



Parametricity for Primitive Nested Types

Patricia Johann , Enrico Ghiorzi , and Daniel Jeffries

Appalachian State University, Boone, NC, USA

{johannp, ghiorzie, jeffriesd}@appstate.edu

Abstract. This paper considers parametricity and its resulting free theorems for nested data types. Rather than representing nested types via their Church encodings in a higher-kinded or dependently typed extension of System F, we adopt a functional programming perspective and design a Hindley-Milner-style calculus with primitives for constructing nested types directly as fixpoints. Our calculus can express all nested types appearing in the literature, including truly nested types. At the term level, it supports primitive pattern matching, map functions, and fold combinators for nested types. Our main contribution is the construction of a parametric model for our calculus. This is both delicate and challenging: to ensure the existence of semantic fixpoints interpreting nested types, and thus to establish a suitable Identity Extension Lemma for our calculus, our type system must explicitly track functoriality of types, and cocontinuity conditions on the functors interpreting them must be appropriately threaded throughout the model construction. We prove that our model satisfies an appropriate Abstraction Theorem and verifies all standard consequences of parametricity for primitive nested types.

1 Introduction

Algebraic data types (ADTs), both built-in and user-defined, have long been at the core of functional languages such as Haskell, ML, Agda, Epigram, and Idris. ADTs, such as that of natural numbers, can be unindexed. But they can also be indexed over other types. For example, the ADT of lists (here coded in Agda)

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

is indexed over its element type A . The instance of `List` at index A depends only on itself, and so is independent of `List B` for any other index B . That is, `List`, like all other ADTs, defines a *family of inductive types*, one for each index type.

Over time, there has been a notable trend toward data types whose non-regular indexing can capture invariants and other sophisticated properties that can be used for program verification and other applications. A simple example of such a type is given by Bird and Meertens' [4] prototypical nested type

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

of perfect trees, which can be thought of as constraining lists to have lengths that are powers of 2. The above code makes clear that perfect trees at index type A are defined in terms of perfect trees at index type $A \times A$. This is typical of nested types, one type instance of which can depend on others, so that the entire family

of types must actually be defined at once. A nested type thus defines not a family of inductive types, but rather an *inductive family of types*. Nested types include simple nested types, like perfect trees, none of whose recursive occurrences occur below another type constructor; “deep” nested types [18], such as the nested type

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

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

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

of bushes, whose recursive occurrences appear below their own type constructors.

Relational parametricity encodes a powerful notion of type-uniformity, or representation independence, for data types in polymorphic languages. It formalizes the intuition that a polymorphic program must act uniformly on all of its possible type instantiations by requiring that every such program preserves all relations between pairs of types at which it is instantiated. Parametricity was originally put forth by Reynolds [24] for System F [11], the calculus at the core of all polymorphic functional languages. It was later popularized as Wadler’s “theorems for free” [27], so called because it can deduce properties of programs in such languages solely from their types, i.e., with no knowledge whatsoever of the text of the programs involved. Most of Wadler’s free theorems are consequences of naturality for polymorphic list-processing functions. However, parametricity can also derive results that go beyond just naturality, such as correctness for ADTs of the program optimization known as *short cut fusion* [10,14].

But what about nested types? Does parametricity still hold if such types are added to polymorphic calculi? More practically, can we justifiably reason type-independently about (functions over) nested types in functional languages?

Type-independent reasoning about ADTs in functional languages is usually justified by first representing ADTs by their Church encodings, and then reasoning type-independently about these encodings. This is typically justified by constructing a parametric model — i.e., a model in which polymorphic functions preserve relations *à la* Reynolds — for a suitable fragment of System F, demonstrating that an initial algebra exists for the positive type constructor corresponding to the functor underlying an ADT of interest, and showing that each such initial algebra is suitably isomorphic to its corresponding Church encoding. In fact, this isomorphism of initial algebras and their Church encodings is one of the “litmus tests” for the goodness of a parametric model.

This approach works well for ADTs, which are always fixpoints of *first-order* functors, and whose Church encodings, which involve quantification over only type variables, are always expressible in System F. For example, `List A` is the fixpoint of the first-order functor $F X = 1 + A \times X$ and has Church encoding $\forall \alpha. \alpha \rightarrow (A \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha$. But despite Cardelli’s [7] claim that “virtually any basic type of interest can be encoded within F_2 ” — i.e., within System



F — non-ADT nested types cannot. Not even our prototypical nested type of perfect trees has a Church encoding expressible in System F! Indeed, $\mathbf{PTreeA}$ cannot be represented as the fixpoint of any *first-order* functor. However, it can be seen as the instance at index $\mathbf{A}$ of the fixpoint of the *higher-order* functor $HFA = (A \rightarrow FA) \rightarrow (F(A \times A) \rightarrow FA) \rightarrow FA$. It thus has Church encoding $\forall f. (\forall \alpha. \alpha \rightarrow f\alpha) \rightarrow (\forall \alpha. f(\alpha \times \alpha) \rightarrow f\alpha) \rightarrow \forall \alpha. f\alpha$, which requires quantification at the higher kind $* \rightarrow *$ for f . A similar situation obtains for any (non-ADT) nested type. Unfortunately, higher-kinded quantification is not available in System F, so if we want to reason type-independently about nested types in a language based on it we have only two options: *i*) move to an extension of System F, such as the higher-kinded calculus F_ω or a dependent type theory, and reason via their Church encodings in a known parametric model for that extension, or *ii*) add nested types to System F as primitives — i.e., as primitive type-level fixpoints — and construct a parametric model for the result.

Since the type systems of F_ω and dependent type theories are designed to extend System F with far more than non-ADT data types, it seems like serious overkill to pass to their parametric models to reason about nested types in System F. Indeed, such calculi support fundamentally new features that add complexity to their models that is entirely unnecessary for reasoning about nested types. This paper therefore pursues the second option above. We first design a Hindley-Milner-style calculus supporting primitive nested types, together with primitive types of natural transformations representing morphisms between them. Our calculus can express all nested types appearing in the literature, including truly nested types. At the term-level, it supports primitive pattern matching, map functions, and fold combinators for nested types.¹ Our main contribution is the construction of a parametric model for our calculus. This is both delicate and challenging. To ensure the existence of semantic fixpoints interpreting nested types, and thus to establish a suitable Identity Extension Lemma, our type system must explicitly track functoriality of types, and co-continuity conditions on the functors interpreting them must be appropriately threaded throughout the model construction. Our model validates all standard consequences of parametricity in the presence of primitive nested types, including the isomorphism of primitive ADTs and their Church encodings, and correctness of short cut fusion for nested types. 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 consequences of parametricity that follow from naturality, from those, such as short cut fusion for nested types, that require the full power of parametricity.

¹ We leave incorporating general term-level recursion to future work because, as Pitts [23] reminds us, “it is hard to construct models of both impredicative polymorphism and fixpoint recursion”. In fact, as the development in this paper shows, constructing a parametric model even for our predicative calculus with primitive nested types — and even without term-level fixpoints — is already rather involved. On the other hand, our calculus is strongly normalizing, so it perhaps edges us toward the kind of provably total practical programming language proposed in [27].



Structure of this Paper We introduce our calculus in Section 2. Its type system is based on the level-2-truncation of the higher-kinded grammar from [17], augmented with a primitive type of natural transformations. (Since [17] contains no term calculus, the issue of parametricity could not even be raised there.) In Section 3 we give set and relational interpretations of our types. Set interpretations are possible precisely because our calculus is predicative — as ensured by our primitive natural transformation types — and [17] guarantees that local finite presentability of **Set** makes it suitable for interpreting nested types. As is standard in categorical models, types are interpreted as functors from environments interpreting their type variable contexts to sets or relations, as appropriate. To ensure that these functors satisfy the cocontinuity properties needed for the semantic fixpoints interpreting nested types to exist, set environments must map k -ary type constructor variables to appropriately cocontinuous k -ary functors on sets, relation environments must map k -ary type constructor variables to appropriately cocontinuous k -ary relation transformers, and these cocontinuity conditions must be threaded through our type interpretations in such a way that an Identity Extension Lemma (Theorem 1) can be proved. Properly propagating the cocontinuity conditions requires considerable care, and Section 4, where it is done, is (apart from tracking functoriality in the calculus so that it is actually possible) where the bulk of the work in constructing our model lies.

In Section 5, we give set and relational interpretations for the terms of our calculus. As usual in categorical models, terms are interpreted as natural transformations from interpretations of their term contexts to interpretations of their types, and these must cohere in what is essentially a fibred way. In Section 6.1 we prove a scheme deriving free theorems that are consequences of naturality of polymorphic functions over nested types. This scheme is very general, and is parameterized over both the data type and the type of the polymorphic function at hand. It has, for example, analogues for nested types of Wadler’s map-rearrangement free theorems as instances. In Section 6.2 we prove that our model satisfies an Abstraction Theorem (Theorem 4), which we use to derive other parametricity results that go beyond naturality. We conclude in Section 7.

Related Work There is a long line of work on categorical models of parametricity for System F; see, e.g., [3,6,8,9,12,13,20,26]. To our knowledge, all such models treat ADTs via their Church encodings, verifying in the just-constructed parametric model that each ADT is isomorphic to its encoding. This paper draws on this rich tradition of categorical models of parametricity for System F, but modifies them to treat nested types (and thus ADTs) as primitive data types. The only other extensions we know of System F with primitive data types are those in [19,21,22,23,27]. Wadler [27] treats full System F, and sketches parametricity for its extension with lists. Martin and Gibbons [21] outline a semantics for a grammar of primitive nested types similar to that in [17], but treat only polynomial nested types. Unfortunately, the model suggested in [21] is not entirely correct (see [17]), and parametricity is nowhere mentioned. Matthes [19] treats System F with non-polynomial ADTs and nested types, but focuses on expressivity of generalized Mendler iteration for them. He gives no semantics.



In [23], Pitts adds list ADTs to full System F with a term-level fixpoint primitive. Other ADTs are included in [22], but nested types are not expressible in either syntax. Pitts constructs parametric models for his calculi based on operational, rather than categorical, semantics. A benefit of using operational semantics to build parametric models is that it avoids needing to work in a suitable metatheory to accommodate System F's impredicativity. It is well-known that there are no set-based parametric models of System F [25], so parametric models for it and its extensions are often constructed in a syntactic metatheory such as the impredicative Calculus of Inductive Constructions (iCIC). By adding primitive nested types to a Hindley-Milner-style calculus and working in a categorical setting we side-step such metatheoretic distractions. It is important to note that different consequences of parametricity are available in syntactic and semantic metatheories. Consequences of parametricity are possible for both closed and open System F terms in a syntactic metatheory — although not all that can be formulated can be always proved; see, e.g., the end of Section 7 of [4]. By contrast, in a categorical metatheory consequences of parametricity are expressible only for *closed* terms. For this reason, validating the standard consequences of parametricity for closed terms is — going all the way back to Reynolds [24] — all that is required for a model of parametricity to be considered good.

Atkey [2] treats parametricity for arbitrary higher kinds, constructing a parametric model for System F_ω within iCIC, rather than in a semantic category. His construction is in some ways similar to ours, but he represents (now higher-kinded) data types using Church encodings rather than as primitives. Moreover, the *fmap* functions associated to Atkey's functors must be *given*, presumably by the programmer, together with their underlying type constructors. This absolves him of imposing cocontinuity conditions on his model to ensure that fixpoints of his functors exist, but, unfortunately, he does not indicate which type constructors support *fmap* functions. We suspect explicitly spelling out which types can be interpreted as strictly positive functors would result in a full higher-kinded extension of a calculus akin to that 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* (i.e., of kind $* \rightarrow \dots \rightarrow * \rightarrow *$, with k arrows and $k+1$ $*$ s in this sequence) 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. Letters from the beginning of the alphabet denote type variables of arity 0, i.e., elements of $\mathbb{T}^0 \cup \mathbb{F}^0$. We write $\bar{\phi}$ for either a set $\{\phi_1, \dots, \phi_n\}$ of type constructor variables or a set of functorial variables when the cardinality n of the set is unimportant or clear from context. If V is a set of type variables we write $V, \bar{\phi}$ for $V \cup \bar{\phi}$ when $V \cap \bar{\phi} = \emptyset$. We omit the vector notation for a singleton set, thus writing ϕ , instead of $\bar{\phi}$, for $\{\phi\}$.

If Γ is a finite subset of $\mathbb{T}$, Φ is a finite subset of $\mathbb{F}$, $\bar{\alpha}$ is a finite subset of $\mathbb{F}^0$ disjoint from Φ , and $\phi^k \in \mathbb{F}^k \setminus \Phi$, then the set $\mathcal{F}$ of well-formed types is given in Definition 1. The notation there entails that type application $\phi F_1 \dots F_k$ is allowed only when ϕ is a type variable of arity k , or ϕ is a subexpression of the form $\mu\psi^k.\lambda\alpha_1 \dots \alpha_k.F'$. Moreover, if ϕ has arity k then ϕ 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. Requiring that types are always in such η -long normal form avoids having to consider β -conversion of types. In a subexpression $\text{Nat}^{\bar{\alpha}} F G$, the Nat operator binds all occurrences of the variables in $\bar{\alpha}$ in F and G ; intuitively, $\text{Nat}^{\bar{\alpha}} F G$ represents the type of a natural transformation in $\bar{\alpha}$ from the functor F to the functor G . In a subexpression $\mu\phi^k.\lambda\bar{\alpha}.F$, the μ operator binds all occurrences of the variable ϕ , and the λ operator binds all occurrences of the variables in $\bar{\alpha}$, in the body F .

A *type constructor*, or *non-functorial context* is a finite set Γ of type constructor variables, and a *functorial context* is a finite set Φ of functorial variables. In Definition 1, 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 1. *The formation rules for the set $\mathcal{F}$ of (well-formed) types are*

$$\begin{array}{c}
\frac{}{\Gamma; \Phi \vdash 0} \quad \frac{}{\Gamma; \Phi \vdash 1} \quad \frac{\Gamma; \Phi \vdash F \quad \Gamma; \Phi \vdash G}{\Gamma; \Phi \vdash F + G} \quad \frac{\Gamma; \Phi \vdash F \quad \Gamma; \Phi \vdash G}{\Gamma; \Phi \vdash F \times G} \\
\\
\frac{\Gamma; \bar{\alpha}^0 \vdash F \quad \Gamma; \bar{\alpha}^0 \vdash G}{\Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}^0} F G} \quad \frac{\phi^k \in \Gamma \cup \Phi \quad \overline{\Gamma; \Phi \vdash F}}{\Gamma; \Phi \vdash \phi^k \overline{F}} \\
\\
\frac{\Gamma; \bar{\alpha}^0, \phi^k \vdash F \quad \overline{\Gamma; \Phi \vdash G}}{\Gamma; \Phi \vdash (\mu\phi^k.\lambda\bar{\alpha}^0.F) \overline{G}}
\end{array}$$

We write $\vdash F$ for $\emptyset; \emptyset \vdash F$. Definition 1 ensures that the expected weakening rules for well-formed types hold (but 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 $\Gamma; \emptyset \vdash \text{Nat}^{\emptyset} F G$, which represents the arrow type $\Gamma \vdash F \rightarrow G$ in our calculus. The type $\Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} 1 F$ represents the $\forall$ -type $\Gamma; \emptyset \vdash \forall \bar{\alpha}. F$. Some System F types, such as $\forall \alpha. (\alpha \rightarrow \alpha) \rightarrow \alpha$, are not representable in our calculus.

Since the body F of a type $(\mu\phi.\lambda\bar{\alpha}.F)\overline{G}$ can only be functorial in ϕ and the variables in $\bar{\alpha}$, the representation of $\text{List } \alpha$ as the ADT $\mu\beta. 1 + \alpha \times \beta$ cannot be functorial in α . By contrast, if $\text{List } \alpha$ is represented as the nested type $(\mu\phi.\lambda\beta. 1 + \beta \times \phi\beta) \alpha$ then we can choose α to be a functorial variable or not when forming the type. This observation holds for other ADTs as well; for example, if $\text{Tree } \alpha \gamma = \mu\beta. \alpha + \beta \times \gamma \times \beta$, then $\alpha, \gamma; \emptyset \vdash \text{Tree } \alpha \gamma$ is well-formed, but $\emptyset; \alpha, \gamma \vdash \text{Tree } \alpha \gamma$ is not. It also applies to some non-ADT types, such as $\text{GRose } \phi \alpha = \mu\beta. 1 + \alpha \times \phi\beta$, in which ϕ and α must both be non-functorial variables. It is in fact possible to allow “extra” 0-ary functorial variables in the body of μ -types (functorial variables of higher arity are the real problem). This would allow the first-order representations of ADTs to be functorial, but doing so requires some changes to the formation rule for μ -types, as well as the delicate threading of some additional



conditions throughout our model construction. But since we can always use an ADT's (semantically equivalent) second-order representation when functoriality is needed, disallowing such “extra” variables does not negatively impact the expressivity of our calculus. We therefore pursue the simpler syntax here.

Definition 1 allows well-formed types to be functorial in no variables. Functorial variables can also be demoted to non-functorial status: if $F[\phi := \psi]$ is the textual replacement of ϕ in F , then $\Gamma, \psi^k; \Phi \vdash F[\phi^k := \psi^k]$ is derivable whenever $\Gamma; \Phi, \phi^k \vdash F$ is. In addition to textual replacement, we also have substitution for types. If $\Gamma; \Phi \vdash F$ is a type, if Γ and Φ 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 $F[\alpha_1 := F_1, \dots, \alpha_k := F_k]$, or $F[\alpha := \overline{F}]$ when convenient, for $F[\alpha_1 := F_1][\alpha_2 := F_2, \dots, \alpha_k := F_k]$. The operation $(\cdot)[\phi :=_{\overline{\alpha}} F]$ of *second-order type substitution along $\overline{\alpha}$* is defined by induction on types exactly as expected. The only interesting clause is that for type application, which defines $(\psi \overline{G})[\phi :=_{\overline{\alpha}} F]$ to be $F[\alpha := \overline{G[\phi :=_{\overline{\alpha}} F]}]$ if $\psi = \phi$ and $\overline{G[\phi :=_{\overline{\alpha}} F]}$ otherwise. Of course, $(\cdot)[\phi^0 :=_{\emptyset} F]$ coincides with first-order substitution. We omit $\overline{\alpha}$ when convenient, but note that it is not correct to substitute along non-functorial variables. It is not hard to see that if $\Gamma; \Phi, \phi^k \vdash H$ and $\Gamma; \Phi, \overline{\alpha} \vdash F$ 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 $\Gamma, \overline{\psi}; \Phi \vdash H[\phi :=_{\overline{\alpha}} F[\psi := \overline{\psi}]]$.

2.2 Terms

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 above conventions for disjoint unions and vectors in term contexts. If Δ is a term context for Γ and Φ then the formation rules for the set of *well-formed terms over Δ* are given in Figure 1. An expression $L_{\overline{\alpha}} x. t$ binds all occurrences of the type variables in $\overline{\alpha}$ in the types of x and 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 variable in $\overline{\alpha}$.

In the rule for $\text{map}_H^{\overline{F}, \overline{G}}$ there is one functorial expression in $\overline{F}$ and one functorial expression in $\overline{G}$ for each variable in $\overline{\phi}$. Moreover, for each ϕ^k in $\overline{\phi}$ the number of variables in $\overline{\beta}$ in the judgments for functorial expressions in $\overline{F}$ and $\overline{G}$ is k . In the rules for in_H and fold_H^F , the variables in $\overline{\beta}$ are fresh with respect to H , and there is one β for every α . Substitution for terms is the obvious extension of the usual capture-avoiding textual substitution, and weakening is respected.

The “extra” functorial variables in $\overline{\gamma}$ in the rules for $\text{map}_H^{\overline{F}, \overline{G}}$ (i.e., those variables not affected by the substitution of ϕ) allow us to map polymorphic functions over nested types. Suppose, for example, that we want to map the polymorphic function $\text{flatten} : \text{Nat}^\beta(\text{PTree } \beta)(\text{List } \beta)$ over lists. The map term for this is typeable as follows:

$$\frac{\Gamma; \alpha, \gamma \vdash \text{List } \alpha \quad \Gamma; \gamma \vdash \text{PTree } \gamma \quad \Gamma; \gamma \vdash \text{List } \gamma}{\Gamma; \emptyset \mid \emptyset \vdash \text{map}_{\text{List } \alpha}^{\text{PTree } \gamma, \text{List } \gamma} : \text{Nat}^\emptyset(\text{Nat}^\gamma(\text{PTree } \gamma)(\text{List } \gamma))(\text{Nat}^\gamma(\text{List } (\text{PTree } \gamma))(\text{List } (\text{List } \gamma)))}$$

$$\begin{array}{c}
\frac{\Gamma; \Phi \vdash F}{\Gamma; \Phi \mid \Delta, x : F \vdash x : F} \quad \frac{\Gamma; \Phi \mid \Delta \vdash t : 0 \quad \Gamma; \Phi \vdash F}{\Gamma; \Phi \mid \Delta \vdash \perp_F t : F} \quad \frac{\Gamma; \Phi \vdash F}{\Gamma; \Phi \mid \Delta \vdash \top : 1} \\
\frac{\Gamma; \Phi \mid \Delta \vdash s : F}{\Gamma; \Phi \mid \Delta \vdash \text{inL } s : F + G} \quad \frac{\Gamma; \Phi \mid \Delta \vdash t : G}{\Gamma; \Phi \mid \Delta \vdash \text{inR } t : F + G} \\
\frac{\Gamma; \Phi \vdash F, G \quad \Gamma; \Phi \mid \Delta \vdash t : F + G \quad \Gamma; \Phi \mid \Delta, x : F \vdash l : K \quad \Gamma; \Phi \mid \Delta, y : G \vdash r : K}{\Gamma; \Phi \mid \Delta \vdash \text{case } t \text{ of } \{x \mapsto l; y \mapsto r\} : K} \\
\frac{\Gamma; \Phi \mid \Delta \vdash s : F \quad \Gamma; \Phi \mid \Delta \vdash t : G}{\Gamma; \Phi \mid \Delta \vdash (s, t) : F \times G} \quad \frac{\Gamma; \Phi \mid \Delta \vdash t : F \times G}{\Gamma; \Phi \mid \Delta \vdash \pi_1 t : F} \quad \frac{\Gamma; \Phi \mid \Delta \vdash t : F \times G}{\Gamma; \Phi \mid \Delta \vdash \pi_2 t : G} \\
\frac{\Gamma; \bar{\alpha} \vdash F \quad \Gamma; \bar{\alpha} \vdash G \quad \Gamma; \bar{\alpha} \mid \Delta, x : F \vdash t : G}{\Gamma; \emptyset \mid \Delta \vdash L_{\bar{\alpha}x}.t : \text{Nat}^{\bar{\alpha}} F G} \\
\frac{\Gamma; \Phi \vdash \bar{K} \quad \Gamma; \emptyset \mid \Delta \vdash t : \text{Nat}^{\bar{\alpha}} F G \quad \Gamma; \Phi \mid \Delta \vdash s : F[\bar{\alpha} := \bar{K}]}{\Gamma; \Phi \mid \Delta \vdash t_{\bar{K}} s : G[\bar{\alpha} := \bar{K}]} \\
\frac{\Gamma; \bar{\phi}, \bar{\gamma} \vdash H \quad \Gamma; \bar{\beta}, \bar{\gamma} \vdash F \quad \Gamma; \bar{\beta}, \bar{\gamma} \vdash G}{\Gamma; \emptyset \mid \emptyset \vdash \text{map}_{\bar{H}}^{\bar{F}, \bar{G}} : \text{Nat}^{\emptyset} (\text{Nat}^{\bar{\beta}, \bar{\gamma}} F G) (\text{Nat}^{\bar{\gamma}} H[\bar{\phi} :=_{\bar{\beta}} \bar{F}] H[\bar{\phi} :=_{\bar{\beta}} \bar{G}])} \\
\frac{\Gamma; \phi, \bar{\alpha} \vdash H}{\Gamma; \emptyset \mid \emptyset \vdash \text{in}_H : \text{Nat}^{\bar{\beta}} H[\phi :=_{\bar{\beta}} (\mu\phi. \lambda\bar{\alpha}. H)] \bar{\beta} [\bar{\alpha} := \bar{\beta}] (\mu\phi. \lambda\bar{\alpha}. H) \bar{\beta}} \\
\frac{\Gamma; \phi, \bar{\alpha} \vdash H \quad \Gamma; \bar{\beta} \vdash F}{\Gamma; \emptyset \mid \emptyset \vdash \text{fold}_H^F : \text{Nat}^{\emptyset} (\text{Nat}^{\bar{\beta}} H[\phi :=_{\bar{\beta}} F] [\bar{\alpha} := \bar{\beta}] F) (\text{Nat}^{\bar{\beta}} (\mu\phi. \lambda\bar{\alpha}. H) \bar{\beta} F)}
\end{array}$$

Fig. 1. Well-formed terms

However, this derivation would not be possible without the “extra” variable γ .

Our calculus is expressive enough to define, e.g., a function $\text{reversePTree} : \text{Nat}^{\alpha} (PTree \alpha) (PTree \alpha)$ that reverses the order of the leaves in a perfect tree. It maps the perfect tree $((1, 2), (3, 4))$ to $((4, 3), (2, 1))$. Unfortunately, we cannot define recursive functions — such as a concatenation function for perfect trees or a zip function for bushes — that take as inputs a nested type and an argument of another type, both of which are parameterized over the same variable. The fundamental issue is that recursion is expressible only via fold , which produces natural transformations in some variables $\bar{\alpha}$ from μ -types to other functors F . The restrictions on Nat -types entail that F cannot itself be a Nat -type containing $\bar{\alpha}$, so, e.g., $\text{Nat}^{\alpha} (PTree \alpha) (\text{Nat}^{\emptyset} (PTree \alpha) (PTree (\alpha \times \alpha)))$ is not well-typed. Uncurrying gives $\text{Nat}^{\alpha} (PTree \alpha \times PTree \alpha) (PTree (\alpha \times \alpha))$, which is well-typed, but fold cannot produce a term of this type because $PTree \alpha \times PTree \alpha$ is not a μ -type. Our calculus can, however, express types of recursive functions that take multiple nested types as arguments, provided they 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. Even for ADTs there is a difference between which folds over them we can type when they are viewed as ADTs (i.e., as fixpoints of first-order functors) versus as proper nested types (i.e., as fixpoints of higher-order functors). This is because, in the return type of fold , the arguments of the μ -type must be variables



bound by Nat . For ADTs, the μ -type takes no arguments, making it possible to write recursive functions, such as a concatenation function for lists of type $\alpha; \emptyset \vdash \text{Nat}^0(\mu\beta.1 + \alpha \times \beta)(\text{Nat}^0(\mu\beta.1 + \alpha \times \beta)(\mu\beta.1 + \alpha \times \beta))$. This is not possible for nested types — even when they are semantically equivalent to ADTs.

Interestingly, even some recursive functions of a single proper nested type — e.g., a reverse function for bushes 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 form as the type of, e.g., a zip function for perfect trees. Expressivity of folds for nested types has long been a vexing issue, and this is naturally inherited by our calculus. Adding more expressive recursion combinators — e.g., generalized folds or Mendler iterators — could help, but since this is orthogonal to the issue of parametricity in the presence of primitive nested types we do not consider it further here.

3 Interpreting Types

We denote the category of sets and functions by Set . The category Rel has as objects triples (A, B, R) , where R is a relation between sets A and B . It has as morphisms from (A, B, R) to (A', B', R') pairs $(f : A \rightarrow A', g : B \rightarrow B')$ of morphisms in Set such that $(fa, gb) \in R'$ if $(a, b) \in R$. We may write $R : \text{Rel}(A, B)$ for (A, B, R) . If $R : \text{Rel}(A, B)$ we write $\pi_1 R$ and $\pi_2 R$ for the *domain* A of R and the *codomain* B of R , respectively, and assume π_1 and π_2 are surjective. We write $\text{Eq}_A = (A, A, \{(x, x) \mid x \in A\})$ for the *equality relation* on the set A .

The key idea underlying Reynolds' parametricity is to give each type $F(\alpha)$ with one free variable α a *set interpretation* F_0 taking sets to sets and a *relational interpretation* F_1 taking relations $R : \text{Rel}(A, B)$ to relations $F_1(R) : \text{Rel}(F_0(A), F_0(B))$, 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 $t_0(A) : G_0(A) \rightarrow F_0(A)$. These interpretations are 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 $F_1(\text{Eq}_A) = \text{Eq}_{F_0(A)}$, 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 $R : \text{Rel}(A, B)$, $(t_0(A), t_0(B))$ 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 interpret terms. Similar theorems are required for types and terms with any number of free variables.

The key to proving our Identity Extension Lemma is a familiar “cutting down” of the interpretations of universally quantified types to include only the “parametric” elements; the relevant types here are Nat types. This requires that the set interpretations of types (Section 3.1) are defined simultaneously with their relational interpretations (Section 3.2). While set interpretations are relatively straightforward, relational interpretations are less so because of the co-continuity conditions needed to know they are well-defined. We develop these conditions in Sections 3.1 and 3.2. This separates our set and relational interpretations in space, but has no other impact on the mutually inductive definitions.

$$\begin{aligned}
& \llbracket \Gamma; \Phi \vdash 0 \rrbracket^{\text{Set}} \rho = 0 \\
& \llbracket \Gamma; \Phi \vdash 1 \rrbracket^{\text{Set}} \rho = 1 \\
& \llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}} \rho = \{ \eta : \lambda \bar{A}. \llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{A}] \Rightarrow \lambda \bar{A}. \llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{A}] \\
& \quad | \forall \bar{A}, \bar{B} : \text{Set}. \forall \bar{R} : \text{Rel}(\bar{A}, \bar{B}). \\
& \quad (\eta_{\bar{A}}, \eta_{\bar{B}}) : \llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Rel}} \text{Eq}_{\rho}[\bar{\alpha} := \bar{R}] \rightarrow \llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Rel}} \text{Eq}_{\rho}[\bar{\alpha} := \bar{R}] \} \\
& \llbracket \Gamma; \Phi \vdash \phi \bar{F} \rrbracket^{\text{Set}} \rho = (\rho \phi) \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho \\
& \llbracket \Gamma; \Phi \vdash F + G \rrbracket^{\text{Set}} \rho = \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho + \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho \\
& \llbracket \Gamma; \Phi \vdash F \times G \rrbracket^{\text{Set}} \rho = \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho \times \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho \\
& \llbracket \Gamma; \Phi \vdash (\mu \phi. \lambda \bar{\alpha}. H) \bar{G} \rrbracket^{\text{Set}} \rho = (\mu T_{H, \rho}^{\text{Set}}) \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho \\
& \quad \text{where } T_{H, \rho}^{\text{Set}} F = \lambda \bar{A}. \llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Set}} \rho[\phi := F][\bar{\alpha} := \bar{A}] \\
& \quad \text{and } T_{H, \rho}^{\text{Set}} \eta = \lambda \bar{A}. \llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Set}} \text{id}_{\rho}[\phi := \eta][\bar{\alpha} := \text{id}_{\bar{A}}]
\end{aligned}$$

Fig. 2. Set interpretation

3.1 Interpreting Types as Sets

We interpret types in our calculus as ω -cocontinuous functors on locally finitely presentable categories [1]. 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 1 are well-defined [17]. To bootstrap this process, we interpret type variables as ω -cocontinuous functors. 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 $\mathcal{D}$.

A *set environment* maps each type variable in $\mathbb{T}^k \cup \mathbb{F}^k$ to an element of $[\text{Set}^k, \text{Set}]$. A morphism $f : \rho \rightarrow \rho'$ 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 componentwise, with the identity morphism mapping each one to itself. This gives a category of set environments and morphisms between them, denoted **SetEnv**. We identify a functor in $[\text{Set}^0, \text{Set}]$ with its value on $*$, and consider a set environment to map a type variable of arity 0 to a set. If $\bar{\alpha} = \{\alpha_1, \dots, \alpha_k\}$ and $\bar{A} = \{A_1, \dots, A_k\}$, then we write $\rho[\bar{\alpha} := \bar{A}]$ for the set environment ρ' such that $\rho'\alpha_i = A_i$ for $i = 1, \dots, k$ and $\rho'\alpha = \rho\alpha$ if $\alpha \notin \{\alpha_1, \dots, \alpha_k\}$. If $\rho \in \text{SetEnv}$ we write Eq_{ρ} for the relation environment (see Section 3) such that $\text{Eq}_{\rho} v = \text{Eq}_{\rho v}$ for every type variable v . The *set interpretation* $\llbracket \cdot \rrbracket^{\text{Set}} : \mathcal{F} \rightarrow [\text{SetEnv}, \text{Set}]$ is defined in Figure 2. The relational interpretations in the second clause of Figure 2 are given in full in Figure 3.

If $\rho \in \text{SetEnv}$ and $\vdash F$ we write $\llbracket \vdash F \rrbracket^{\text{Set}}$ for $\llbracket \vdash F \rrbracket^{\text{Set}} \rho$ since the environment is immaterial. The third clause of Figure 2 does indeed define a set: local finite presentability of **Set** and ω -cocontinuity of $\llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Set}} \rho$ ensure that the set of natural transformations $\{ \eta : \llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Set}} \rho \Rightarrow \llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Set}} \rho \}$ (which contains

$\llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}} \rho$ is a subset of $\{(\llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{S}]) (\llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{S}]) \mid \bar{S} = (S_1, \dots, S_{|\bar{\alpha}|}), \text{ and } S_i \text{ is a finite set for } i = 1, \dots, |\bar{\alpha}|\}$. There are countably many tuples $\bar{S}$, each giving a morphism from $\llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{S}]$ to $\llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{S}]$, and only **Set**-many such morphisms since **Set** is locally small. In addition, $\llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}}$ is ω -cocontinuous since it is constant on ω -directed sets. Interpretations of **Nat** types ensure that $\llbracket \Gamma \vdash F \rightarrow G \rrbracket^{\text{Set}}$ and $\llbracket \Gamma \vdash \forall \bar{\alpha}. F \rrbracket^{\text{Set}}$ are as expected in parametric models.

To make sense of the last clause in Figure 2, we need to know that, for each $\rho \in \text{SetEnv}$, $T_{H,\rho}^{\text{Set}}$ is an ω -cocontinuous endofunctor on $[\text{Set}^k, \text{Set}]$, and thus admits a fixpoint. Since $T_{H,\rho}^{\text{Set}}$ is defined in terms of $\llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Set}}$, interpretations of types must be such functors, which entails that the actions of set interpretations of types on objects and on morphisms in **SetEnv** are intertwined. We know from [17] that, for every $\Gamma; \bar{\alpha} \vdash G$, $\llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Set}}$ is actually in $[\text{Set}^k, \text{Set}]$ where $k = |\bar{\alpha}|$, so that, for each $\llbracket \Gamma; \phi^k, \bar{\alpha} \vdash H \rrbracket^{\text{Set}}$, the corresponding operator T_H^{Set} can be extended to a functor from **SetEnv** to $[[\text{Set}^k, \text{Set}], [\text{Set}^k, \text{Set}]]$. The action of T_H^{Set} on an object $\rho \in \text{SetEnv}$ is given by the higher-order functor $T_{H,\rho}^{\text{Set}}$, whose actions on objects (functors in $[\text{Set}^k, \text{Set}]$) and morphisms between them are given in Figure 2. Its action on a morphism $f : \rho \rightarrow \rho'$ is the higher-order natural transformation $T_{H,f}^{\text{Set}} : T_{H,\rho}^{\text{Set}} \rightarrow T_{H,\rho'}^{\text{Set}}$, whose action on $F : [\text{Set}^k, \text{Set}]$ is the natural transformation $T_{H,f}^{\text{Set}} F : T_{H,\rho}^{\text{Set}} F \rightarrow T_{H,\rho'}^{\text{Set}} F$ whose component at $\bar{A}$ is $(T_{H,f}^{\text{Set}} F)_{\bar{A}} = \llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Set}} f[\phi := id_F][\bar{\alpha} := id_{\bar{A}}]$. The next definition uses T_H^{Set} to define the functorial action of set interpretation.

Definition 2. The action of $\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}}$ on $f : \rho \rightarrow \rho'$ in **SetEnv** is given by:

- $\llbracket \Gamma; \Phi \vdash 0 \rrbracket^{\text{Set}} f = id_0$
- $\llbracket \Gamma; \Phi \vdash 1 \rrbracket^{\text{Set}} f = id_1$
- $\llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}} f = id_{\llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}} \rho}$
- $\llbracket \Gamma; \Phi \vdash \phi \bar{F} \rrbracket^{\text{Set}} f : \llbracket \Gamma; \Phi \vdash \phi \bar{F} \rrbracket^{\text{Set}} \rho \rightarrow \llbracket \Gamma; \Phi \vdash \phi \bar{F} \rrbracket^{\text{Set}} \rho' = (\rho\phi) \overline{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho} \rightarrow (\rho'\phi) \overline{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho'}$ is defined by $\llbracket \Gamma; \Phi \vdash \phi \bar{F} \rrbracket^{\text{Set}} f = (f\phi) \overline{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho} \circ (\rho\phi) \overline{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho} = (\rho'\phi) \overline{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} f} \circ (f\phi) \overline{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho}$. The latter equality holds because $\rho\phi$ and $\rho'\phi$ are functors and $f\phi : \rho\phi \rightarrow \rho'\phi$ is a natural transformation.
- $\llbracket \Gamma; \Phi \vdash F + G \rrbracket^{\text{Set}} f$ is defined by $\llbracket \Gamma; \Phi \vdash F + G \rrbracket^{\text{Set}} f(\text{inL } x) = \text{inL } (\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} f x)$ and $\llbracket \Gamma; \Phi \vdash F + G \rrbracket^{\text{Set}} f(\text{inR } y) = \text{inR } (\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} f y)$
- $\llbracket \Gamma; \Phi \vdash F \times G \rrbracket^{\text{Set}} f = \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} f \times \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} f$
- $\llbracket \Gamma; \Phi \vdash (\mu\phi. \lambda\bar{\alpha}. H) \bar{G} \rrbracket^{\text{Set}} f : \llbracket \Gamma; \Phi \vdash (\mu\phi. \lambda\bar{\alpha}. H) \bar{G} \rrbracket^{\text{Set}} \rho \rightarrow \llbracket \Gamma; \Phi \vdash (\mu\phi. \lambda\bar{\alpha}. H) \bar{G} \rrbracket^{\text{Set}} \rho' = (\mu T_{H,\rho}^{\text{Set}}) \overline{\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho} \rightarrow (\mu T_{H,\rho'}^{\text{Set}}) \overline{\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho'}$ is defined by $(\mu T_{H,f}^{\text{Set}}) \overline{\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho'}$ $(\mu T_{H,\rho}^{\text{Set}}) \overline{\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} f} = (\mu T_{H,\rho'}^{\text{Set}}) \overline{\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} f} \circ (\mu T_{H,f}^{\text{Set}}) \overline{\llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Set}} \rho}$. The latter equality holds because $\mu T_{H,\rho}^{\text{Set}}$ and $\mu T_{H,\rho'}^{\text{Set}}$ are functors and $\mu T_{H,f}^{\text{Set}} : \mu T_{H,\rho}^{\text{Set}} \rightarrow \mu T_{H,\rho'}^{\text{Set}}$ is a natural transformation.

3.2 Interpreting Types as Relations

A k -ary relation transformer F is a triple (F^1, F^2, F^*) , where $F^1, F^2 : [\text{Set}^k, \text{Set}]$ and $F^* : [\text{Rel}^k, \text{Rel}]$ are functors, if $R_i : \text{Rel}(A_i, B_i)$ for $i = 1, \dots, k$ then $F^* \bar{R} :$

$\text{Rel}(F^1\bar{A}, F^2\bar{B})$, and if $(\alpha_i, \beta_i) \in \text{Hom}_{\text{Rel}}(\bar{R}_i, \bar{S}_i)$ for $i = 1, \dots, k$, then $F^*(\alpha, \beta) = (F^1\bar{\alpha}, F^2\bar{\beta})$. We define $F^*\bar{R}$ to be $F^*\bar{R}$ and $F(\alpha, \beta)$ to be $F^*(\alpha, \beta)$. The last clause above expands to: if $(a, b) \in \bar{R}$ implies $(\alpha a, \beta b) \in \bar{S}$ then $(c, d) \in F^*\bar{R}$ implies $(F^1\bar{\alpha}c, F^2\bar{\beta}d) \in F^*\bar{S}$. We identify a 0-ary relation transformer (A, B, R) with $R : \text{Rel}(A, B)$, and write $\pi_1 F$ for F^1 and $\pi_2 F$ for F^2 . Below we extend these conventions to relation environments in the obvious ways.

The category RT_k of k -ary relation transformers is given by the following data: an object of RT_k is a k -ary relation transformer; a morphism $\delta : (G^1, G^2, G^*) \rightarrow (H^1, H^2, H^*)$ in RT_k is a pair of natural transformations (δ^1, δ^2) where $\delta^1 : G^1 \rightarrow H^1$, $\delta^2 : G^2 \rightarrow H^2$ such that, for all $\bar{R} : \text{Rel}(A, B)$, if $(x, y) \in G^*\bar{R}$ then $(\delta^1_{\bar{A}}x, \delta^2_{\bar{B}}y) \in H^*\bar{R}$; and identity morphisms and composition are inherited from the category of functors on Set . An endofunctor H on RT_k is a triple $H = (H^1, H^2, H^*)$, where H^1 and H^2 are functors from $[\text{Set}^k, \text{Set}]$ to $[\text{Set}^k, \text{Set}]$; H^* is a functor from RT_k to $[\text{Rel}^k, \text{Rel}]$; for all $\bar{R} : \text{Rel}(A, B)$, $\pi_1((H^*(\delta^1, \delta^2))_{\bar{R}}) = (H^1\delta^1)_{\bar{A}}$ and $\pi_2((H^*(\delta^1, \delta^2))_{\bar{R}}) = (H^2\delta^2)_{\bar{B}}$; the action of H on objects is given by $H(F^1, F^2, F^*) = (H^1F^1, H^2F^2, H^*(F^1, F^2, F^*))$; and the action of H on morphisms is given by $H(\delta^1, \delta^2) = (H^1\delta^1, H^2\delta^2)$ for $(\delta^1, \delta^2) : (F^1, F^2, F^*) \rightarrow (G^1, G^2, G^*)$. Since applying an endofunctor H to k -ary relation transformers and morphisms between them must give k -ary relation transformers and morphisms between them, this definition implicitly requires the following three conditions to hold: *i*) $H^*(F^1, F^2, F^*)\bar{R} : \text{Rel}(H^1F^1\bar{A}, H^2F^2\bar{B})$ whenever $R_1 : \text{Rel}(A_1, B_1), \dots, R_k : \text{Rel}(A_k, B_k)$; *ii*) $H^*(F^1, F^2, F^*)(\alpha, \beta) = (H^1F^1\bar{\alpha}, H^2F^2\bar{\beta})$ whenever $(\alpha_1, \beta_1) \in \text{Hom}_{\text{Rel}}(R_1, S_1), \dots, (\alpha_k, \beta_k) \in \text{Hom}_{\text{Rel}}(R_k, S_k)$; and *iii*) $(\delta^1, \delta^2) : (F^1, F^2, F^*) \rightarrow (G^1, G^2, G^*)$ and $R_1 : \text{Rel}(A_1, B_1), \dots, R_k : \text{Rel}(A_k, B_k)$, then $((H^1\delta^1)_{\bar{A}}x, (H^2\delta^2)_{\bar{B}}y) \in H^*(G^1, G^2, G^*)\bar{R}$ whenever $(x, y) \in H^*(F^1, F^2, F^*)\bar{R}$. Note, however, that this last condition is automatically satisfied because it is implied by the third condition on functors on relation transformers.

If H and K are endofunctors on RT_k , then a *natural transformation* $\sigma : H \rightarrow K$ is a pair $\sigma = (\sigma^1, \sigma^2)$, where $\sigma^1 : H^1 \rightarrow K^1$ and $\sigma^2 : H^2 \rightarrow K^2$ are natural transformations between endofunctors on $[\text{Set}^k, \text{Set}]$ and the component of σ at $F \in RT_k$ is given by $\sigma_F = (\sigma^1_{F^1}, \sigma^2_{F^2})$. This definition entails that $\sigma^i_{F^i}$ is natural in $F^i : [\text{Set}^k, \text{Set}]$, and, for every F , both $(\sigma^1_{F^1})_{\bar{A}}$ and $(\sigma^2_{F^2})_{\bar{B}}$ are natural in $\bar{A}$. Moreover, since the results of applying σ to k -ary relation transformers must be morphisms of k -ary relation transformers, it implicitly requires that $(\sigma_F)_{\bar{R}} = ((\sigma^1_{F^1})_{\bar{A}}, (\sigma^2_{F^2})_{\bar{B}})$ is a morphism in Rel for any k -tuple of relations $\bar{R} : \text{Rel}(A, B)$, i.e., that if $(x, y) \in H^*F\bar{R}$, then $((\sigma^1_{F^1})_{\bar{A}}x, (\sigma^2_{F^2})_{\bar{B}}y) \in K^*F\bar{R}$.

Critically, we can compute ω -directed colimits in RT_k . Indeed, if $\mathcal{D}$ is an ω -directed set then $\lim_{d \in \mathcal{D}} (F^1_d, F^2_d, F^*_d) = (\lim_{d \in \mathcal{D}} F^1_d, \lim_{d \in \mathcal{D}} F^2_d, \lim_{d \in \mathcal{D}} F^*_d)$. 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 $[\text{Set}^k, \text{Set}]$ and T^* is an ω -cocontinuous functor from RT_k to $[\text{Rel}^k, \text{Rel}]$, i.e., is in $[RT_k, [\text{Rel}^k, \text{Rel}]]$. Now, for any k , any $A : \text{Set}$, and any $R : \text{Rel}(A, B)$, 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 the initial object of either Set or Rel , as appropriate. Observing that, for every k , K^{Set}_0 is initial in $[\text{Set}^k, \text{Set}]$, and K^{Rel}_0 is initial in $[\text{Rel}^k, \text{Rel}]$,

we have that, for each k , $K_0 = (K_0^{\text{Set}}, K_0^{\text{Set}}, K_0^{\text{Rel}})$ is initial in RT_k . Thus, if $T = (T^1, T^2, T^*) : RT_k \rightarrow RT_k$ is an endofunctor on RT_k we can define the relation transformer μT to be $\lim_{\rightarrow n \in \mathbb{N}} T^n K_0 = (\mu T^1, \mu T^2, \lim_{\rightarrow n \in \mathbb{N}} (T^n K_0)^*)$. If $T : [RT_k, RT_k]$ then μT is a fixpoint for T , i.e., $\mu T \cong T(\mu T)$. The isomorphism is given by $(in_1, in_2) : T(\mu T) \rightarrow \mu T$ and $(in_1^{-1}, in_2^{-1}) : \mu T \rightarrow T(\mu T)$ in RT_k . The latter is always a morphism in RT_k , but the former need not be if T is not ω -cocontinuous. Since μT 's third component is the colimit in $[\text{Rel}^k, \text{Rel}]$ of third components of relation transformers, rather than a fixpoint of an endofunctor on $[\text{Rel}^k, \text{Rel}]$, there is an asymmetry between μT 's first two and third components.

A *relation environment* maps each type variable in $\mathbb{T}^k \cup \mathbb{F}^k$ to a k -ary relation transformer. A morphism $f : \rho \rightarrow \rho'$ between relation environments ρ and ρ' with $\rho|_{\mathbb{T}} = \rho'|_{\mathbb{T}}$ maps each $\psi^k \in \mathbb{T}$ to the identity morphism on $\rho\psi^k = \rho'\psi^k$ and each $\phi^k \in \mathbb{F}$ 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 componentwise, with the identity morphism mapping each to itself; this gives a category RelEnv of relation environments and their morphisms. We identify a 0-ary relation transformer with its codomain, and consider a relation environment to map a type variable of arity 0 to a relation. We write $\rho[\alpha := \bar{R}]$ for the relation environment ρ' such that $\rho'\alpha_i = R_i$ for $i = 1, \dots, k$ and $\rho'\alpha = \rho\alpha$ if $\alpha \notin \{\alpha_1, \dots, \alpha_k\}$. If $\rho \in \text{RelEnv}$ 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.

For each k , an ω -cocontinuous functor $H : [\text{RelEnv}, RT_k]$ is a triple $H = (H^1, H^2, H^*)$, where $H^1, H^2 : [\text{SetEnv}, [\text{Set}^k, \text{Set}]]$; $H^* : [\text{RelEnv}, [\text{Rel}^k, \text{Rel}]]$; for all $\bar{R} : \text{Rel}(A, B)$ and morphisms f in RelEnv , $\pi_1(H^*f\bar{R}) = H^1(\pi_1f)\bar{A}$ and $\pi_2(H^*f\bar{R}) = H^2(\pi_2f)\bar{B}$; the action of H on ρ in RelEnv is given by $H\rho = (H^1(\pi_1\rho), H^2(\pi_2\rho), H^*\rho)$; and the action of H on morphisms $f : \rho \rightarrow \rho'$ in RelEnv is given by $Hf = (H^1(\pi_1f), H^2(\pi_2f))$. The last two points above give: *i*) if $\bar{R}_i : \text{Rel}(A_i, B_i)$ for $i = 1, \dots, k$ then $H^*\rho\bar{R} : \text{Rel}(H^1(\pi_1\rho)\bar{A}, H^2(\pi_2\rho)\bar{B})$; *ii*) if $(\alpha_i, \beta_i) \in \text{Hom}_{\text{Rel}}(R_i, S_i)$ for $i = 1, \dots, k$ then $H^*\rho(\alpha, \beta) = (H^1(\pi_1\rho)\bar{\alpha}, H^2(\pi_2\rho)\bar{\beta})$; and *iii*) if $f : \rho \rightarrow \rho'$ and $\bar{R}_i : \text{Rel}(A_i, B_i)$ for $i = 1, \dots, k$, then if $(x, y) \in H^*\rho\bar{R}$ then $(H^1(\pi_1f)\bar{A}x, H^2(\pi_2f)\bar{B}y) \in H^*\rho'\bar{R}$.

Computation of ω -directed colimits in RT_k extends componentwise to colimits in RelEnv . Similarly, ω -cocontinuity for endofunctors on RT_k extends to functors from RelEnv to RT_k . Our relational interpretation $\llbracket \cdot \rrbracket^{\text{Rel}} : \mathcal{F} \rightarrow [\text{RelEnv}, \text{Rel}]$ is given in Figure 3. It ensures that $\llbracket \Gamma \vdash F \rightarrow G \rrbracket^{\text{Rel}}$ and $\llbracket \Gamma \vdash \forall \bar{\alpha}. F \rrbracket^{\text{Rel}}$ are as expected. As for set interpretations, $\llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Rel}}$ is ω -cocontinuous because it is constant on ω -directed sets. If $\rho \in \text{RelEnv}$ we write $\llbracket \vdash F \rrbracket^{\text{Rel}}$ for $\llbracket \vdash F \rrbracket^{\text{Rel}}\rho$. For the last clause in Figure 3 to be well-defined we need $T_{H,\rho}$ to be an ω -cocontinuous endofunctor on RT , so that it admits a fixpoint. Since $T_{H,\rho}$ is defined in terms of $\llbracket \Gamma; \phi^k, \bar{\alpha} \vdash H \rrbracket^{\text{Rel}}$, 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. We know from [17] that, for every $\Gamma; \bar{\alpha} \vdash F$, $\llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Rel}}$ is actually in $[\text{Rel}^k, \text{Rel}]$ where $k = |\bar{\alpha}|$. We first define the actions of each of these functors on morphisms between relation environments, and then

$$\begin{aligned}
& \llbracket \Gamma; \Phi \vdash 0 \rrbracket^{\text{Rel}} \rho = 0 \\
& \llbracket \Gamma; \Phi \vdash 1 \rrbracket^{\text{Rel}} \rho = 1 \\
& \llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Rel}} \rho = \{ \eta : \lambda \bar{R}. \llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Rel}} \rho[\bar{\alpha} := \bar{R}] \Rightarrow \lambda \bar{R}. \llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Rel}} \rho[\bar{\alpha} := \bar{R}] \} \\
& \quad = \{ (t, t') \in \llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}}(\pi_1 \rho) \times \llbracket \Gamma; \emptyset \vdash \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}}(\pi_2 \rho) \mid \\
& \quad \quad \forall R_1 : \text{Rel}(A_1, B_1) \dots R_k : \text{Rel}(A_k, B_k). \\
& \quad \quad (t_{\bar{A}}, t'_{\bar{B}}) \in (\llbracket \Gamma; \bar{\alpha} \vdash G \rrbracket^{\text{Rel}} \rho[\bar{\alpha} := \bar{R}])^{\llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Rel}} \rho[\bar{\alpha} := \bar{R}]} \} \\
& \llbracket \Gamma; \Phi \vdash \phi \bar{F} \rrbracket^{\text{Rel}} \rho = (\rho \phi) \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}} \rho \\
& \llbracket \Gamma; \Phi \vdash F + G \rrbracket^{\text{Rel}} \rho = \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}} \rho + \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Rel}} \rho \\
& \llbracket \Gamma; \Phi \vdash F \times G \rrbracket^{\text{Rel}} \rho = \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}} \rho \times \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Rel}} \rho \\
& \llbracket \Gamma; \Phi \vdash (\mu \phi. \lambda \bar{\alpha}. H) \bar{G} \rrbracket^{\text{Rel}} \rho = (\mu T_{H, \rho}) \llbracket \Gamma; \Phi \vdash G \rrbracket^{\text{Rel}} \rho \\
& \quad \text{where } T_{H, \rho} = (T_{H, \pi_1 \rho}^{\text{Set}}, T_{H, \pi_2 \rho}^{\text{Set}}, T_{H, \rho}^{\text{Rel}}) \\
& \quad \text{and } T_{H, \rho}^{\text{Rel}} F = \lambda \bar{R}. \llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Rel}} \rho[\phi := F][\bar{\alpha} := \bar{R}] \\
& \quad \text{and } T_{H, \rho}^{\text{Rel}} \delta = \lambda \bar{R}. \llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Rel}} id_{\rho}[\phi := \delta][\bar{\alpha} := id_{\bar{R}}]
\end{aligned}$$

Fig. 3. Relational interpretation

argue that they are well-defined and have the required properties. To do this, we extend T_H to a *functor* from RelEnv to $[[\text{Rel}^k, \text{Rel}], [\text{Rel}^k, \text{Rel}]]$. Its action on an object $\rho \in \text{RelEnv}$ is given by the higher-order functor $T_{H, \rho}$ whose actions on objects and morphisms are given in Figure 3. Its action on a morphism $f : \rho \rightarrow \rho'$ is the higher-order natural transformation $T_{H, f} : T_{H, \rho} \rightarrow T_{H, \rho'}$ whose action on any $F : [\text{Rel}^k, \text{Rel}]$ is the natural transformation $T_{H, f} F : T_{H, \rho} F \rightarrow T_{H, \rho'} F$ whose component at $\bar{R}$ is $(T_{H, f} F)_{\bar{R}} = \llbracket \Gamma; \phi, \bar{\alpha} \vdash H \rrbracket^{\text{Rel}} \rho[\phi := id_F][\bar{\alpha} := id_{\bar{R}}]$.

Using T_H , we can define the functorial action of relational interpretation. The action $\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}} f$ of $\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}}$ on $f : \rho \rightarrow \rho'$ in RelEnv is given as in Definition 2, except that all interpretations are relational interpretations and all occurrences of $T_{H, f}^{\text{Set}}$ are replaced by $T_{H, f}$. For this definition and Figure 3 to be well-defined we need that, for every H , $T_{H, \rho} F$ is a relation transformer, and $T_{H, f} F : T_{H, \rho} F \rightarrow T_{H, \rho'} F$ is a morphism of relation transformers, whenever F is a relation transformer and $f : \rho \rightarrow \rho'$ is in RelEnv . This is immediate from

$$\llbracket \Gamma; \Phi \vdash F \rrbracket = (\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}}, \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}}, \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}}) \in [\text{RelEnv}, RT_0] \quad (1)$$

The proof is a straightforward induction on the structure of F , using an appropriate result from [17] to deduce ω -cocontinuity of $\llbracket \Gamma; \Phi \vdash F \rrbracket$ in each case.

We can prove by simultaneous induction that set and relational interpretations of types respect demotion of functorial variables to non-functorial ones and, for $D \in \{\text{Set}, \text{Rel}\}$, $\llbracket \Gamma; \Phi \vdash G[\bar{\alpha} := \bar{K}] \rrbracket^D \rho = \llbracket \Gamma; \Phi, \bar{\alpha} \vdash G \rrbracket^D \rho[\bar{\alpha} := \llbracket \Gamma; \Phi \vdash K \rrbracket^D \rho]$, and $\llbracket \Gamma; \Phi \vdash G[\bar{\alpha} := \bar{K}] \rrbracket^D f = \llbracket \Gamma; \Phi, \bar{\alpha} \vdash G \rrbracket^D f[\bar{\alpha} := \llbracket \Gamma; \Phi \vdash K \rrbracket^D f]$, and $\llbracket \Gamma; \Phi \vdash F[\phi := H] \rrbracket^D \rho = \llbracket \Gamma; \Phi, \phi \vdash F \rrbracket^D \rho[\phi := \lambda \bar{A}. \llbracket \Gamma; \Phi, \bar{\alpha} \vdash H \rrbracket^D \rho[\bar{\alpha} := \bar{A}]]$, and, finally, $\llbracket \Gamma; \Phi \vdash F[\phi := H] \rrbracket^D f = \llbracket \Gamma; \Phi, \phi \vdash F \rrbracket^D f[\phi := \lambda \bar{A}. \llbracket \Gamma; \Phi, \bar{\alpha} \vdash H \rrbracket^D f[\bar{\alpha} := id_{\bar{A}}]]$.



4 The Identity Extension Lemma

In most treatments of parametricity, equality relations are taken as *given*, either directly as diagonal relations or perhaps via reflexive graphs. By contrast, we give a categorical definition of graph relations for natural transformations and *construct* equality relations as particular such relations. Our definitions specialize to the usual ones for morphisms between sets and equality relations on sets.

The standard definition $(x, y) \in \langle f \rangle$ iff $fx = y$ of the graph $\langle f \rangle$ of a morphism $f : A \rightarrow B$ in **Set** naturally generalizes to associate to each natural transformation between k -ary functors on **Set** a k -ary relation transformer. Indeed, if $F, G : \mathbf{Set}^k \rightarrow \mathbf{Set}$ and $\alpha : F \rightarrow G$ is a natural transformation, then the functor $\langle \alpha \rangle^* : \mathbf{Rel}^k \rightarrow \mathbf{Rel}$ is defined as follows. Given $R_1 : \mathbf{Rel}(A_1, B_1), \dots, R_k : \mathbf{Rel}(A_k, B_k)$, let $\iota_{R_i} : R_i \hookrightarrow A_i \times B_i$, for $i = 1, \dots, k$, 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 left diagram below commute, and let $h_{\overline{R}} : F\overline{R} \rightarrow F\overline{A} \times G\overline{B}$ 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 in the right diagram below. Then $\alpha^{\wedge \overline{R}} : \mathbf{Rel}(F\overline{A}, G\overline{B})$ by construction, so the action of $\langle \alpha \rangle^*$ on objects can be given by $\langle \alpha \rangle^*(A, B, R) = (F\overline{A}, G\overline{B}, \iota_{\alpha^{\wedge \overline{R}}} \alpha^{\wedge \overline{R}})$. Its action on morphisms is given by $\langle \alpha \rangle^*(\beta, \beta') = (F\overline{\beta}, G\overline{\beta'})$.

$$\begin{array}{ccc}
 F\overline{A} & \xleftarrow{F\overline{\pi_1}} F(\overline{A \times B}) & \xrightarrow{F\overline{\pi_2}} F\overline{B} & \xrightarrow{\alpha_{\overline{B}}} G\overline{B} \\
 & \searrow \pi_1 & \downarrow h_{\overline{A \times B}} & \nearrow \pi_2 \\
 & & F\overline{A} \times G\overline{B} & \\
 \end{array}
 \qquad
 \begin{array}{ccc}
 F\overline{R} & \xrightarrow{h_{\overline{R}}} & F\overline{A} \times G\overline{B} \\
 \searrow q_{\alpha^{\wedge \overline{R}}} & & \nearrow \iota_{\alpha^{\wedge \overline{R}}} \\
 & \alpha^{\wedge \overline{R}} &
 \end{array}$$

Lemma 1. *If $F, G : [\mathbf{Set}^k, \mathbf{Set}]$, and if $\alpha : F \rightarrow G$ is a natural transformation, then the graph relation transformer for α defined by $\langle \alpha \rangle = (F, G, \langle \alpha \rangle^*)$ is in RT_k .*

The action of a graph relation transformer on a graph relation can be computed explicitly: if $\alpha : F \rightarrow G$ is a morphism in $[\mathbf{Set}^k, \mathbf{Set}]$ and $f_1 : A_1 \rightarrow B_1, \dots, f_k : A_k \rightarrow B_k$, then $\langle \alpha \rangle^* \langle f \rangle = \langle Gf \circ \alpha_{\overline{A}} \rangle = \langle \alpha_{\overline{B}} \circ Ff \rangle$.

To prove the IEL we also need to know that equality relation transformers preserve equality relations. The *equality relation transformer* on $F : [\mathbf{Set}^k, \mathbf{Set}]$ is $\text{Eq}_F = \langle id_F \rangle = (F, F, \langle id_F \rangle^*)$. The above definition then gives that, for all $\overline{A} : \mathbf{Set}, \text{Eq}_F^* \overline{\text{Eq}}_A = \langle id_F \rangle^* \langle id_{\overline{A}} \rangle = \langle F id_{\overline{A}} \circ (id_F)_{\overline{A}} \rangle = \langle id_{F\overline{A}} \circ id_{F\overline{A}} \rangle = \langle id_{F\overline{A}} \rangle = \text{Eq}_{F\overline{A}}$. In addition, if $\rho, \rho' \in \mathbf{SetEnv}$ and $f : \rho \rightarrow \rho'$, then the *graph relation environment* $\langle f \rangle$ is defined pointwise by $\langle f \rangle \phi = \langle f \phi \rangle$ for every ϕ . This entails that $\pi_1 \langle f \rangle = \rho$ and $\pi_2 \langle f \rangle = \rho'$. The *equality relation environment* Eq_ρ is defined to be $\langle id_\rho \rangle$. Our IEL is thus:

Theorem 1 (IEL). *If $\rho \in \mathbf{SetEnv}$, then $\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}} \text{Eq}_\rho = \text{Eq}_{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} \rho}$.*

The IEL's highly non-trivial proof is by induction on the structure of F . Only the **Nat**, application, and fixpoint cases are non-routine. The latter two explicitly calculate actions of graph relation transformers as above. The fixpoint case also uses that, for every $n \in \mathbb{N}$, the following intermediate results can be proved by simultaneous induction with Theorem 1: for any $H, \rho, \overline{A}$, and subformula J

$$\begin{array}{ll}
\llbracket \Gamma; \Phi \mid \Delta, x : F \vdash x : F \rrbracket^{\mathbf{D}} \rho & = \pi_{|\Delta|+1} \\
\llbracket \Gamma; \emptyset \mid \Delta \vdash L_{\bar{\alpha}} x.t : \mathbf{Nat}^{\bar{\alpha}} F G \rrbracket^{\mathbf{D}} \rho & = \text{curry}(\llbracket \Gamma; \bar{\alpha} \mid \Delta, x : F \vdash t : G \rrbracket^{\mathbf{D}} \rho[\bar{\alpha} := \bar{\alpha}]) \\
\llbracket \Gamma; \Phi \mid \Delta \vdash t_{\bar{K}} s : G[\bar{\alpha} := \bar{K}] \rrbracket^{\mathbf{D}} \rho & = \text{eval} \circ \langle \lambda d. (\llbracket \Gamma; \emptyset \mid \Delta \vdash t : \mathbf{Nat}^{\bar{\alpha}} F G \rrbracket^{\mathbf{D}} \rho d) \overline{\llbracket \Gamma; \Phi \vdash \bar{K} \rrbracket^{\mathbf{D}} \rho}, \\
& \quad \llbracket \Gamma; \Phi \mid \Delta \vdash s : F[\bar{\alpha} := \bar{K}] \rrbracket^{\mathbf{D}} \rho \rangle \\
\\
\llbracket \Gamma; \Phi \mid \Delta \vdash \perp_F t : F \rrbracket^{\mathbf{D}} \rho & = !_{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\mathbf{D}} \rho}^0 \circ \llbracket \Gamma; \Phi \mid \Delta \vdash t : 0 \rrbracket^{\mathbf{D}} \rho, \text{ where } \\
& \quad !_{\llbracket \Gamma; \Phi \vdash F \rrbracket^{\mathbf{D}} \rho}^0 \text{ is the unique morphism from } 0 \\
& \quad \text{to } \llbracket \Gamma; \Phi \vdash F \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \Phi \mid \Delta \vdash \top : 1 \rrbracket^{\mathbf{D}} \rho & = !_1^{\llbracket \Gamma; \Phi \vdash 1 \rrbracket^{\mathbf{D}} \rho}, \text{ where } !_1^{\llbracket \Gamma; \Phi \vdash 1 \rrbracket^{\mathbf{D}} \rho} \\
& \quad \text{is the unique morphism from } \llbracket \Gamma; \Phi \vdash 1 \rrbracket^{\mathbf{D}} \rho \text{ to } 1 \\
\llbracket \Gamma; \Phi \mid \Delta \vdash (s, t) : F \times G \rrbracket^{\mathbf{D}} \rho & = \llbracket \Gamma; \Phi \mid \Delta \vdash s : F \rrbracket^{\mathbf{D}} \rho \times \llbracket \Gamma; \Phi \mid \Delta \vdash t : G \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \Phi \mid \Delta \vdash \pi_1 t : F \rrbracket^{\mathbf{D}} \rho & = \pi_1 \circ \llbracket \Gamma; \Phi \mid \Delta \vdash t : F \times G \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \Phi \mid \Delta \vdash \pi_2 t : G \rrbracket^{\mathbf{D}} \rho & = \pi_2 \circ \llbracket \Gamma; \Phi \mid \Delta \vdash t : F \times G \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \Phi \mid \Delta \vdash \text{case } t \text{ of } \{x \mapsto l; y \mapsto r\} : K \rrbracket^{\mathbf{D}} \rho & = \text{eval} \circ \langle \text{curry}(\llbracket \Gamma; \Phi \mid \Delta, x : F \vdash l : K \rrbracket^{\mathbf{D}} \rho, \\
& \quad \llbracket \Gamma; \Phi \mid \Delta, y : G \vdash r : K \rrbracket^{\mathbf{D}} \rho), \\
& \quad \llbracket \Gamma; \Phi \mid \Delta \vdash t : F + G \rrbracket^{\mathbf{D}} \rho \rangle \\
\llbracket \Gamma; \Phi \mid \Delta \vdash \text{inL } s : F + G \rrbracket^{\mathbf{D}} \rho & = \text{inL} \circ \llbracket \Gamma; \Phi \mid \Delta \vdash s : F \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \Phi \mid \Delta \vdash \text{inR } t : F + G \rrbracket^{\mathbf{D}} \rho & = \text{inR} \circ \llbracket \Gamma; \Phi \mid \Delta \vdash t : G \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \emptyset \mid \emptyset \vdash \text{map}_{\bar{F}, \bar{G}}^{\bar{F}, \bar{G}} : \mathbf{Nat}^{\bar{\alpha}}(\mathbf{Nat}^{\bar{\beta}, \bar{\gamma}} F G) & = \lambda d \bar{\eta} \bar{C}. \llbracket \Gamma; \bar{\alpha}, \bar{\beta}, \bar{\gamma} \vdash H \rrbracket^{\mathbf{D}} id_{\rho[\bar{\gamma} := \bar{C}]}[\bar{\phi} := \lambda \bar{B}. \bar{\eta} \bar{B} \bar{C}] \\
& \quad (\mathbf{Nat}^{\bar{\gamma}} H[\bar{\phi} := \bar{\beta} F] H[\bar{\phi} := \bar{\beta} G]) \rrbracket^{\mathbf{D}} \rho \\
\llbracket \Gamma; \emptyset \mid \emptyset \vdash \text{in}_H : \mathbf{Nat}^{\bar{\beta}} H[\bar{\phi} := (\mu \phi. \lambda \bar{\alpha}. H) \bar{\beta}] [\bar{\alpha} := \bar{\beta}] & = \lambda d. \text{in}_{T_{H, \rho}^X} \text{ where } X \text{ is Set when } \\
& \quad (\mu \phi. \lambda \bar{\alpha}. H) \bar{\beta} \rrbracket^{\mathbf{D}} \rho \quad \mathbf{D} = \mathbf{Set} \text{ and not present when } \mathbf{D} = \mathbf{Rel} \\
\llbracket \Gamma; \emptyset \mid \emptyset \vdash \text{fold}_H^{\bar{F}} : \mathbf{Nat}^{\bar{\alpha}}(\mathbf{Nat}^{\bar{\beta}} H[\bar{\phi} := \bar{\beta} F] [\bar{\alpha} := \bar{\beta}] F) & = \lambda d. \text{fold}_{T_{H, \rho}^X, \rho} \\
& \quad (\mathbf{Nat}^{\bar{\beta}} (\mu \phi. \lambda \bar{\alpha}. H) \bar{\beta} F) \rrbracket^{\mathbf{D}} \rho \quad \text{where } X \text{ is as above}
\end{array}$$

Fig. 4. Term semantics

of H , both $T_{H, \text{Eq}_\rho}^n K_0 \overline{\text{Eq}_A} = (\text{Eq}_{(T_{H, \rho}^{\text{Set}})^n K_0})^* \overline{\text{Eq}_A}$ and $\llbracket \Gamma; \Phi, \phi, \bar{\alpha} \vdash J \rrbracket^{\text{Rel}} \text{Eq}_\rho[\phi := T_{H, \text{Eq}_\rho}^n K_0][\bar{\alpha} := \overline{\text{Eq}_A}] = \llbracket \Gamma; \Phi, \phi, \bar{\alpha} \vdash J \rrbracket^{\text{Rel}} \text{Eq}_\rho[\phi := \text{Eq}_{(T_{H, \rho}^{\text{Set}})^n K_0}][\bar{\alpha} := \overline{\text{Eq}_A}]$ hold. The case of the proof when F and J are both μ -types makes clear that if functorial variables of arity greater than 0 were allowed to appear in the bodies of μ -types, then the IEL would fail.

With the IEL in hand we can prove a Graph Lemma for our setting:

Lemma 2. *If $\rho, \rho' \in \text{SetEnv}$ and $f : \rho \rightarrow \rho'$ then $\langle \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Set}} f \rangle = \llbracket \Gamma; \Phi \vdash F \rrbracket^{\text{Rel}} \langle f \rangle$.*

5 Interpreting Terms

If $\Delta = x_1 : F_1, \dots, x_n : F_n$ is a term context for Γ and Φ , define $\llbracket \Gamma; \Phi \vdash \Delta \rrbracket^{\mathbf{D}} = \llbracket \Gamma; \Phi \vdash F_1 \rrbracket^{\mathbf{D}} \times \dots \times \llbracket \Gamma; \Phi \vdash F_n \rrbracket^{\mathbf{D}}$, where $\mathbf{D}$ is **Set** or **Rel** as appropriate. Then every well-formed term has a set (resp., relational) interpretation as a natural transformation from the set (resp., relational) interpretation of its term context to that of its type. These interpretations, given in Figure 4, respect weakening, so that $\llbracket \Gamma; \Phi \mid \Delta, x : F \vdash t : G \rrbracket^{\mathbf{D}} \rho = (\llbracket \Gamma; \Phi \mid \Delta \vdash t : G \rrbracket^{\mathbf{D}} \rho) \circ \pi_\Delta$, where $\rho \in \text{SetEnv}$ or $\rho \in \text{RelEnv}$, and π_Δ is the projection $\llbracket \Gamma; \Phi \vdash \Delta, x : F \rrbracket^{\mathbf{D}} \rho \rightarrow \llbracket \Gamma; \Phi \vdash \Delta \rrbracket^{\mathbf{D}} \rho$.

The return type for the semantic fold is $\llbracket \Gamma; \bar{\beta} \vdash F \rrbracket^{\mathbf{D}} \rho[\bar{\beta} := \bar{B}]$. This interpretation gives $\llbracket \Gamma; \emptyset \mid \Delta \vdash \lambda x.t : F \rightarrow G \rrbracket^{\mathbf{D}} \rho = \text{curry}(\llbracket \Gamma; \emptyset \mid \Delta, x : F \vdash t : G \rrbracket^{\mathbf{D}} \rho)$ and $\llbracket \Gamma; \emptyset \mid \Delta \vdash st : G \rrbracket^{\mathbf{D}} \rho = \text{eval} \circ \langle \llbracket \Gamma; \emptyset \mid \Delta \vdash s : F \rightarrow G \rrbracket^{\mathbf{D}} \rho, \llbracket \Gamma; \emptyset \mid \Delta \vdash t : F \rrbracket^{\mathbf{D}} \rho \rangle$, 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 $\llbracket t : F \rrbracket^D$ instead of $\llbracket \emptyset; \emptyset \mid \emptyset \vdash t : F \rrbracket^D$. In addition, term interpretation respects substitution for both functorial and non-functorial type variables, as well as term substitution. Direct calculation reveals that interpretations of terms also satisfy $\llbracket \Gamma; \Phi \mid \Delta \vdash (L_{\bar{\alpha}x}.t)_{\bar{K}} s \rrbracket^D = \llbracket \Gamma; \Phi \mid \Delta \vdash t[\bar{\alpha} := \bar{K}][x := s] \rrbracket^D$. Term extensionality for both types and terms — i.e., $\llbracket \Gamma; \Phi \vdash (L_{\alpha x}.t)_{\alpha} \top : F \rrbracket^D = \llbracket \Gamma; \Phi \vdash t : F \rrbracket^D$ and $\llbracket \Gamma; \Phi \vdash (L_{\alpha x}.t)_{\alpha} x : F \rrbracket^D = \llbracket \Gamma; \Phi \vdash t : F \rrbracket^D$ — follow (when both sides of these equations are defined).

6 Free Theorems for Nested Types

6.1 Consequences of Naturality

Define, for $\Gamma; \bar{\alpha} \vdash F$, the term id_F to be $\Gamma; \emptyset \mid \emptyset \vdash L_{\bar{\alpha}x}.x : \text{Nat}^{\bar{\alpha}} F F$ and, for terms $\Gamma; \emptyset \mid \Delta \vdash t : \text{Nat}^{\bar{\alpha}} F G$ and $\Gamma; \emptyset \mid \Delta \vdash s : \text{Nat}^{\bar{\alpha}} G H$, the *composition* $s \circ t$ of t and s to be $\Gamma; \emptyset \mid \Delta \vdash L_{\bar{\alpha}x}.s_{\bar{\alpha}}(t_{\bar{\alpha}x}) : \text{Nat}^{\bar{\alpha}} F H$. Then $\llbracket \Gamma; \emptyset \mid \emptyset \vdash id_F : \text{Nat}^{\bar{\alpha}} F F \rrbracket^{\text{Set}} \rho_* = id_{\lambda \bar{A}. \llbracket \Gamma; \bar{\alpha} \vdash F \rrbracket^{\text{Set}} \rho[\bar{\alpha} := \bar{A}]}$ for any set environment ρ , and $\llbracket \Gamma; \emptyset \mid \Delta \vdash s \circ t : \text{Nat}^{\bar{\alpha}} F H \rrbracket^{\text{Set}} = \llbracket \Gamma; \emptyset \mid \Delta \vdash s : \text{Nat}^{\bar{\alpha}} G H \rrbracket^{\text{Set}} \circ \llbracket \Gamma; \emptyset \mid \Delta \vdash t : \text{Nat}^{\bar{\alpha}} F G \rrbracket^{\text{Set}}$. Also, terms of Nat type behave as natural transformations with respect to their source and target types:

Theorem 2. *If $\Gamma; \emptyset \mid \Delta \vdash s : \text{Nat}^{\bar{\alpha}, \bar{\gamma}} F G$ and $\Gamma; \emptyset \mid \Delta \vdash t : \text{Nat}^{\bar{\gamma}} K H$, then*

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

Theorem 2 gives rise to an entire family of free theorems that are consequences of naturality, and thus do not require the full power of parametricity. In particular, we can prove that the interpretation of every map_H is a functor, and that map is itself a higher-order functor. For example, the former property can be stated as: if $\Gamma; \bar{\alpha}, \bar{\gamma} \vdash H$, $\Gamma; \emptyset \mid \Delta \vdash g : \text{Nat}^{\bar{\gamma}} F G$, and $\Gamma; \emptyset \mid \Delta \vdash f : \text{Nat}^{\bar{\gamma}} G K$, then

$$\begin{aligned} & \llbracket \Gamma; \emptyset \mid \Delta \vdash (\text{map}_H^{\bar{F}, \bar{K}})_{\emptyset} (\bar{f} \circ \bar{g}) : \text{Nat}^{\bar{\gamma}} H[\bar{\alpha} := \bar{F}] H[\bar{\alpha} := \bar{K}] \rrbracket^{\text{Set}} \\ &= \llbracket \Gamma; \emptyset \mid \Delta \vdash (\text{map}_H^{\bar{G}, \bar{K}})_{\emptyset} \bar{f} \circ (\text{map}_H^{\bar{F}, \bar{G}})_{\emptyset} \bar{g} : \text{Nat}^{\bar{\gamma}} H[\bar{\alpha} := \bar{F}] H[\bar{\alpha} := \bar{K}] \rrbracket^{\text{Set}} \end{aligned}$$

We can also prove the expected properties of map , in , and fold , and their interpretations, e.g., uniqueness and the universal property of the interpretation of fold , and the interpretation of in is an isomorphism.

6.2 The Abstraction Theorem

To get consequences of parametricity that are not merely consequences of naturality, we prove an Abstraction Theorem (Theorem 4). As usual for such theorems, we prove a more general result (Theorem 3) for open terms, and recover our Abstraction Theorem as its special case for closed terms of closed type.

Theorem 3. *Every well-formed term $\Gamma; \Phi \mid \Delta \vdash t : F$ induces a natural transformation from $\llbracket \Gamma; \Phi \vdash \Delta \rrbracket$ to $\llbracket \Gamma; \Phi \vdash F \rrbracket$, i.e., a triple of natural transformations $(\llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\text{Set}}, \llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\text{Set}}, \llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\text{Rel}})$, where, for $D \in \{\text{Set}, \text{Rel}\}$, and for $\rho \in \text{SetEnv}$ or $\rho \in \text{RelEnv}$ as appropriate, $\llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^D : \llbracket \Gamma; \Phi \vdash \Delta \rrbracket^D \rightarrow \llbracket \Gamma; \Phi \vdash F \rrbracket^D$ has component $\llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^D \rho : \llbracket \Gamma; \Phi \vdash \Delta \rrbracket^D \rho \rightarrow \llbracket \Gamma; \Phi \vdash F \rrbracket^D \rho$ at ρ . Moreover, for all $\rho \in \text{RelEnv}$, we have $\llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\text{Rel}} \rho = (\llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\text{Set}}(\pi_1 \rho), \llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\text{Set}}(\pi_2 \rho))$.*

The proof is by induction on t . It requires showing that set and relational interpretations of term judgments are natural transformations, and that all set interpretations of terms of $\mathbf{Nat}$ -types satisfy the appropriate equality preservation conditions from Figure 2. For the interesting cases of abstraction, application, **map**, **in**, and **fold** terms, propagating the naturality conditions is somewhat involved; the latter two especially require some delicate diagram chasing. That it is possible provides strong evidence that our development is sensible, natural, and at an appropriate level of abstraction.

Using Theorem 3 we can prove that our calculus admits no terms with the type $\mathbf{Nat}^\alpha \mathbf{1} \alpha$ of the polymorphic bottom, and every closed term g of type $\mathbf{Nat}^\alpha \alpha$ denotes the polymorphic identity function. Moreover, an immediate consequence of Theorem 3 is that if $\rho \in \mathbf{RelEnv}$, and $(a, b) \in \llbracket \Gamma; \Phi \vdash \Delta \rrbracket^{\mathbf{Rel}} \rho$, then $(\llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\mathbf{Set}}(\pi_1 \rho) a, \llbracket \Gamma; \Phi \mid \Delta \vdash t : F \rrbracket^{\mathbf{Set}}(\pi_2 \rho) b) \in \llbracket \Gamma; \Phi \vdash F \rrbracket^{\mathbf{Rel}} \rho$. Its instantiation to closed terms of closed type gives

Theorem 4 (Abstraction Theorem). $(\llbracket \vdash t : F \rrbracket^{\mathbf{Set}}, \llbracket \vdash t : F \rrbracket^{\mathbf{Set}}) \in \llbracket \vdash F \rrbracket^{\mathbf{Rel}}$

Using Theorem 4 we can recover free theorems, such as that for the type of the standard *filter* function for lists, that go beyond mere naturality, and extend them to those nested types for which analogous functions can be defined. In particular, we can extend short cut fusion for lists [10] to nested types, thereby formally proving correctness of the categorically inspired theorem from [16]. As shown there, replacing $\mathbf{1}$ with any type $\emptyset; \alpha \vdash C$ generalizes Theorem 5 to a free theorem whose conclusion is $\text{fold}_H B \circ G \mu H \text{ in}_H = G \llbracket \emptyset; \alpha \vdash K \rrbracket^{\mathbf{Set}} B$.

Theorem 5. *If $\emptyset; \phi, \alpha \vdash F$, $\emptyset; \alpha \vdash K$, $H : [\mathbf{Set}, \mathbf{Set}] \rightarrow [\mathbf{Set}, \mathbf{Set}]$ is defined by $H f x = \llbracket \emptyset; \phi, \alpha \vdash F \rrbracket^{\mathbf{Set}}[\phi := f][\alpha := x]$, and $G = \llbracket \emptyset; \emptyset \mid \emptyset \vdash g : \mathbf{Nat}^\emptyset (\mathbf{Nat}^\alpha F(\phi\alpha)) (\mathbf{Nat}^\alpha \mathbf{1}(\phi\alpha)) \rrbracket^{\mathbf{Set}}$ for some g , then for every $B \in H \llbracket \emptyset; \alpha \vdash K \rrbracket^{\mathbf{Set}} \rightarrow \llbracket \emptyset; \alpha \vdash K \rrbracket^{\mathbf{Set}}$ we have $\text{fold}_H B (G \mu H \text{ in}_H) = G \llbracket \emptyset; \alpha \vdash K \rrbracket^{\mathbf{Set}} B$.*

7 Conclusion and Directions for Future Work

We have constructed a parametric model for a calculus supporting primitive nested types, and used its Abstraction Theorem to derive free theorems for these types. This was not possible before [17] because these types were not previously known to have well-defined interpretations in locally finitely presentable categories (here, **Set** and **Rel**), and, to our knowledge, no term calculus for them existed either. We naturally hope (some appropriate variant of) the construction elaborated here will generalize to more advanced data types. For example, GADTs can be represented using left Kan extensions, and it was shown in [17] that adding a **Lan** construct to a calculus such as ours preserves the λ -cocontinuity needed for the data types it defines to have well-defined interpretations in locally λ -presentable categories. (Interestingly, $\lambda > \aleph_1$ is required to interpret even common GADTs.) This suggests carrying out our model construction in locally λ -presentable cartesian closed categories (**lpcccs**) $\mathcal{C}$ whose categories of (abstract) relations, obtained by pullback as in [13], are also **lpcccs** and are appropriately fibred over $\mathcal{C}$. Adding term-level fixpoints further requires our semantic categories not just to be locally λ -presentable, but to support some kind of domain structure as well.



References

1. Adámek, J., Rosický, J.: Locally Presentable and Accessible Categories. Cambridge University Press (1994)
2. Atkey, R.: Relational Parametricity for Higher Kinds. In: Computer Science Logic, pp. 46–61. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik (2012)
3. Bainbridge, E. S., Freyd, P. J., Scedrov, A., Scott, P. J.: Functorial Polymorphism. Theoretical Computer Science 70, 35–64 (1990)
4. Bird, R., Meertens, L.: Nested datatypes. In: Mathematics of Program Construction, pp. 52–67. Springer (1998)
5. Bird, R., Paterson, R.: Generalised folds for nested datatypes. Formal Aspects of Computing 11, 200–222 (1999)
6. Birkedal, L., Møgelberg, R. E.: Categorical models for Abadi and Plotkin’s logic for parametricity. Mathematical Structures in Computer Science 15, 709–772 (2005)
7. Cardelli, L.: Type Systems. In: CRC Handbook of Computer Science and Engineering, pp. 2208–2236. CRC Press (1984)
8. Dunphy, B., Reddy, U.: Parametric Limits. In: Logic in Computer Science, pp. 242–252. IEEE (2004)
9. Ghani, N., Johann, P., Nordvall Forsberg, F., Orsanigo, F., Revell, T.: Bifibrational Functorial Semantics for Parametric Polymorphism. Electronic Notes in Theoretical Computer Science 319, 165–181. (2015)
10. Gill, A., Launchbury, J., Peyton Jones, S. L.: A short cut to deforestation. In: Functional Programming Languages and Computer Architecture, Proceedings, pp. 223–232. Association for Computing Machinery (1993)
11. Girard, J.-Y.: Interprétation fonctionnelle et élimination des coupures de l’arithmétique d’ordre supérieur. PhD thesis, University of Paris (1972)
12. Hasegawa, R.: Categorical data types in parametric polymorphism. Mathematical Structures in Computer Science 4, 71–109 (1994)
13. Jacobs, B.: Categorical Logic and Type Theory. Elsevier (1999)
14. Johann, P.: A Generalization of Short-Cut Fusion and Its Correctness Proof. Higher-Order and Symbolic Computation 15, 273–300 (2002)
15. Johann, P., Ghani, N.: Foundations for Structured Programming with GADTs. In: Principles of Programming Languages, pp. 297–308. Association for Computing Machinery (2008)
16. Johann, P., Ghani, N.: Haskell Programming with Nested Types: A Principled Approach Higher-Order and Symbolic Computation 22(2), 155–189 (2010)
17. Johann, P., Polonsky, A.: Higher-kinded data types: Syntax and Semantics. In: Logic in Computer Science, pp. 1–13. IEEE (2019)
18. Johann, P., Polonsky, A.: Deep Induction: Induction Rules for (Truly) Nested Types. In: Foundations of Software Science and Computation Structures, pp. 339–358. Springer (2020)
19. Matthes, R.: Map Fusion for Nested Datatypes in Intensional Type Theory. Science of Computer Programming 76(3), 204–224 (2011)
20. Ma, Q., Reynolds, J. C.: Types, abstraction, and parametric polymorphism, part 2. In: Mathematical Foundations of Program Semantics, pp. 1–40. Springer-Verlag (1992)
21. Martin, C., Gibbons, J.: On the semantics of nested datatypes. Information Processing Letters 80(5), 233–238 (2001)
22. Pitts, A.: Parametric polymorphism, recursive types, and operational equivalence. (1998)

23. Pitts, A.: Parametric polymorphism and operational equivalence. *Mathematical Structures in Computer Science* 10, 321–359 (2000)
24. Reynolds, J. C.: Types, abstraction, and parametric polymorphism. *Information Processing* 83(1), 513–523 (1983)
25. Reynolds, J. C.: Polymorphism is not set-theoretic. *Semantics of Data Types*, 145–156 (1984)
26. Robinson, E., Rosolini, G.: Reflexive graphs and parametric polymorphism. In: *Logic in Computer Science*, pp. 364–371. IEEE (1994)
27. Wadler, P.: Theorems for free!. In: *Functional Programming Languages and Computer Architecture, Proceedings*, pp. 347–359. Association for Computing Machinery (1989)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

