash ((\mathrm{map}^{F,G}_{Tree\,\alpha\,\gamma})_\emptyset\, y) \circ (L_\gamma z.reverseTree_{F,\gamma}\, z) : \mathrm{Nat}^{\gamma}(Tree\, F\,\gamma)\,(Tree\, G\,\gamma)]\!]^{\mathrm{Set}} \\ &= [\![\emptyset;\emptyset \mid y : \mathrm{Nat}^{\gamma} F\, G \vdash (L_\gamma z.reverseTree_{G,\gamma}\, z) \circ ((\mathrm{map}^{F,G}_{Tree\,\alpha\,\gamma})_\emptyset\, y) : \mathrm{Nat}^{\gamma}(Tree\, F\,\gamma)\,(Tree\, G\,\gamma)]\!]^{\mathrm{Set}} \end{aligned}$$

We observed in Section 2 that $\phi;\alpha \vdash GRose\,\phi\,\alpha$ cannot appear as the (co)domain of a Nat-type binding both ϕ and α. As a result, there are no instances of, say, a `map`/`reverse` free theorem for this data type that maps replacements for both ϕ and α. On the other hand, $\phi;\alpha \vdash GRose\,\phi\,\alpha$ does support a `map`/`reverse` theorem that maps only a replacement for α.

As usual, analogous results hold for relation environments and relational interpretations.

5.3 Properties of Initial Algebraic Constructs

Our semantics interprets applications of map as applications of semantic *map* functions. In particular, if $\Gamma; \overline{\alpha} \vdash H$, $\overline{\Gamma; \emptyset \vdash F}$, and $\overline{\Gamma; \emptyset \vdash G}$, then Definition 24 gives, for all $\overline{f : [\![\Gamma; \emptyset \vdash \mathsf{Nat}^{\emptyset} F\, G]\!]^{\mathrm{Set}} \rho}$, that

$$\begin{aligned} & [\![\Gamma; \emptyset \mid \overline{x : \mathsf{Nat}^{\emptyset} F\, G} \vdash (\mathsf{map}_H^{\overline{F},\overline{G}})_{\emptyset}\, \overline{x} : \mathsf{Nat}^{\emptyset}\, H[\overline{\alpha := F}]\, H[\overline{\alpha := G}]]\!]^{\mathrm{Set}} \rho\, \overline{f} \\ = \;& [\![\Gamma; \overline{\alpha} \vdash H]\!]^{\mathrm{Set}} id_{\rho}[\overline{\alpha := f}] \\ = \;& map_{\lambda \overline{A}.\, [\![\Gamma; \overline{\alpha} \vdash H]\!]^{\mathrm{Set}} \rho[\overline{\alpha := A}]}\, \overline{f} \end{aligned}$$

Here, we obtain the first equality from the appropriate instance of Definition 24, and the second one by noting that $\lambda \overline{A}.\, [\![\Gamma; \overline{\alpha} \vdash H]\!]^{\mathrm{Set}} \rho[\overline{\alpha := A}]$ is a functor in α and using $map_G\, \overline{f}$ to denote the action of the semantic functor G on morphisms $\overline{f}$.

We have the expected properties of the interpretations of map, in, and fold, namely:

- If $\Gamma; \overline{\psi}, \overline{\gamma} \vdash H$, $\overline{\Gamma; \overline{\alpha}, \overline{\gamma}, \overline{\phi} \vdash K}$, $\overline{\Gamma; \overline{\beta}, \overline{\gamma} \vdash F}$, and $\overline{\Gamma; \overline{\beta}, \overline{\gamma} \vdash G}$, then

$$[\![\Gamma; \emptyset \mid \emptyset \vdash \mathsf{map}^{\overline{F},\overline{G}}_{H[\overline{\psi := K}]} : \xi]\!]^{\mathrm{Set}} = [\![\Gamma; \emptyset \mid \emptyset \vdash L_{\emptyset} \overline{z}.(\mathsf{map}_H^{\overline{K[\overline{\phi := F}]}, \overline{K[\overline{\phi := G}]}})_{\emptyset}\, (\overline{(\mathsf{map}_K^{\overline{F},\overline{G}})_{\emptyset}\, \overline{z}}) : \xi]\!]^{\mathrm{Set}}$$

for $\xi = \mathsf{Nat}^{\emptyset}(\overline{\mathsf{Nat}^{\overline{\alpha},\overline{\beta},\overline{\gamma}} F\, G})\, (\mathsf{Nat}^{\overline{\gamma}} H[\overline{\psi := K}][\overline{\phi := F}]\, H[\overline{\psi := K}][\overline{\phi := G}])$

- If $\Gamma; \overline{\beta}, \overline{\gamma} \vdash H$, $\Gamma; \overline{\beta}, \overline{\gamma} \vdash K$, $\overline{\Gamma; \overline{\alpha}, \overline{\gamma} \vdash F}$, $\overline{\Gamma; \emptyset, \overline{\alpha}, \overline{\gamma} \vdash G}$, $\overline{\Gamma; \phi, \overline{\psi}, \overline{\gamma} \vdash M}$, $\overline{I}$ is the sequence $\overline{F}, H$, $\overline{J}$ is the sequence $\overline{G}, K$, and $\Gamma; \emptyset \mid \Delta \vdash s : \mathsf{Nat}^{\overline{\beta},\overline{\gamma}} H\, K$ and $\Gamma; \emptyset \mid \Delta \vdash t : \mathsf{Nat}^{\overline{\alpha},\overline{\gamma}} F\, G$, then

$$[\![\Gamma; \emptyset \mid \Delta \vdash L_{\overline{\gamma}} z.\, s_{\overline{M[\overline{\psi:=F}][\phi:=H]}, \overline{M[\overline{\psi:=G}][\phi:=K]}, \overline{\gamma}} \Big(((\mathsf{map}_H^{\overline{M[\overline{\psi:=F}][\phi:=H]}, \overline{M[\overline{\psi:=G}][\phi:=K]}})_{\emptyset} (\overline{(\mathsf{map}_M^{\overline{I},\overline{J}})_{\emptyset}(s, \overline{t})}))_{\overline{\gamma}} z \Big) : \xi]\!]^{\mathrm{Set}}$$

$$= [\![\Gamma; \emptyset \mid \Delta \vdash L_{\overline{\gamma}} z.\, ((\mathsf{map}_K^{\overline{M[\overline{\psi:=F}][\phi:=H]}, \overline{M[\overline{\psi:=G}][\phi:=K]}})_{\emptyset} (\overline{(\mathsf{map}_M^{\overline{I},\overline{J}})_{\emptyset}(s, \overline{t})}))_{\overline{\gamma}} \Big(s_{\overline{M[\overline{\psi:=F}][\phi:=H]}, \overline{\gamma}} z \Big) : \xi]\!]^{\mathrm{Set}}$$

for $\xi = \mathsf{Nat}^{\overline{\gamma}}\, H[\overline{\beta := M}][\overline{\psi := F}][\phi := H]\; K[\overline{\beta := M}][\overline{\psi := G}][\phi := K]$

- If $\xi = \mathsf{Nat}^{\overline{\beta},\overline{\gamma}}\, H[\phi := (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}][\overline{\alpha := \beta}]\; F$, then

$$\begin{aligned} & [\![\Gamma; \emptyset \mid x : \mathsf{Nat}^{\overline{\beta},\overline{\gamma}} H[\phi := F][\overline{\alpha := \beta}]\, F \vdash ((\mathsf{fold}_{H,F})_{\emptyset} x) \circ \mathsf{in}_H : \xi]\!]^{\mathrm{Set}} \\ = \;& [\![\Gamma; \emptyset \mid x : \mathsf{Nat}^{\overline{\beta},\overline{\gamma}} H[\phi := F][\overline{\alpha := \beta}]\, F \vdash x \circ \big((\mathsf{map}^{(\mu\phi.\lambda\overline{\alpha}H)\overline{\beta},F}_{H[\overline{\alpha:=\beta}]})_{\emptyset} ((\mathsf{fold}_{H,F})_{\emptyset} x) \big) : \xi]\!]^{\mathrm{Set}} \end{aligned}$$

- If $\xi = \mathsf{Nat}^{\overline{\beta},\overline{\gamma}}\, (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}\; (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}$, then

$$\begin{aligned} & [\![\Gamma; \emptyset \mid \emptyset \vdash \mathsf{in}_H \circ (\mathsf{fold}_{H,H[\phi:=(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]})_{\emptyset} ((\mathsf{map}_H^{H[\phi:=(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}][\overline{\alpha:=\beta}], (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}})_{\emptyset} \mathsf{in}_H) : \xi]\!]^{\mathrm{Set}} \\ = \;& [\![\Gamma; \emptyset \mid \emptyset \vdash id_{(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}} : \xi]\!]^{\mathrm{Set}} \end{aligned}$$

- If $\xi = \mathsf{Nat}^{\overline{\beta},\overline{\gamma}} H[\phi := (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]\, H[\phi := (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]$, then

$$\begin{aligned} & [\![\Gamma; \emptyset \mid \emptyset \vdash (\mathsf{fold}_{H,H[\phi:=(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]})_{\emptyset} ((\mathsf{map}_H^{H[\phi:=(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}][\overline{\alpha:=\beta}], (\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}})_{\emptyset} \mathsf{in}_H) \circ \mathsf{in}_H : \xi]\!]^{\mathrm{Set}} \\ = \;& [\![\Gamma; \emptyset \mid \emptyset \vdash id_{H[\phi:=(\mu\phi.\lambda\overline{\alpha}.H)\overline{\beta}]} : \xi]\!]^{\mathrm{Set}} \end{aligned}$$

Analogous results hold for relational interpretations of terms and relational environments.

5.4 The Abstraction Theorem

To go beyond naturality and get *all* consequences of parametricity, we prove an Abstraction Theorem for our calculus. As usual for such theorems, we prove a more general result in Theorem 26 that handles possibly open terms. We then recover the Abstraction Theorem (Theorem 28) as the special case Theorem 26 for closed terms of closed type.

Theorem 26. *Every well-formed term* $\Gamma;\Phi \mid \Delta \vdash t : F$ *induces a natural transformation from* $[\![\Gamma;\Phi \vdash \Delta]\!]$ *to* $[\![\Gamma;\Phi \vdash F]\!]$*, i.e., a triple of natural transformations*

$$([\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}},\ [\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}},\ [\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Rel}})$$

where

$$[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}} \quad : \quad [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{Set}} \to [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Set}}$$

has as its component at $\rho : \mathsf{SetEnv}$ *a morphism*

$$[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}}\rho \quad : \quad [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{Set}}\rho \to [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Set}}\rho$$

in Set*,*

$$[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Rel}} \quad : \quad [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{Rel}} \to [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}$$

has as its component at $\rho : \mathsf{RelEnv}$ *a morphism*

$$[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Rel}}\rho \quad : \quad [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{Rel}}\rho \to [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}\rho$$

in Rel*, and, for all* $\rho : \mathsf{RelEnv}$*,*

$$[\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Rel}}\rho = ([\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}}(\pi_1\rho),\ [\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}}(\pi_2\rho)) \tag{12}$$

Proof. By structural induction on the type judgment for t. The overall challenge of our development manifests in this proof. It lies in showing that the set and relational interpretations of each term judgment are natural transformations, and that all set interpretations of terms of Nat-types satisfy the appropriate equality preservation conditions from Definition 7. For the interesting cases of abstraction, application, map, in, and fold terms, propagating the naturality conditions is quite involved; the latter two cases especially require some rather delicate diagram chasing. That this is possible provides strong evidence that our definitions and development are sensible, natural, and at an appropriate level of abstraction. Detailed proofs for the set interpretations for the five interesting forms of terms are included in the accompanying anonymous supplementary material. Even in the supplementary material we omit the other, straightforward, cases. The proofs for the relational interpretations of all term judgments are entirely analogous to the proofs for their set interpretations. The proof that Equation 12 holds in each case is by direct calculation, using the facts that projections are surjective and that the set and relational interpretations are defined "in parallel", i.e., are fibred. □

The following theorem is an immediate consequence of Theorem 26:

Theorem 27. *If* $\Gamma;\Phi \mid \Delta \vdash t : F$ *and* $\rho \in \mathsf{RelEnv}$*, and if* $(a, b) \in [\![\Gamma;\Phi \vdash \Delta]\!]^{\mathsf{Rel}}\rho$*, then* $([\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}}(\pi_1\rho)\,a\,,\ [\![\Gamma;\Phi \mid \Delta \vdash t : F]\!]^{\mathsf{Set}}(\pi_2\rho)\,b) \in [\![\Gamma;\Phi \vdash F]\!]^{\mathsf{Rel}}\rho$

Finally, the Abstraction Theorem is the instantiation of Theorem 27 to closed terms of closed type:

Theorem 28 (Abstraction Theorem). *If* $\vdash t : F$*, then* $([\![\vdash t : F]\!]^{\mathsf{Set}}, [\![\vdash t : F]\!]^{\mathsf{Set}}) \in [\![\vdash F]\!]^{\mathsf{Rel}}$.

6 FREE THEOREMS FOR NESTED TYPES

In this section we show how Theorem 28 can be used to prove more advanced free theorems than those of Section 5.2 in the presence of directly constructed nested types.

6.1 Free Theorem for Type of Polymorphic Bottom

Suppose $\vdash g : \mathsf{Nat}^\alpha\, \mathbb{1}\, \alpha$, let $G^{\mathsf{Set}} = [\![\vdash g : \mathsf{Nat}^\alpha\, \mathbb{1}\, \alpha]\!]^{\mathsf{Set}}$, and let $G^{\mathsf{Rel}} = [\![\vdash g : \mathsf{Nat}^\alpha\, \mathbb{1}\, \alpha]\!]^{\mathsf{Rel}}$. By Theorem 26, $(G^{\mathsf{Set}}(\pi_1\rho), G^{\mathsf{Set}}(\pi_2\rho)) = G^{\mathsf{Rel}}\rho$. Thus, for all $\rho \in \mathsf{RelEnv}$ and any $(a, b) \in [\![\vdash \emptyset]\!]^{\mathsf{Rel}}\rho = 1$, eliding the only possible instantiations of a and b gives that $(G^{\mathsf{Set}}, G^{\mathsf{Set}}) = (G^{\mathsf{Set}}(\pi_1\rho), G^{\mathsf{Set}}(\pi_2\rho)) \in [\![\vdash \mathsf{Nat}^\alpha\, \mathbb{1}\, \alpha]\!]^{\mathsf{Rel}}\rho = \{\eta : K_1 \Rightarrow id\} = \{(\eta_1 : K_1 \Rightarrow id, \eta_2 : K_1 \Rightarrow id)\}$ That is, G^{Set} is a natural transformation from the constantly 1-valued functor to the identity functor in Set. In particular, for every $S : \mathsf{Set}$, $G^{\mathsf{Set}}_S : 1 \to S$. Note, however, that if $S = \emptyset$, then there can be no such morphism, so no such natural transformation, and thus no term $\vdash g : \mathsf{Nat}^\alpha \mathbb{1}\, \alpha$, can exist. That is, our calculus admits no non-terminating terms, i.e., terms with the closed type $\mathsf{Nat}^\alpha \mathbb{1}\, \alpha$ of the polymorphic bottom.

6.2 Free Theorem for Type of Polymorphic Identity

Suppose $\vdash g : \mathsf{Nat}^\alpha\, \alpha\, \alpha$, let $G^{\mathsf{Set}} = [\![\vdash g : \mathsf{Nat}^\alpha\, \alpha\, \alpha]\!]^{\mathsf{Set}}$, and let $G^{\mathsf{Rel}} = [\![\vdash g : \mathsf{Nat}^\alpha\, \alpha\, \alpha]\!]^{\mathsf{Rel}}$. By Theorem 26, $(G^{\mathsf{Set}}(\pi_1\rho), G^{\mathsf{Set}}(\pi_2\rho)) = G^{\mathsf{Rel}}\rho$. Thus, for all $\rho \in \mathsf{RelEnv}$ and any $(a, b) \in [\![\vdash \emptyset]\!]^{\mathsf{Rel}}\rho = 1$, eliding the only possible instantiations of a and b gives that $(G^{\mathsf{Set}}, G^{\mathsf{Set}}) = (G^{\mathsf{Set}}(\pi_1\rho), G^{\mathsf{Set}}(\pi_2\rho)) \in [\![\vdash \mathsf{Nat}^\alpha\, \alpha\, \alpha]\!]^{\mathsf{Rel}}\rho = \{\eta : id \Rightarrow id\} = \{(\eta_1 : id \Rightarrow id, \eta_2 : id \Rightarrow id)\}$ That is, G^{Set} is a natural transformation from the identity functor on Set to itself. Now let S be any set. If $S = \emptyset$, then there is exactly one morphism $id_S : S \to S$, so $G^{\mathsf{Set}}_S : S \to S$ must be id_S. If $S \neq \emptyset$, if a is any element of S, and if $K_a : S \to S$ is the constantly a-valued morphism on S, then instantiating the naturality square implied by the above equality gives that $G^{\mathsf{Set}}_S \circ K_a = K_a \circ G^{\mathsf{Set}}_S$, i.e., $G^{\mathsf{Set}}_S\, a = a$, i.e., $G^{\mathsf{Set}}_S = id_S$. Putting these two cases together we have that for every $S : \mathsf{Set}$, $G^{\mathsf{Set}}_S = id_S$, i.e., G^{Set} is the identity natural transformation for the identity functor on Set. So every closed term g of closed type $\mathsf{Nat}^\alpha\, \alpha\, \alpha$ always denotes the identity natural transformation for the identity functor on Set, i.e., every closed term g of type $\mathsf{Nat}^\alpha\, \alpha\, \alpha$ denotes the polymorphic identity function.

6.3 Standard Free Theorems for ADTs and Their Analogues for Nested types

We can derive in our calculus even those free theorems for polymorphic functions over ADTs that are not consequences of naturality. We can, e.g., prove the free theorem for *filter*'s type as follows:

Theorem 29. *If* $g : A \to B$, $\rho : \mathsf{RelEnv}$, $\rho\alpha = (A, B, \langle g \rangle)$, $(a, b) \in [\![\alpha; \emptyset \vdash \Delta]\!]^{\mathsf{Rel}}\rho$, $(s \circ g, s) \in [\![\alpha; \emptyset \vdash \mathsf{Nat}^\emptyset \alpha\, \mathit{Bool}]\!]^{\mathsf{Rel}}\rho$, *and* $\mathit{filter} = [\![\alpha; \emptyset \mid \Delta \vdash t : \mathsf{Nat}^\emptyset(\mathsf{Nat}^\emptyset\, \alpha\, \mathit{Bool})\, (\mathsf{Nat}^\emptyset(\mathit{List}\, \alpha)\, (\mathit{List}\, \alpha))]\!]^{\mathsf{Set}}$ *for some* t, *then*

$$\mathit{map}_{\mathit{List}}\, g \circ \mathit{filter}\, (\pi_1\rho)\, a\, (s \circ g) = \mathit{filter}\, (\pi_2\rho)\, b\, s \circ \mathit{map}_{\mathit{List}}\, g$$

Proof. By Theorem 27,

$$(\mathit{filter}\, (\pi_1\rho)\, a, \mathit{filter}\, (\pi_2\rho)\, b) \in [\![\alpha; \emptyset \vdash \mathsf{Nat}^\emptyset(\mathsf{Nat}^\emptyset\, \alpha\, \mathit{Bool})\, (\mathsf{Nat}^\emptyset(\mathit{List}\, \alpha)\, (\mathit{List}\, \alpha))]\!]^{\mathsf{Rel}}\rho$$

so if $(s', s) \in [\![\alpha; \emptyset \vdash \mathsf{Nat}^\emptyset \alpha\, \mathit{Bool}]\!]^{\mathsf{Rel}}\rho = \rho\alpha \to \mathsf{Eq}_{\mathit{Bool}}$ and $(xs', xs) \in [\![\alpha; \emptyset \vdash \mathit{List}\, \alpha]\!]^{\mathsf{Rel}}\rho$ then

$$(\mathit{filter}\, (\pi_1\rho)\, a\, s'\, xs', \mathit{filter}\, (\pi_2\rho)\, b\, s\, xs) \in [\![\alpha; \emptyset \vdash \mathit{List}\, \alpha]\!]^{\mathsf{Rel}}\rho \tag{13}$$

If $\rho\alpha = (A, B, \langle g \rangle)$, then $[\![\alpha; \emptyset \vdash \mathit{List}\, \alpha]\!]^{\mathsf{Rel}}\rho = \langle \mathit{map}_{\mathit{List}}\, g \rangle$ by Lemma 23 and Equation 4. Moreover, $xs = \mathit{map}_{\mathit{List}}\, g\, xs'$ and $(s', s) \in \langle g \rangle \to \mathsf{Eq}_{\mathit{Bool}}$, so $s' = s \circ g$. The result follows from Equation 13. □

A similar proof establishes the analogous result for, say, generalized rose trees.

Theorem 30. *If* $g : A \to B$, $F, G : \mathsf{Set} \to \mathsf{Set}$, $\eta : F \to G$ *in* Set, $\rho : \mathsf{RelEnv}$, $\rho\alpha = (A, B, \langle g \rangle)$, $\rho\psi = (F, G, \langle \eta \rangle)$, $(a, b) \in [\![\alpha, \psi; \emptyset \vdash \Delta]\!]^{\mathsf{Rel}}\rho$, $(s \circ g, s) \in [\![\alpha; \emptyset \vdash \mathsf{Nat}^\emptyset \alpha\, \mathit{Bool}]\!]^{\mathsf{Rel}}\rho$, *and*

$$\mathit{filter} = [\![\alpha, \psi; \emptyset \mid \Delta \vdash t : \mathsf{Nat}^\emptyset(\mathsf{Nat}^\emptyset\, \alpha\, \mathit{Bool})\, (\mathsf{Nat}^\emptyset(\mathit{GRose}\, \psi\, \alpha)\, (\mathit{GRose}\, \psi\, (\alpha + \mathbb{1})))]\!]^{\mathsf{Set}}$$

for some t, *then*

$$\mathit{map}_{\mathit{GRose}}\, \eta\, (g + 1) \circ \mathit{filter}\, (\pi_1\rho)\, a\, (s \circ g) = \mathit{filter}\, (\pi_2\rho)\, b\, s \circ \mathit{map}_{\mathit{GRose}}\, \eta\, g$$

This is not surprising since rose trees are essentially ADT-like. However, as noted in Section 2.2, our calculus cannot express the type of a polymorphic filter function for a proper nested type.

6.4 Short Cut Fusion for Lists

We can recover standard short cut fusion for lists [Gill et al. 1993] in our calculus:

Theorem 31. *If* $\vdash F$, $\vdash H$, *and* $G = [\![\beta; \emptyset \mid \emptyset \vdash g : \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\emptyset}(\mathbb{1} + F \times \beta)\,\beta)\,\beta]\!]^{\mathrm{Set}}$ *for some* g, *then*

$$fold_{1+[\![\vdash F]\!]^{\mathrm{Set}}\times\_}\ n\ c\ (G\ (List\ [\![\vdash F]\!]^{\mathrm{Set}})\ nil\ cons) = G\,[\![\vdash H]\!]^{\mathrm{Set}}\ n\ c$$

Proof. Theorem 28 gives that, for any $\rho : \mathrm{RelEnv}$,

$(G\,(\pi_1\rho), G\,(\pi_2\rho)) \in [\![\beta; \emptyset \vdash \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\emptyset}(\mathbb{1} + F \times \beta)\,\beta)\,\beta]\!]^{\mathrm{Rel}}\rho \cong ((([\![\vdash F]\!]^{\mathrm{Rel}}\rho \times \rho\beta) \to \rho\beta) \times \rho\beta) \to \rho\beta$

so if $(c', c) \in [\![\vdash F]\!]^{\mathrm{Rel}}\rho \times \rho\beta \to \rho\beta$ and $(n', n) \in \rho\beta$ then $(G\,(\pi_1\rho)\,n'\,c', G\,(\pi_2\rho)\,n\,c) \in \rho\beta$. In addition,

$$[\![\vdash \mathsf{fold}^{H}_{\mathbb{1}+F\times\beta} : \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\emptyset}(\mathbb{1} + F \times H)\,H)\,(\mathsf{Nat}^{\emptyset}(\mu\alpha.\mathbb{1} + F \times \alpha)\,H)]\!]^{\mathrm{Set}} = fold_{1+[\![\vdash F]\!]^{\mathrm{Set}}\times\_}$$

so that if $c \in [\![\vdash F]\!]^{\mathrm{Set}} \times [\![\vdash H]\!]^{\mathrm{Set}} \to [\![\vdash H]\!]^{\mathrm{Set}}$ and $n \in [\![\vdash H]\!]^{\mathrm{Set}}$, then $(n, c) \in [\![\vdash \mathsf{Nat}^{\emptyset}(\mathbb{1} + F \times H)\,H]\!]^{\mathrm{Set}}$. The instantiation $\pi_1\rho\beta = [\![\vdash \mu\alpha.\mathbb{1} + F \times \alpha]\!]^{\mathrm{Set}} = List\ [\![\vdash F]\!]^{\mathrm{Set}}$, $\pi_2\rho\beta = [\![\vdash H]\!]^{\mathrm{Set}}$, $\rho\beta = \langle fold_{1+[\![\vdash F]\!]^{\mathrm{Set}}\times\_}\ n\ c\rangle : \mathrm{Rel}(\pi_1\rho\beta, \pi_2\rho\beta)$, $c' = cons$, and $n' = nil$ thus gives that $(G\ (List\ [\![\vdash F]\!]^{\mathrm{Set}})\ nil\ cons, G\,[\![\vdash H]\!]^{\mathrm{Set}}\ n\ c) \in \langle fold_{1+[\![\vdash F]\!]^{\mathrm{Set}}\times\_}\ n\ c\rangle$, i.e., that $fold_{1+[\![\vdash F]\!]^{\mathrm{Set}}\times\_}\ n\ c\ (G\ (List\ [\![\vdash F]\!]^{\mathrm{Set}})\ nil\ cons) = G\,[\![\vdash H]\!]^{\mathrm{Set}}\ n\ c$. □

We can extend short cut fusion results to arbitrary ADTs, as in [Johann 2002; Pitts 1998].

6.5 Short Cut Fusion for Arbitrary Nested Types

We can also, finally, formally prove correctness of the categorically inspired short cut fusion for nested types from [Johann and Ghani 2010]. Indeed, we have:

Theorem 32. *If* $\emptyset; \phi, \alpha \vdash F$, $\emptyset; \alpha \vdash K$, $H : [\mathrm{Set}, \mathrm{Set}] \to [\mathrm{Set}, \mathrm{Set}]$ *is defined by*

$$H\,f\,x \quad = \quad [\![\emptyset; \phi, \alpha \vdash F]\!]^{\mathrm{Set}}[\phi := f][\alpha := x]$$

and

$$G = [\![\phi; \emptyset \mid \emptyset \vdash g : \mathsf{Nat}^{\emptyset}\,(\mathsf{Nat}^{\alpha}\,F\,(\phi\alpha))\,(\mathsf{Nat}^{\alpha}\,\mathbb{1}\,(\phi\alpha))]\!]^{\mathrm{Set}}$$

for some g, *then, for every* $B \in H[\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}} \to [\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}}$, $fold_H\,B\,(G\ \mu H\ in_H) = G\,[\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}}\,B$.

Proof. Theorem 28 gives that, for any $\rho : \mathrm{RelEnv}$,

$$\begin{array}{rcl}(G\,(\pi_1\rho), G\,(\pi_2\rho)) & \in & [\![\phi; \emptyset \vdash \mathsf{Nat}^{\emptyset}(\mathsf{Nat}^{\alpha}F\,(\phi\alpha))\,(\mathsf{Nat}^{\alpha}\,\mathbb{1}\,(\phi\alpha))]\!]^{\mathrm{Rel}}\rho \\ & = & [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Rel}}\rho \to [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}\,\mathbb{1}\,(\phi\alpha)]\!]^{\mathrm{Rel}}\rho \\ & = & [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Rel}}\rho \to \rho\phi\end{array}$$

so if $(A, B) \in [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Rel}}\rho$ then $(G\,(\pi_1\rho)\,A, G\,(\pi_2\rho)\,B) \in \rho\phi$. In addition,

$$[\![\vdash \mathsf{fold}^{K}_{F} : \mathsf{Nat}^{\emptyset}\,(\mathsf{Nat}^{\alpha}F[\phi := K]\,K)\,(\mathsf{Nat}^{\alpha}((\mu\phi.\lambda\alpha.F)\alpha)\,K)]\!]^{\mathrm{Set}} = fold_H$$

Consider the instantiation $A = in_H : H(\mu H) \Rightarrow \mu H$, $B : H[\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}} \Rightarrow [\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}}$, $\rho\phi = \langle fold_H\,B\rangle$, $\pi_1\rho\phi = \mu H$, $\pi_2\rho\phi = [\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}}$, $\rho\phi : \mathrm{Rel}(\pi_1\rho\phi, \pi_2\rho\phi)$, $A : [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Set}}(\pi_1\rho)$, and $B : [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Set}}(\pi_2\rho)$. Equation 4 ensures that $A = in_H : H(\mu H) \Rightarrow \mu H = [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Set}}(\pi_1\rho)$, and Equation 4 and Lemma 23 together give that

$$\begin{array}{rcl}(A, B) = (in_H, B) & \in & [\![\phi; \emptyset \vdash \mathsf{Nat}^{\alpha}F\,(\phi\alpha)]\!]^{\mathrm{Rel}}\rho \\ & = & \lambda A.[\![\phi; \alpha \vdash F]\!]^{\mathrm{Rel}}[\phi := \langle fold_H\,B\rangle][\alpha := A] \Rightarrow \langle fold_H\,B\rangle \\ & = & [\![\emptyset; \phi, \alpha \vdash F]\!]^{\mathrm{Rel}}\langle fold_H\,B\rangle \Rightarrow \langle fold_H\,B\rangle \\ & = & \langle [\![\emptyset; \phi, \alpha \vdash F]\!]^{\mathrm{Set}}\,(fold_H\,B)\rangle \Rightarrow \langle fold_H\,B\rangle \\ & = & \langle map_H\,(fold_H\,B)\rangle \Rightarrow \langle fold_H\,B\rangle\end{array}$$

since if $(x, y) \in \langle map_H\,(fold_H\,B)\rangle$, then $fold_H\,B\,(in_H\,x) = B\,y = B\,(map_H\,(fold_H\,B)\,x)$ by the definition of $fold_H$ as a (indeed, the unique) morphism from in_H to B. Thus, $(G\,(\pi_1\rho)\,A, G\,(\pi_2\rho)\,B) \in \langle fold_H\,B\rangle$, i.e., $fold_H\,B\,(G\,(\pi_1\rho)\,in_H) = G\,(\pi_2\rho)\,B$. But since ϕ is the only free variable in G, this simplifies to $fold_H\,B\,(G\,\mu H\,in_H) = G\,[\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}}\,B$. □

As in [Johann and Ghani 2010], replacing $\mathbb{1}$ with any type $\emptyset; \alpha \vdash C$ generalizes Theorem 32 to deliver more general a free theorem whose conclusion is $fold_H\,B \circ G\,\mu H\,in_H = G\,[\![\emptyset; \alpha \vdash K]\!]^{\mathrm{Set}}\,B$. Although it is standard to prove that the parametric model constructed verifies the existence of initial algebras, this is unnecessary here since initial algebras are built directly into our model.

7 CONCLUSION AND DIRECTIONS FOR FUTURE WORK

We have constructed a parametric model for a calculus providing primitives for constructing nested types directly as fixpoints. We have also used the Abstraction Theorem for this model to derive free theorems in the presence of nested types. This was not possible before [Johann and Polonsky 2019] because the nested types in our calculus were not previously known to have well-defined interpretations in locally finitely presentable categories like Set and Rel. No calculus for terms of these types existed, either. The key to obtaining our parametric model is the delicate threading of functoriality and its accompanying naturality conditions throughout the model construction.

Even when nested types have well-defined interpretations in a semantic category, the types of standard functions over them may not. This is because all recursion in our calculus must be coded using standard folds. Since folds for nested types can only express computations that return natural transformations, their expressivity has long been a vexing issue, and this is naturally inherited by our calculus. In addition, while there are many papers on higher-kinded types, we know of none that considers expressibility issues arising from the local λ-presentability cardinal λ of the categories in which higher-kinded types are interpreted. Because $\lambda = \omega$ for Set and Rel, the expressivity of our calculus is, unfortunately, not easily improved there. Construction of a Set/Rel-based parametric model for our calculus is nevertheless worthwhile. For one thing, the expressivity issue is orthogonal to constructing parametric models, and trying to address both in a single paper would detract from the conceptual clarity of our model construction. For another, if the construction in this paper weren't possible, then trying to construct a parametric model from richer semantic categories for fancier data types and/or recursion schemes would be futile. We naturally expect the construction reported here to guide its generalization to the locally λ-presentable categories for $\lambda > \omega$ that will be needed to accommodate more general data types and recursion schemes for them.

Generalized folds [Bird and Paterson 1999] are an extension of standard folds that can express computations whose results are not natural transformations. These were shown in [Johann and Ghani 2010] to be equivalent to standard folds in the presence of object-level *right* Kan extensions. It was also shown in [Johann and Ghani 2008] that GADTs can be represented using object-level *left* Kan extensions, and adding a carefully designed left Kan extension construct to our calculus was shown in [Johann and Polonsky 2019] to preserve the cocontinuity needed for GADTs to have well-defined interpretations. (Note, however, that locally λ-presentable categories for $\lambda > \aleph_1$ are required to interpret even commonly used GADTs.) This suggests extending our calculus with Kan extension constructs and carrying out our model construction in a locally λ-presentable cartesian closed category (lpccc) $\mathcal{C}$ whose category of (abstract) relations, obtained by pullback as in [Jacobs 1999], is also a lpccc and appropriately fibred over $\mathcal{C}$. This would give a framework for constructing parametric models for calculi with nested types built from primitives that is based on locally λ-presentable fibrations, for some appropriate definition thereof. Another option is to add term-level fixpoints to the calculus presented here. Of course, this would require the categories interpreting types to be not just locally λ-presentable, but to support some kind of domain structure as well.

REFERENCES

J. Adámek and J. Rosický. 1994. *Locally Presentable and Accessible Categories*. Cambridge University Press.

R. Atkey. 2012. Relational Parametricity for Higher Kinds. In *Computer Science Logic*. 46–61.

E. S. Bainbridge, P. J. Freyd, A. Scedrov, and P. J. Scott. 1990. Functorial Polymorphism. *Theoretical Computer Science* 70 (1990), 35–64.

R. Bird and L. Meertens. 1998. Nested datatypes. In *Mathematics of Program Construction*. 52–67.

R. Bird and R. Paterson. 1999. Generalised folds for nested datatypes. *Formal Aspects of Computing* 11 (1999), 200–222.

L. Birkedal and R. E. Møgelberg. 2005. Categorical models for Abadi and Plotkin's logic for parametricity. *Mathematical Structures in Computer Science* 15 (2005), 709–772.

B. Dunphy and U. Reddy. 2004. Parametric limits. In *Logic in Computer Science*.

N. Ghani, P. Johann, F. Nordvall Forsberg, F. Orsanigo, and T. Revell. 2015. Bifibrational Functorial Semantics for Parametric Polymorphism. In *Mathematical Foundations of Program Semantics*. 165–181.

A. Gill, J. Launchbury, and S.L. Peyton Jones. 1993. A short cut to deforestation. In *Functional Programming Languages and Computer Architecture, Proceedings*. 223–232.

J.-Y. Girard, P. Taylor, and Y. Lafont. 1989. *Proofs and Types*. Cambridge University Press.

R. Hasegawa. 1994. Categorical data types in parametric polymorphism. *Mathematical Structures in Computer Science* 4 (1994), 71–109.

B. Jacobs. 1999. *Categorical Logic and Type Theory*. Elsevier.

P. Johann. 2002. A Generalization of Short-Cut Fusion and Its Correctness Proof. *Higher-Order and Symbolic Computation* 15 (2002), 273–300.

P. Johann and N. Ghani. 2008. Foundations for Structured Programming with GADTs. In *Principles of Programming Languages*. 297–308.

P. Johann and N. Ghani. 2010. Haskell Programming with Nested Types: A Principled Approach. *Higher-Order and Symbolic Computation* 22(2) (2010), 155–189.

P. Johann and A. Polonsky. 2019. Higher-kinded Data Types: Syntax and Semantics. In *Logic in Computer Science*. 1–13. https://doi.org/10.1109/LICS.2019.8785657

P. Johann and A. Polonsky. 2020. Deep Induction: Induction Rules for (Truly) Nested Types. In *Foundations of Software Science and Computation Structures*.

Q. Ma and J. C. Reynolds. 1992. Types, abstractions, and parametric polymorphism, part 2. In *Mathematical Foundations of Program Semantics*. 1–40.

C. Okasaki. 1999. *Purely Functional Data Structures*. Cambridge University Press.

A. Pitts. 1998. Parametric polymorphism, recursive types, and operational equivalence. (1998).

A. Pitts. 2000. Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science* 10 (2000), 1–39.

J. C. Reynolds. 1983. Types, abstraction, and parametric polymorphism. *Information Processing* 83(1) (1983), 513–523.

J. C. Reynolds. 1984. Polymorphism is not set-theoretic. *Semantics of Data Types* (1984), 145–156.

E. Robinson and G. Rosolini. 1994. Reflexive graphs and parametric polymorphism. In *Logic in Computer Science*. 364–371.

P. Wadler. 1989. Theorems for free!. In *Functional Programming Languages and Computer Architecture, Proceedings*. 347–359.