High-level signatures and initial semantics

High-level signatures and initial semantics

Benedikt Ahrens
University of Birmingham, UK
B.Ahrens@cs.bham.ac.uk

André Hirschowitz
Université Nice Sophia Antipolis, France
ah@unice.fr

Ambroise Lafont
IMT Atlantique
Inria, LS2N CNRS, France
ambroise.lafont@inria.fr

Marco Maggesi
Università degli Studi di Firenze, Italy
marco.maggesi@unifi.it

Abstract

We present a device for specifying and reasoning about syntax for datatypes, programming languages, and logic calculi. More precisely, we study a notion of 'signature' for specifying syntactic constructions.

In the spirit of Initial Semantics, we define the 'syntax generated by a signature' to be the initial object—if it exists—in a suitable category of models. In our framework, the existence of an associated syntax to a signature is not automatically guaranteed. We identify, via the notion of *presentation of a signature*, a large class of signatures that do generate a syntax.

Our (presentable) signatures subsume classical algebraic signatures (i.e., signatures for languages with variable binding, such as the pure lambda calculus) and extend them to include several other significant examples of syntactic constructions.

One key feature of our notions of signature, syntax, and presentation is that they are highly compositional, in the sense that complex examples can be obtained by assembling simpler ones. Moreover, through the Initial Semantics approach, our framework provides, beyond the desired algebra of terms, a well-behaved substitution and the induction and recursion principles associated to the syntax.

This paper builds upon ideas from a previous attempt by Hirschowitz-Maggesi, which, in turn, was directly inspired by some earlier work of Ghani-Uustalu-Hamana and Matthes-Uustalu.

The main results presented in the paper are computer-checked within the UniMath system.

2012 ACM Subject Classification Theory of computation → Algebraic language theory

Keywords and phrases initial semantics, signature, syntax, monadic substitution, computer-checked proof

Supplement Material Computer-checked proofs with compilation instructions on `https://github.com/UniMath/largecatmodules/tree/cee7580`

Funding This work has partly been funded by the CoqHoTT ERC Grant 637339 and by the EPSRC Grant EP/T000252/1. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0363.
Benedikt Ahrens: Ahrens acknowledges the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project *Homotopy Type Theory and Univalent Foundations* during the 2018/19 academic year.
Marco Maggesi: Maggesi acknowledges the support of Gruppo Nazionale per le Struttore Algebriche, Geometriche e le loro Applicazioni (GNSAGA), Istituto Nazionale di Alta Matematematica "F. Severi" (INdAM), and Italian Ministry of Education, University and Research (MIUR).

Acknowledgements We thank the anonymous referees for their helpful and constructive comments.

1 Introduction

1.1 Initial Semantics

The concept of characterising data through an initiality property is standard in computer science, where it is known under the terms *Initial Semantics* and *Algebraic Specification* [26], and has been popularised by the movement of *Algebra of Programming* [14].

This concept offers the following methodology to define a *formal language*[1]:

1. Introduce a notion of signature.
2. Construct an associated notion of model. Such models should form a category.
3. Define the *syntax generated by a signature* to be its initial model, when it exists.
4. Find a satisfactory sufficient condition for a signature to generate a syntax[2].

The models of a signature should be understood as domain of interpretation of the syntax generated by the signature: initiality of the syntax should give rise to a convenient *recursion* principle.

For a notion of signature to be satisfactory, it should satisfy the following conditions:

- it should extend the notion of algebraic signature, and
- complex signatures should be built by assembling simpler ones, thereby opening room for compositionality properties.

In the present work, we consider a general notion of signature—together with its associated notion of model—which is suited for the specification of untyped programming languages with variable binding. On the one hand, our signatures are fairly more general than those introduced in some of the seminal papers on this topic [21, 27, 22], which are essentially given by a family of lists of natural numbers indicating the number of variables bound in each subterm of a syntactic construction (we call them 'algebraic signatures' below). On the other hand, the existence of an initial model in our setting is not automatically guaranteed.

One main result of this paper is a sufficient condition on a signature to ensure such an existence. Our condition is still satisfied far beyond the algebraic signatures mentioned above. Specifically, our signatures form a cocomplete category and our condition is preserved by colimits (Section 6). Examples are given in Section 8.

Our notions of signature and syntax enjoy modularity in the sense introduced by [24]: indeed, we define a 'total' category of models where objects are pairs consisting of a signature together with one of its models; and in this total category of models, merging two extensions of a syntax corresponds to building an amalgamated sum.

The present work improves on a previous attempt [31] in two main ways: firstly, it gives a much simpler condition for the existence of an initial model; secondly, it provides computer-checked proofs for all the main statements.

1.2 Computer-checked formalization

This article is accompanied by computer-checked proofs of the main results (Theorem 39, Theorem 43, and its variant, Theorem 44). These proofs are based on the `UniMath` library

[1] Here, the word 'language' encompasses data types, programming languages and logic calculi, as well as languages for algebraic structures as considered in Universal Algebra.

[2] In the literature, the word signature is often reserved for the case where such sufficient condition is automatically ensured.

[40], which itself is based on the proof assistant Coq [39]. The main reasons for our choice of proof assistant are twofold: firstly, the logical basis of the Coq proof assistant, dependent type theory, is well suited for abstract algebra, in particular, for category theory. Secondly, a suitable library of category theory, ready for use by us, had already been developed [7, 8].

The formalization consists of about 8,000 lines of code, and can be consulted on `https://github.com/UniMath/largecatmodules`. A guide is given in the README.

For the purpose of this article, we refer to a fixed version of our library, with the short hash cee7580. This version compiles with version bcc8344 of `UniMath`.

Throughout the article, statements and human-readable proofs are accompanied by their corresponding identifiers in the formalization. These identifiers are also hyperlinks to the online documentation stored at `https://initialsemantics.github.io/doc/cee7580/index.html`.

While a computer checked proof does not constitute an absolute guarantee of correctness, it seems fair to affirm that it increases trustworthiness drastically.

1.3 Related work

The Initial Semantics approach to syntax has been introduced in the seminal paper of Goguen, Thatcher, and Wagner [26].

The idea that the notion of monad is suited for modelling substitution concerning syntax (and semantics) goes back at least to Bellegarde and Hook [13] (see also, e.g., [15, 24, 38]), while, in the present context, the notion of module over a monad appears in the work of Hirschowitz and Maggesi [30], where they give a characterization of the monad of the lambda calculus modulo $\alpha\beta\eta$-equivalences using the Initial Semantics approach.

Matthes and Uustalu [38] introduce a very general notion of signature, and subsequently, Ghani, Uustalu, and Hamana [24] consider a form of colimits (namely coends) of signatures. Their treatment rests on the technical device of *strength*, and so did our preliminary version [31] of the present work. Notably, the present version simplifies the treatment by avoiding the consideration of strengths. Any signature with strength gives rise to a signature in our sense, cf. Proposition 25. Research on signatures with strength is actively developed, see also [9] for a more recent account.

We should mention several other mathematical approaches to syntax (and semantics).

Fiore, Plotkin, and Turi [21] develop a notion of substitution monoid. Following [11], this setting can be rephrased in terms of relative monads and modules over them [3]. Accordingly, our present contribution could probably be customised for this 'relative' approach.

The work by Fiore with collaborators [21, 19, 20] and the work by Uustalu with collaborators [38, 24] share two traits: firstly, the modelling of variable binding by *nested abstract syntax*, and, secondly, the reliance on tensorial strengths in the specification of substitution. In the present work, variable binding is modelled using nested abstract syntax; however, we do without strengths.

Gabbay and Pitts [22] employ a different technique for modelling variable binding, based on nominal sets. We do not see yet how our treatment of more general syntax carries over to nominal techniques.

Yet another approach to syntax is based on Lawvere Theories. This is clearly illustrated in the paper [33], where Hyland and Power also outline the link with the language of monads and put in an historical perspective.

Finally, let us mention the classical approach based on Cartesian closed categories recently revisited and extended by T. Hirschowitz [32].

1.4 Organisation of the paper

Section 2 gives a succinct account of the notion of module over a monad, which is the crucial tool underlying our definition of signatures. Our categories of signatures and models are described in Sections 3 and 4 respectively. In Section 5, we give our definition of a syntax, and we present our first main result, a modularity result about merging extensions of syntax. Our notion of *presentation of a signature* appears in Section 6. There, we also state our second main result: presentable signatures generate a syntax. The proof of that result is given in Section 7. In Section 8, we give examples of presentable signatures. In Section 9, we show through examples how recursion can be recovered from initiality.

1.5 Publication history

This is a revision of the conference paper [4] presented at Computer Science Logic (CSL) 2018. Besides several minor changes to improve overall readability, the following content has been added:

- A comparison between signatures with strength and our signatures (Proposition 25);
- An analogue of Lambek's Lemma (Lemma 34), as well as an example of a signature that is not representable (Non-example 35);
- A variant of one of the main results (Theorem 44);
- A fix in Example 9.4 counting the redexes of a lambda-term;
- A more uniform treatment of several examples—both new and previously presented—in Section 8.1;
- Explicit statements about the use of the axiom of choice;
- Hyperlinks to an online documentation of the source code of our formalisation.

2 Categories of modules over monads

This work employs category theory extensively. The essential notions required are those of category, functor, natural transformation, limit, adjunction, and monad. Our primary reference on the subject is the classical handbook of Mac Lane [37].

The main mathematical notion underlying our signatures is that of module over a monad. In this section, we recall the definition and some basic facts about modules over a monad in the specific case of the category Set of sets, although most of this material is generalizable. For a more extensive introduction to this topic we refer to [30]. Our running example is the untyped lambda calculus.

2.1 Modules over monads

A *monad* (over Set) is a monoid in the category $\mathsf{Set} \longrightarrow \mathsf{Set}$ of endofunctors of Set, i.e., a triple $R = (R, \mu, \eta)$ given by a functor $R\colon \mathsf{Set} \longrightarrow \mathsf{Set}$, and two natural transformations $\mu\colon R \cdot R \longrightarrow R$ and $\eta\colon I \longrightarrow R$ such that the following diagrams commute:

$$\begin{tikzcd} R\cdot R\cdot R \arrow[r, "R\mu"] \arrow[d, "\mu R"'] & R\cdot R \arrow[d, "\mu"] \\ R\cdot R \arrow[r, "\mu"'] & M \end{tikzcd} \qquad \begin{tikzcd} R\cdot I \arrow[r, "R\eta"] \arrow[dr, "id"'] & R\cdot R \arrow[d, "\mu"] & I\cdot R \arrow[l, "\eta R"'] \arrow[dl, "id"] \\ & R & \end{tikzcd}$$

▶ **Example 1.** The functor $\mathsf{Maybe} : \mathsf{Set} \to \mathsf{Set}$ mapping a set X to $X + 1$ is given a monad structure as follows. The unit $\eta : \mathsf{Id} \to \mathsf{Maybe}$ is given, on a set X, by the left inclusion $\mathsf{inl} : X \to X + 1$. The multiplication $\mu : \mathsf{Maybe} \cdot \mathsf{Maybe} \to \mathsf{Maybe}$ is given, on a set X, by the map $(X + 1) + 1 \to X + 1$ collapsing the two distinguished elements.

▶ **Example 2.** Our main example of monad is the monad of terms of the lambda calculus [13, 12]. Its underlying functor $\mathsf{LC} : \mathsf{Set} \longrightarrow \mathsf{Set}$ is generated by three constructions,

- $\mathsf{var} : X \to \mathsf{LC}(X)$;
- $\mathsf{app} : \mathsf{LC}(X) \times \mathsf{LC}(X) \to \mathsf{LC}(X)$; and
- $\mathsf{abs} : \mathsf{LC}(X + 1) \to \mathsf{LC}(X)$;

for any set X. Here, the distinguished variable $\mathsf{inr}(*) \in X + 1$ is the fresh variable that is bound by the construction abs. The unit $\eta : \mathsf{Id} \to \mathsf{LC}$ is given by var, and the multiplication $\mu : \mathsf{LC} \cdot \mathsf{LC} \to \mathsf{LC}$ is given by "flattening", which intuitively removes a layer of var's from within the terms in $\mathsf{LC}(\mathsf{LC}(X))$.

Given two monads $R = (R, \eta, \mu)$ and $R' = (R', \eta', \mu')$, a *morphism* $f : R \longrightarrow R'$ *of monads* is given by a natural transformation $f : R \longrightarrow R'$ between the underlying functors such that the following diagrams commute:

$$\begin{tikzcd}
\mathsf{Id} \arrow[r, "\eta"] \arrow[dr, "\eta"'] & R \arrow[d, "f"] \\
& R'
\end{tikzcd}
\qquad
\begin{tikzcd}
R \cdot R \arrow[r, "f\,f"] \arrow[d, "\mu"'] & R' \cdot R' \arrow[d, "\mu'"] \\
R \arrow[r, "f"'] & R'
\end{tikzcd}$$

Now we want to explain in which sense syntactic constructions such as app and abs of LC commute with substitution. Indeed, it is *not* the notion of morphism of monads that captures this commutativity:

▶ **Non-example 3.** The natural transformation $\mathsf{abs} : \mathsf{LC} \cdot \mathsf{Maybe} \to \mathsf{LC}$ is *not* a morphism of monads from the composition of monads $\mathsf{LC} \cdot \mathsf{Maybe}$ to LC, see, e.g., [10, Example 3.18].

Instead, we will explain how the natural transformation abs above is a morphism of *modules over the monad* LC [29].

Let R be a monad.

▶ **Definition 4** (Modules). *A* (left) R-module *is given by a functor* $M : \mathsf{Set} \longrightarrow \mathsf{Set}$ *equipped with a natural transformation* $\rho^M : M \cdot R \longrightarrow M$*, called* module substitution*, which is compatible with the monad composition and identity, in the sense that the following two diagrams commute:*

$$\begin{tikzcd}
M \cdot R \cdot R \arrow[r, "\rho^M R"] \arrow[d, "M\rho^M"'] & M \cdot R \arrow[d, "\rho_M"] \\
M \cdot R \arrow[r, "\rho_M"'] & M
\end{tikzcd}
\qquad
\begin{tikzcd}
M \cdot I \arrow[r, "M\eta"] \arrow[dr, "id"'] & M \cdot R \arrow[d, "\rho^M"] \\
& M
\end{tikzcd}$$

There is an obvious corresponding definition of right R-modules that we do not need to consider in this paper. From now on, we will write 'R-module' instead of 'left R-module' for brevity.

▶ **Example 5.**
- Every monad R is a module over itself, which we call the *tautological* module.
- For any functor $F\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ and any R-module $M\colon \mathsf{Set} \longrightarrow \mathsf{Set}$, the composition $F \cdot M$ is an R-module (in the obvious way).
- For every set W we denote by $\underline{W}\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ the constant functor $\underline{W} := X \mapsto W$. Then $\underline{W}$ is trivially an R-module since $\underline{W} = \underline{W} \cdot R$.
- Let M_1, M_2 be two R-modules. Then the product functor $M_1 \times M_2$ is an R-module (see Proposition 8 for a general statement).
- Let R be a monad. Then $R \cdot \mathsf{Maybe}$ is an R-module in a natural way (see Section 2.3 for a general statement).

▶ **Definition 6** (Linearity). *We say that a natural transformation of R-modules $\tau\colon M \longrightarrow N$ is* linear[3] *if it is compatible with module substitution in M and N:*

$$\begin{tikzcd} M \cdot R \arrow[r, "\tau R"] \arrow[d, "\rho^M"'] & N \cdot R \arrow[d, "\rho^N"] \\ M \arrow[r, "\tau"'] & N \end{tikzcd}$$

We take linear natural transformations as morphisms among modules. It can be easily verified that we obtain in this way a category that we denote $\mathrm{Mod}(R)$*.*

▶ **Example 7** ([29, Section 5.1]). The natural transformations $\mathsf{app} : \mathsf{LC} \times \mathsf{LC} \to \mathsf{LC}$ and $\mathsf{abs} : \mathsf{LC} \cdot \mathsf{Maybe} \to \mathsf{LC}$ are morphisms of modules over the monad LC.

Beyond binary products, the category $\mathrm{Mod}(R)$ of modules over some monad R has general limits and colimits. These are constructed pointwise:

▶ **Proposition 8** (`LModule_Colims_of_shape`, `LModule_Lims_of_shape`). $\mathrm{Mod}(R)$ *is complete and cocomplete.*

These limits and colimits will in turn lift to signatures in Section 3, see Proposition 23.

2.2 The total category of modules

We already introduced the category $\mathrm{Mod}(R)$ of modules with fixed base R. Here we consider a larger category which collects modules with different bases. To this end, we need first to introduce the notion of pullback.

▶ **Definition 9** (Pullback). *Let $f\colon R \longrightarrow S$ be a morphism of monads and M an S-module. The composition $M \cdot R \xrightarrow{Mf} M \cdot S \xrightarrow{\rho^M} M$ turns the endofunctor M into an R-module which is called* pullback *of M along f and denoted by f^*M.*[4]

---

[3] Given a monoidal category $\mathcal{C}$, there is a notion of (left or right) module over a monoid object in $\mathcal{C}$ (see, e.g., [16, Section 4.1] for details). The term 'module' comes from the case of rings: indeed, a ring is just a monoid in the monoidal category of Abelian groups. Similarly, our monads are just the monoids in the monoidal category of endofunctors on Set, and our modules are just modules over these monoids. Accordingly, the term 'linear(ity)' for morphisms among modules comes from the paradigmatic case of rings.

[4] The term 'pullback' is standard in the terminology of Grothendieck fibrations (see Proposition 11).

▶ **Definition 10** (Total module category). *We define the* total module category $\int_R \mathrm{Mod}(R)$, *or* $\int \mathrm{Mod}$ *for short, as follows*[5]*:*

- *its objects are pairs* (R, M) *of a monad* R *and an* R*-module* M*.*
- *a morphism from* (R, M) *to* (S, N) *is a pair* (f, m) *where* $f\colon R \longrightarrow S$ *is a morphism of monads, and* $m\colon M \longrightarrow f^*N$ *is a morphism of* R*-modules.*

The category $\int \mathrm{Mod}$ *comes equipped with a forgetful functor to the category of monads, given by the projection* $(R, M) \mapsto R$*.*

▶ **Proposition 11** (`cleaving_bmod`)**.** *The forgetful functor* $\int \mathrm{Mod} \to \mathrm{Mon}$ *is a Grothendieck fibration with fibre* $\mathrm{Mod}(R)$ *over a monad* R*. In particular, any monad morphism* $f : R \longrightarrow S$ *gives rise to a functor*

$$f^*\colon\ \mathrm{Mod}(S) \longrightarrow \mathrm{Mod}(R)$$

given on objects by Definition 9.

▶ **Proposition 12** (`pb_LModule_colim_iso`, `pb_LModule_lim_iso`)**.** *For any monad morphism* $f : R \longrightarrow S$*, the functor* $f^* : \mathrm{Mod}(S) \longrightarrow \mathrm{Mod}(R)$ *preserves limits and colimits.*

2.3 Derivation

For our purposes, important examples of modules are given by the following general construction.

▶ **Definition 13** (Derivation). *For any* R*-module* M*, the* derivative *of* M *is the functor* $M' := M{\cdot}\mathsf{Maybe} : X \mapsto M(X{+}1)$*. It is an* R*-module with the substitution* $\rho^{M'} : M'{\cdot}R \longrightarrow M'$ *defined by the commutative diagram*

$$\begin{tikzcd}
M(R(X)+*) \arrow[r, "\rho^{M'}_X"] \arrow[d, "M(R(i_X)+\eta_{X+*}\circ\underline{*})"'] & M(X+*) \\
M(R(X+*)) \arrow[ur, "\rho^M_{X+*}"'] &
\end{tikzcd} \tag{1}$$

where $i_X\colon X \longrightarrow X + *$ *and* $\underline{*}\colon * \longrightarrow X + *$ *are the obvious maps.*

It is easy to check that derivation yields an endofunctor on the category $\mathrm{Mod}(R)$ of modules over a fixed monad R. In particular, derivation can be iterated: we denote by $M^{(k)}$ the k-th derivative of M. Moreover, notice that derivation preserves the product of modules (`commutes_binproduct_derivation`).

▶ **Definition 14.** *Given a list of nonnegative integers* $\ell = [a_1, \ldots, a_n]$ *and a module* M *over a monad* R*, we denote by* $M^\ell = M^{[a_1,\ldots,a_n]}$ *the module* $M^{(a_1)} \times \cdots \times M^{(a_n)}$*. Observe that, when* $\ell = []$ *is the empty list,* $M^{[]}$ *is the final module* $*$*.*

---

[5] Our notation for the total category is modelled after the category of elements of a presheaf, and, more generally, after the Grothendieck construction of a fibration. It overlaps with the notation for categorical ends.

▶ **Definition 15.** *For every monad R and R-module M we define the* (unary) substitution morphism $\sigma\colon M' \times R \longrightarrow M$ *by* $\sigma_X = \rho^M_X \circ w_X$*, where* $w_X : M(X + *) \times R(X) \to M(R(X))$ *is the map*

$$w_X\colon (a, b) \mapsto M(\eta_X + \underline{b})(a), \qquad \underline{b}\colon * \mapsto b.$$

▶ **Lemma 16** (`substitution_laws`). *The transformation σ is linear.*

The substitution σ allows us to interpret the derivative M' as the 'module M with one formal parameter added'.

Abstracting over the module turns the substitution morphism into a natural transformation that is the unit of the following adjunction:

▶ **Proposition 17** (`deriv_adj`). *The endofunctor of* $\mathrm{Mod}(R)$ *mapping M to the R-module $M \times R$ is left adjoint to the derivation endofunctor, the unit being the substitution morphism σ.*

3 The category of signatures

In this section, we give our notion of signature. The destiny of a signature is to have actions in monads. An action of a signature Σ in a monad R should be a morphism from a module $\Sigma(R)$ to the tautological one R. For instance, in the case of the signature Σ of a binary operation, we have $\Sigma(R) := R^2 = R \times R$. Hence a signature assigns, to each monad R, a module over R in a functorial way.

▶ **Definition 18** (`signature`). *A* signature *is a section of the forgetful functor from the category* $\int \mathrm{Mod}$ *to the category* Mon.

Now we give our basic examples of signatures.

▶ **Example 19.**
1. The assignment $R \mapsto R$ yields a signature, which we denote by Θ.
2. For any functor $F\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ and any signature Σ, the assignment $R \mapsto F \cdot \Sigma(R)$ yields a signature which we denote $F \cdot \Sigma$.
3. The assignment $R \mapsto *_R$, where $*_R$ denotes the final module over R, yields a signature which we denote by $*$.
4. Given two signatures Σ and Υ, the assignment $R \mapsto \Sigma(R) \times \Upsilon(R)$ yields a signature which we denote by $\Sigma \times \Upsilon$. For instance, $\Theta^2 = \Theta \times \Theta$ is the signature of any (first-order) binary operation, and, more generally, Θ^n is the signature of n-ary operations.
5. Given two signatures Σ and Υ, the assignment $R \mapsto \Sigma(R) + \Upsilon(R)$ yields a signature which we denote by $\Sigma + \Upsilon$. For instance, $\Theta^2 + \Theta^2$ is the signature of a pair of binary operations.

This last example explains why we do not need to distinguish here between 'arities'—usually used to specify a single syntactic construction—and 'signatures'—usually used to specify a family of syntactic constructions; our signatures allow us to do both (via Proposition 23 for families that are not necessarily finitely indexed).

Elementary signatures are of a particularly simple shape:

▶ **Definition 20.** *For each list of nonnegative integers* $\ell = [s_1, \ldots, s_n]$*, the assignment* $R \mapsto R^{(s_1)} \times \cdots \times R^{(s_n)}$ *(see Definition 14 and Example 19.1) is a signature, which we denote by* Θ^ℓ*, or by* Θ' *in the specific case of* $s = [1]$*. Signatures of this form are said* elementary.

▶ **Remark 21.** The product of two elementary signatures is elementary.

▶ **Definition 22** (signature_category)**.** *A* morphism between two signatures $\Sigma_1, \Sigma_2 \colon \mathrm{Mon} \longrightarrow \int \mathrm{Mod}$ *is a natural transformation* $m \colon \Sigma_1 \longrightarrow \Sigma_2$ *which, post-composed with the projection* $\int \mathrm{Mod} \longrightarrow \mathrm{Mon}$*, becomes the identity. Signatures form a subcategory* Sig *of the category of functors from* Mon *to* $\int \mathrm{Mod}$.

Limits and colimits of signatures can be easily constructed pointwise:

▶ **Proposition 23** (Sig_Lims_of_shape, Sig_Colims_of_shape, Sig_isDistributive)**.** *The category of signatures is complete and cocomplete. Furthermore, it is distributive: for any signature* Σ *and family of signatures* $(S_o)_{o\in O}$*, the canonical morphism* $\coprod_{o\in O}(S_o \times \Sigma) \to (\coprod_{o\in O} S_o) \times \Sigma$ *is an isomorphism.*

▶ **Definition 24.** *An* algebraic signature *is a (possibly infinite) coproduct of elementary signatures.*

These signatures are those which appear in [21]. For instance, the algebraic signature of the lambda-calculus is $\Sigma_{\mathsf{LC}} = \Theta^2 + \Theta'$.

To conclude this section, we explain the connection between *signatures with strength* (on the category Set) and our signatures.

Signatures with strength were introduced in [38] (even though they were not given an explicit name there). The relevant definitions regarding signatures with strength are summarized in [9], to which we refer the interested reader.

We recall that a signature with strength [9, Definition 4] is a pair of an endofunctor $H : [\mathcal{C}, \mathcal{C}] \to [\mathcal{C}, \mathcal{C}]$ together with a strength-like datum. Here, we only consider signatures with strength over the base category $\mathcal{C} := \mathsf{Set}$. Given a signature with strength H, we also refer to the underlying endofunctor on the functor category $[\mathsf{Set}, \mathsf{Set}]$ as $H : [\mathsf{Set}, \mathsf{Set}] \to [\mathsf{Set}, \mathsf{Set}]$.

A morphism of signatures with strength [9, Definition 5] is a natural transformation between the underlying functors that is compatible with the strengths in a suitable sense. Together with the obvious composition and identity, these objects and morphisms form a category $\mathsf{SigStrength}$ [9].

Any signature with strength H gives rise to a signature $\tilde{H}$ [31, Section 7]. This signature associates, to a monad R, an R-module whose underlying functor is $H(UR)$, where UR is the functor underlying the monad R. Similarly, given two signatures with strength H_1 and H_2, and a morphism $\alpha : H_1 \to H_2$ of signatures with strength, we associate to it a morphism of signatures $\tilde{\alpha} : \tilde{H_1} \to \tilde{H_2}$. This morphism sends a monad R to a module morphism $\tilde{\alpha}(R) : \tilde{H_1}(R) \longrightarrow \tilde{H_2}(R)$ whose underlying natural transformation is given by $\alpha(UR)$, where, as before, UR is the functor underlying the monad R. These maps assemble into a functor:

▶ **Proposition 25** (sigWithStrength_to_sig_functor)**.** *The maps sketched above yield a functor* $\tilde{(-)} : \mathsf{SigStrength} \longrightarrow \mathsf{Sig}$.

4 Categories of models

We define the notions of *model of a signature* and *action of a signature in a monad.*

▶ **Definition 26** (Actions and models)**.** *Let* Σ *be a signature. Given a monad* R*, an* action of Σ in R *is an* R*-module morphism* $r : \Sigma(R) \to R$*. A* model of Σ *is a pair* (R, r) *of a monad* R *equipped with an action of* Σ *in* R.[6] *A* morphism of models of Σ from (R, r) to (S, s) *is a*

[6] This terminology is borrowed from the vocabulary of algebras for a functor: an algebra for an endofunctor F on a category $\mathcal{C}$ is an object X of $\mathcal{C}$ with a morphism $\nu : F(X) \longrightarrow X$. This morphism is sometimes called an action.

morphism of monads $m : R \to S$ compatible with the actions, in the sense that the following diagram of R-modules commutes:

$$\begin{tikzcd}
\Sigma(R) \arrow[r, "r"] \arrow[d, "\Sigma(m)"'] & R \arrow[d, "m"] \\
m^*(\Sigma(S)) \arrow[r, "m^* s"'] & m^* S
\end{tikzcd}$$

Here, the horizontal arrows come from the actions, the left vertical arrow comes from the functoriality of signatures, and $m\colon R \longrightarrow m^*S$ is the morphism of monads seen as morphism of R-modules.

▶ **Example 27.** The usual $\mathsf{app}\colon \mathsf{LC}^2 \longrightarrow \mathsf{LC}$ is an action of the elementary signature Θ^2 in the monad LC of syntactic lambda calculus. The usual $\mathsf{abs}\colon \mathsf{LC}' \longrightarrow \mathsf{LC}$ is an action of the elementary signature Θ' in the monad LC. Then $[\mathsf{app}, \mathsf{abs}] : \mathsf{LC}^2 + \mathsf{LC}' \longrightarrow \mathsf{LC}$ is an action of the algebraic signature of the lambda calculus $\Theta^2 + \Theta'$ in the monad LC.

▶ **Proposition 28.** *Let Σ be a signature. Models of Σ, and their morphisms, together with the obvious composition and identity, form a category.*

▶ **Definition 29.** *We denote the category of Proposition 28 by Mon^Σ. It comes equipped with a forgetful functor to the category of monads.*

In the formalisation, this category is recovered as the fiber category over Σ of the displayed category [8] of models, see `rep_disp`. We have also formalized a direct definition (`rep_fiber_category`) and shown that the two definitions yield isomorphic categories: `catiso_modelcat`.

The following notion will be useful in the next section in Lemma 38.

▶ **Definition 30** (Pullback)**.** *Let $f\colon \Upsilon \longrightarrow \Sigma$ be a morphism of signatures and (R, r) a model of Σ. The linear morphism $\Upsilon(R) \xrightarrow{f(R)} \Sigma(R) \xrightarrow{r} R$ defines an action of Υ in R. The induced model of Υ is called* pullback of (R, r) along f *and denoted by $f^*(R, r)$.*

5 Syntax

We are primarily interested in the existence of an initial object in the category Mon^Σ of models of a signature Σ. We call such an essentially unique object *the syntax generated by* Σ.

5.1 Representations of a signature

▶ **Definition 31.** *If Mon^Σ has an initial object, this object is essentially unique; we say that it is a* representation of Σ *and call it the* syntax generated by Σ*, denoted by $\hat{\Sigma}$. By abuse of notation, we also denote by $\hat{\Sigma}$ the monad underlying the model $\hat{\Sigma}$.*

If an initial model for Σ exists, we say that Σ is representable[7]*.*

In this work, we aim to identify signatures that are representable. This is not automatic: below, in Non-example 35, we give a signature that is not representable. Afterwards, we give suitable sufficient criteria for signatures to be representable.

[7] For an algebraic signature Σ without binding constructions, the map assigning to any monad R its set of Σ-actions can be upgraded into a functor which is corepresented by the initial model.

We deduce the counter-example as a simple consequence of a stronger result that we consider interesting in itself: an analogue of Lambek's Lemma [36], given in Lemma 34.

The following preparatory lemma explains how to construct a new model of a signature Σ from a given one:

▶ **Lemma 32.** *Let (R, r) be a model of a signature Σ. Let $\eta : \mathsf{Id} \to R$ be the unit of the monad R, and let $\rho^{\Sigma(R)} : \Sigma(R) \cdot R \to \Sigma(R)$ be the module substitution of the R-module $\Sigma(R)$.*

- *The injection $\mathsf{Id} \to \Sigma(R) + \mathsf{Id}$ together with the natural transformation*

$$\begin{array}{c} (\Sigma(R) + \mathsf{Id}) \cdot (\Sigma(R) + \mathsf{Id}) \simeq \Sigma(R) \cdot (\Sigma(R) + \mathsf{Id}) + \Sigma(R) + \mathsf{Id} \\ \Big\downarrow {\scriptstyle \Sigma(R)[r,\eta] + \_ + \_} \\ \Sigma(R) \cdot R + \Sigma(R) + \mathsf{Id} \\ \Big\downarrow {\scriptstyle [\rho^{\Sigma(R)}, id] + \_} \\ \Sigma(R) + \mathsf{Id} \end{array}$$

  give the endofunctor $\Sigma(R) + \mathsf{Id}$ the structure of a monad.
- *Moreover, this monad can be given the following Σ-action:*

$$\Sigma\big(\Sigma(R) + \mathsf{Id}\big) \xrightarrow{\Sigma([r,\eta])} \Sigma(R) \cdot R \xrightarrow{\rho^{\Sigma(R)}} \Sigma(R) \longrightarrow \Sigma(R) + \mathsf{Id} \tag{2}$$

- *The natural transformation $[r, \eta] : \Sigma(R) + \mathsf{Id} \to R$ is a model morphism, that is, it commutes suitably with the Σ-actions of Diagram (2) in the source and $r : \Sigma(R) \longrightarrow R$ in the target.*

In the computer-checked library, the construction of the model and of the model morphism are given in `mod_id_model` and `mod_id_model_mor`, respectively.

▶ **Notation 33.** Given a model M of Σ, we denote by $M^\sharp$ the Σ-model constructed in Lemma 32, and by $\epsilon_M : M^\sharp \longrightarrow M$ the morphism of models defined there.

▶ **Lemma 34** (`iso_mod_id_model`)**.** *If Σ is representable, then the morphism of Σ-models*

$$\epsilon_{\hat{\Sigma}} : \hat{\Sigma}^\sharp \longrightarrow \hat{\Sigma}$$

is an isomorphism.

Now, we are able to give a non-representable signature.

▶ **Non-example 35.** Let $\mathcal{P}$ denote the powerset functor and consider the signature $\mathcal{P} \cdot \Theta$ (see Example 19, Item 2): it associates, to any monad R, the module $\mathcal{P} \cdot R$ that sends a set X to the powerset $\mathcal{P}(RX)$ of RX. This signature is not representable, otherwise from Lemma 34 we would have $\mathcal{P}\hat{\Sigma}X + X \cong \hat{\Sigma}X$. In particular, we would have an injective map from $\mathcal{P}\hat{\Sigma}X$ to $\hat{\Sigma}X$—contradiction.

On the other hand, as a starting point, we can identify the following class of representable signatures:

▶ **Theorem 36** (`algebraic_sig_representable`)**.** *Algebraic signatures are representable.*

This result is proved in a previous work [29, Theorems 1 and 2]. The construction of the syntax proceeds as follows: an algebraic signature induces an endofunctor on the category of endofunctors on Set. Its initial algebra (constructed as the colimit of the initial chain) is given the structure of a monad with an action of the algebraic signature, and then a routine verification shows that it is actually initial in the category of models. The computer-checked proof uses the construction of a monad from an algebraic signature formalized in [9].

In Section 6, we show a more general representability result: Theorem 43 states that *presentable* signatures, which form a superclass of algebraic signatures, are representable.

5.2 Modularity

In this section, we study the problem of how to merge two syntax extensions. Our answer, a 'modularity' result (Theorem 39), was stated already in the preliminary version [31, Section 6], there without proof.

Suppose that we have a pushout square of representable signatures,

$$\begin{tikzcd} \Sigma_0 \arrow[r] \arrow[d] & \Sigma_1 \arrow[d] \\ \Sigma_2 \arrow[r] \arrow[ur, phantom, "\ulcorner", very near end] & \Sigma \end{tikzcd}$$

Intuitively, the signatures Σ_1 and Σ_2 specify two extensions of the signature Σ_0, and Σ is the smallest extension containing both these extensions. Modularity means that the corresponding diagram of representations,

$$\begin{tikzcd} \hat{\Sigma}_0 \arrow[r] \arrow[d] & \hat{\Sigma}_1 \arrow[d] \\ \hat{\Sigma}_2 \arrow[r] & \hat{\Sigma} \end{tikzcd}$$

is a pushout as well—but we have to take care to state this in the 'right' category. The right category for this purpose is the following:

▶ **Definition 37** (Total category of models). *We denote by $\int_\Sigma \mathrm{Mon}^\Sigma$, or $\int \mathrm{Mon}$ for short, the* total category of models*:*

- *An object of $\int \mathrm{Mon}$ is a triple (Σ, R, r) where Σ is a signature, R is a monad, and r is an action of Σ in R.*
- *A morphism in $\int \mathrm{Mon}$ from (Σ_1, R_1, r_1) to (Σ_2, R_2, r_2) consists of a pair (i, m) of a signature morphism $i : \Sigma_1 \longrightarrow \Sigma_2$ and a morphism m of Σ_1-models from (R_1, r_1) to $(R_2, i^*(r_2))$.*
- *It is easily checked that the obvious composition turns $\int \mathrm{Mon}$ into a category.*

▶ **Lemma 38** (`rep_cleaving`). *The projection $\pi : \int \mathrm{Mon} \to \mathsf{Sig}$ is a Grothendieck fibration. In particular, given a morphism $f\colon \Upsilon \longrightarrow \Sigma$ of signatures, the pullback map defined in Definition 30 extends to a functor*

$$f^* : \mathrm{Mon}^\Sigma \longrightarrow \mathrm{Mon}^\Upsilon \quad .$$

Now for each signature Σ, we have an obvious inclusion from the fiber Mon^Σ into $\int \mathrm{Mon}$, through which we may see the syntax $\hat{\Sigma}$ of any representable signature as an object in $\int \mathrm{Mon}$. Furthermore, a morphism $i\colon \Sigma_1 \longrightarrow \Sigma_2$ of representable signatures yields a morphism $i_* := \hat{\Sigma}_1 \longrightarrow \hat{\Sigma}_2$ in $\int \mathrm{Mon}$. Hence our pushout square of representable signatures as described above yields a square in $\int \mathrm{Mon}$.

▶ **Theorem 39** (pushout_in_big_rep). *Modularity holds in $\int \mathrm{Mon}$, in the sense that given a pushout square of representable signatures as above, the associated square in $\int \mathrm{Mon}$ is a pushout again.*

▶ **Remark 40.** Note that Theorem 39 does *not* say that a pushout of representable signatures is representable again; it only tells us that if all of the signatures in a pushout square are representable, then the syntax generated by the pushout is the pushout of the syntaxes. In general, we do not know whether a colimit (or even a binary coproduct) of representable signatures is representable again.

6 Presentations of signatures and syntaxes

In this section, we identify a superclass of algebraic signatures that are still representable: we call them *presentable* signatures.

▶ **Definition 41.** *Given a signature Σ, a* presentation[8] of Σ *is given by an algebraic signature Υ and an epimorphism of signatures $p : \Upsilon \longrightarrow \Sigma$. In that case, we say that Σ* is presented by $p : \Upsilon \longrightarrow \Sigma$*. A signature for which a presentation exists is called* presentable*.*

Of course, any algebraic signature is presentable.

Unlike representations, presentations for a signature are not essentially unique; indeed, signatures can have many different presentations.

▶ **Remark 42.** By definition, any construction which can be encoded through a presentable signature Σ can alternatively be encoded through any algebraic signature 'presenting' Σ. The former encoding is finer than the latter in the sense that terms which are different in the latter encoding can be identified by the former. In other words, a certain amount of semantics is integrated into the syntax.

The main desired property of our presentable signatures is that, thanks to the following theorem, they are representable:

▶ **Theorem 43** (PresentableisRepresentable). *Any presentable signature is representable.*

The proof is discussed in Section 7.

Using the axiom of choice, we can prove a stronger statement:

▶ **Theorem 44** (is_right_adjoint_functor_of_reps_from_pw_epi_choice). *We assume the axiom of choice. Let Σ be a signature, and let $p : \Upsilon \longrightarrow \Sigma$ be a presentation of Σ. Then the functor $p^* : \mathrm{Mon}^\Sigma \longrightarrow \mathrm{Mon}^\Upsilon$ has a left adjoint.*

In the proof of Theorem 44, the axiom of choice is used to show that endofunctors on Set preserve epimorphisms.

Theorem 43 follows from Theorem 44 since the left adjoint $p^! : \mathrm{Mon}^\Upsilon \longrightarrow \mathrm{Mon}^\Sigma$ preserves colimits, in particular, initial objects. However, our (formalized) proof of Theorem 43 discussed in Section 7 does not invoke the axiom of choice: there, only some specific endofunctor on Set is considered, for which preservation of epimorphisms can be proved without using the axiom of choice.

For the examples of Section 8, we will use the following constructions of presentable signatures:

[8] In algebra, a presentation of a group G is an epimorphism $F \to G$ where F is free (together with a generating set of relations among the generators).

▶ **Proposition 45** (`har_binprodR_isPresentable`). *Given a presentable signature Σ, the product signature $\Sigma \times \Theta$ of Σ and the tautological signature is again presentable.*

More generally, if Σ_1 and Σ_2 are presented by $\coprod_i \Upsilon_i$ and $\coprod_j \Phi_j$ respectively, then $\Sigma_1 \times \Sigma_2$ is presented by $\coprod_{i,j} \Upsilon_i \times \Phi_j$.

▶ **Proposition 46.** *Any colimit of presentable signatures is presentable.*

▶ **Corollary 47.** *Any colimit of algebraic signatures is representable.*

7 Proof of Theorem 43

In this section, we prove Theorem 43. This proof is mechanically checked in our library; the reader may thus prefer to look at the formalised statements in the library.

Note that the proof of Theorem 43 rests on the more technical Lemma 52 below.

We will need the following characterization of epimorphisms of signatures.

▶ **Proposition 48** (`epiSig_equiv_pwEpi_SET`). *Epimorphisms of signatures are exactly pointwise epimorphisms.*

Proof. In any category, a morphism $f : a \to b$ is an epimorphism if and only if the following diagram is a pushout diagram ([37, Exercise III.4.4]) :

$$\begin{tikzcd} a \arrow[r, "f"] \arrow[d, "f"'] & b \arrow[d, "id"] \\ b \arrow[r, "id"'] & b \end{tikzcd}$$

Using this characterization of epimorphisms, the proof follows from the fact that colimits are computed pointwise in the category of signatures. ◀

Another important ingredient will be the following quotient construction for monads. Let R be a monad preserving epimorphisms, and let $\sim$ be a 'compatible' family of relations on (the functor underlying) R, that is, for any $X : \mathsf{Set}_0$, $\sim_X$ is an equivalence relation on RX such that, for any $f : X \to Y$, the function $R(f)$ maps related elements in RX to related elements in RY. Taking the pointwise quotient, we obtain a quotient $\pi : R \to \overline{R}$ in the functor category, satisfying the usual universal property. We want to equip $\overline{R}$ with a monad structure that upgrades $\pi : R \to \overline{R}$ into a quotient in the category of monads. In particular, this means that we need to fill in the square

$$\begin{tikzcd} R \cdot R \arrow[r, "\mu"] \arrow[d, "\pi \cdot \pi"'] & R \arrow[d, "\pi"] \\ \overline{R} \cdot \overline{R} \arrow[r, dashed, "\overline{\mu}"'] & \overline{R} \end{tikzcd}$$

with a suitable $\overline{\mu} : \overline{R} \cdot \overline{R} \longrightarrow \overline{R}$ satisfying the monad laws. But since π, and hence $\pi \cdot \pi$, is epi as R preserves epimorphisms, this is possible when any two elements in RRX that are mapped to the same element by $\pi \cdot \pi$ (the left vertical morphism) are also mapped to the same element by $\pi \circ \mu$ (the top-right composition). It turns out that this is the only extra condition needed for the upgrade. We summarize the construction in the following lemma:

▶ **Lemma 49** (projR_monad). *Given a monad R preserving epimorphisms, and a compatible relation $\sim$ on R such that for any set X and $x, y \in RRX$, we have that if $(\pi \cdot \pi)_X(x) \sim (\pi \cdot \pi)_X(y)$ then $\pi(\mu(x)) \sim \pi(\mu(y))$. Then we can construct the quotient $\pi : R \to \overline{R}$ in the category of monads, satisfying the usual universal property.*

▶ **Definition 50.** *An* epi-signature *is a signature Σ that preserves the epimorphicity in the category of endofunctors on Set: for any monad morphism $f : R \longrightarrow S$, if $U(f)$ is an epi of functors, then so is $U(\Sigma(f))$. Here, we denote by U the forgetful functor from monads resp. modules to endofunctors.*

If we admit the axiom of choice, then epimorphisms in Set have a retraction, and thus any endofunctor on Set preserves epimorphisms. Hence, in that case, any signature is an epi-signature, and the previous definition becomes superfluous.

▶ **Example 51** (BindingSigAreEpiSig). Any algebraic signature is an epi-signature.

We are now in a position to state and prove the main technical lemma:

▶ **Lemma 52** (push_initiality). *Let Υ be representable, such that both $\hat{\Upsilon}$ and $\Upsilon(\hat{\Upsilon})$ preserve epimorphisms (as noted above, this condition is automatically fulfilled if one assumes the axiom of choice). Let $F : \Upsilon \to \Sigma$ be a morphism of signatures. Suppose that Υ is an epi-signature and F is an epimorphism. Then Σ is representable.*

Sketch of the proof. As before, we denote by $\hat{\Upsilon}$ the initial Υ-model, as well as—by abuse of notation—its underlying monad. For each set X, we consider the equivalence relation $\sim_X$ on $\hat{\Upsilon}(X)$ defined as follows: for all $x, y \in \hat{\Upsilon}(X)$ we stipulate that $x \sim_X y$ if and only if $i_X(x) = i_X(y)$ for each (initial) morphism of Υ-models $i : \hat{\Upsilon} \to F^*S$ with S a Σ-model and F^*S the Υ-model induced by $F : \Upsilon \to \Sigma$.

By virtue of Lemma 49, since $\hat{\Upsilon}$ preserves epimorphisms, we obtain the quotient monad, which we call $\hat{\Upsilon}/F$, and the epimorphic projection $\pi : \hat{\Upsilon} \to \hat{\Upsilon}/F$. We now equip $\hat{\Upsilon}/F$ with a Σ-action, and show that the induced model is initial, in four steps:

(i) We equip $\hat{\Upsilon}/F$ with a Σ-action, i.e., with a morphism of $\hat{\Upsilon}/F$-modules $m_{\hat{\Upsilon}/F} : \Sigma(\hat{\Upsilon}/F) \to \hat{\Upsilon}/F$. We define $u : \Upsilon(\hat{\Upsilon}) \to \Sigma(\hat{\Upsilon}/F)$ as $u = F_{\hat{\Upsilon}/F} \circ \Upsilon(\pi)$. Then u is epimorphic, by composition of epimorphisms and by using Corollary 48. Let $m_{\hat{\Upsilon}} : \Upsilon(\hat{\Upsilon}) \to \hat{\Upsilon}$ be the action of the initial model of Υ. We define $m_{\hat{\Upsilon}/F}$ as the unique morphism making the following diagram commute in the category of endofunctors on Set:

$$\begin{tikzcd}
\Upsilon(\hat{\Upsilon}) \arrow[r, "m_{\hat{\Upsilon}}"] \arrow[d, "u"'] & \hat{\Upsilon} \arrow[d, "\pi"] \\
\Sigma(\hat{\Upsilon}/F) \arrow[r, dashed, "m_{\hat{\Upsilon}/F}"'] & \hat{\Upsilon}/F
\end{tikzcd}$$

Uniqueness is given by the pointwise surjectivity of u. Existence follows from the compatibility of $m_{\hat{\Upsilon}}$ with the congruence $\sim_X$. The diagram necessary to turn $m_{\hat{\Upsilon}/F}$ into a module morphism on $\hat{\Upsilon}/F$ is proved by pre-composing it with the epimorphism $(\Sigma(\pi) \circ F_{\hat{\Upsilon}}) \cdot \pi : \Upsilon(\hat{\Upsilon}) \cdot \hat{\Upsilon} \to \Sigma(\hat{\Upsilon}/F) \cdot \hat{\Upsilon}/F$ (this is where the preservation of epimorphims by $\Upsilon(\hat{\Upsilon})$ is required) and unfolding the definitions.

(ii) Now, π can be seen as a morphism of Υ-models between $\hat{\Upsilon}$ and $F^*\hat{\Upsilon}/F$, by naturality of F and using the previous diagram.
It remains to show that $(\hat{\Upsilon}/F, m_{\hat{\Upsilon}/F})$ is initial in the category of Σ-models.

(iii) Given a Σ-model (S, m_s), the initial morphism of Υ-models $i_S : \hat{\Upsilon} \to F^* S$ induces a monad morphism $\iota_S : \hat{\Upsilon}/F \to S$. We need to show that the morphism ι is a morphism of Σ-models. Pre-composing the involved diagram by the epimorphism $\Sigma(\pi) \circ F_{\hat{\Upsilon}} : \Upsilon(\hat{\Upsilon}) \to \Sigma(\hat{\Upsilon}/F)$ and unfolding the definitions show that $\iota_S : \hat{\Upsilon}/F \to S$ is a morphism of Σ-models.

(iv) We show that ι_S is the only morphism $\hat{\Upsilon}/F \to S$. Let g be such a morphism. Then $g \circ \pi : \hat{\Upsilon} \to S$ defines a morphism in the category of Υ-models. Uniqueness of i_S yields $g \circ \pi = i_S$, and by uniqueness of the diagram defining ι_S it follows that $g = i'_S$. ◀

▶ **Lemma 53** (`algebraic_model_Epi` and `BindingSig_on_model_isEpi`)**.** *Let Σ be an algebraic signature. Then $\hat{\Sigma}$ and $\Sigma(\hat{\Sigma})$ preserve epimorphisms.*

Proof. The initial model of an algebraic signature Σ is obtained as the initial chain of the endofunctor $R \mapsto \mathsf{Id} + \Sigma(R)$, where Σ denotes (by abuse of notation) the endofunctor on endofunctors on Set corresponding to the signature Σ. Then the proof follows from the fact that this endofunctor preserves preservation of epimorphisms. ◀

Proof of Thm. 43. Let $p : \Upsilon \to \Sigma$ be a presentation of Σ. We need to construct a representation for Σ.

Since the signature Υ is algebraic, it is representable (by Theorem 36), and it is an episignature (by Example 51). We can thus instantiate Lemma 52 to see that Σ is representable, thanks to Lemma 53. ◀

8 Examples of presentable signatures

Complex signatures are naturally built as the sum of basic components, generally referred as 'arities' (which in our settings are signatures themselves, see remark after Example 19). Thanks to Proposition 46, direct sums (or, more generally, colimits) of presentable signatures are presentable, hence representable by Theorem 43.

In this section, we show that, besides algebraic signatures, there are other interesting examples of signatures which are presentable, and which hence can be *safely* added to any presentable signature. *Safely* here means that the resulting signature is still presentable.

8.1 Post-composition with a presentable functor

A functor $F : \mathsf{Set} \to \mathsf{Set}$ is *polynomial* if it is of the form $FX = \coprod_{n \in \mathbb{N}} a_n \times X^n$ for some sequence $(a_n)_{n \in \mathbb{N}}$ of sets. Note that if F is polynomial, then the signature $F \cdot \Theta$ is algebraic.

▶ **Definition 54.** *Let $G : \mathsf{Set} \to \mathsf{Set}$ be a functor. A* presentation of G *is a pair consisting of a polynomial functor $F : \mathsf{Set} \to \mathsf{Set}$ and an epimorphism $p : F \to G$. The functor G is called* presentable *if there is a presentation of G.*

▶ **Proposition 55.** *Given a presentable functor G, the signature $G \cdot \Theta$ is presentable.*

Proof. Let $p : F \to G$ be a presentation of G; then a presentation of $G \cdot \Theta$ is given by the induced epimorphism $F \cdot \Theta \to G \cdot \Theta$. ◀

The next statement shows how the difference between finitary endofunctors and our presentable endofunctors is small:

▶ **Proposition 56.** *Here we assume the axiom of excluded middle. An endofunctor on* Set *is presentable if and only if it is finitary (i.e., it preserves filtered colimits).*

Proof. This is a corollary of Proposition 5.2 of [2], since ω-accessible functors are exactly the finitary ones. ◀

We now give several examples of presentable signatures obtained from presentable functors.

8.1.1 Example: Adding a syntactic commutative binary operator, e.g., parallel-or

Consider the functor $\mathsf{square} : \mathsf{Set} \to \mathsf{Set}$ mapping a set X to $X \times X$; it is polynomial. The associated signature $\mathsf{square} \cdot \Theta$ encodes a binary operator, such as the application of the lambda calculus.

Sometimes such binary operators are asked to be *commutative*; a simple example of such a commutative binary operator is the addition of two numbers.

Another example, more specific to formal computer languages, is a 'concurrency' operator $P \mid Q$ of a process calculus, such as the π-calculus, for which it is natural to require commutativity as a structural congruence relation: $P \mid Q \equiv Q \mid P$.

Such a commutative binary operator can be specified via the following presentable signature: we denote by $\mathcal{S}_2 : \mathsf{Set} \to \mathsf{Set}$ the endofunctor that assigns, to each set X, the set $(X \times X)/(x, y) \sim (y, x)$ of unordered pairs of elements of X. This functor is presented by the obvious projection $\mathsf{square} \to \mathcal{S}_2$. By Proposition 55, the signature $\mathcal{S}_2 \cdot \Theta$ is presentable, it encodes a commutative binary operator.

8.1.2 Example: Adding a maximum operator

Let $\mathsf{list} : \mathsf{Set} \to \mathsf{Set}$ be the functor associating, to any set X, the set $\mathsf{list}(X)$ of (finite) lists with entries in X; specifically, it is given on objects as $X \mapsto \coprod_{n \in \mathbb{N}} X^n$.

We now consider the syntax of a "maximum" operator, acting, e.g., on a list of natural numbers:

$$\mathsf{max} : \mathsf{list}(\mathbb{N}) \to \mathbb{N}$$

It can be specified via the algebraic signature $\mathsf{list} \cdot \Theta$.

However, this signature is 'rough' in the sense that it does not take into account some semantic aspects of a maximum operator, such as invariance under repetition or permutation of elements in a list.

For a finer encoding, consider the functor $\mathcal{P}_{\mathrm{fin}} : \mathsf{Set} \to \mathsf{Set}$ associating, to a set X, the set $\mathcal{P}_{\mathrm{fin}}(X)$ of its finite subsets. This functor is presented by the epimorphism $\mathsf{list} \to \mathcal{P}_{\mathrm{fin}}$.

By Proposition 55, the signature $\mathcal{P}_{\mathrm{fin}} \cdot \Theta$ is presentable; it encodes the syntax of a 'maximum' operator accounting for invariance under repetition or permutation of elements in a list.

8.1.3 Example: Adding an application à la Differential LC

Let R be a commutative (semi)ring. To any set S, we can associate the *free R-module* $R\langle S\rangle$; its elements are formal linear combinations $\sum_{s \in S} a_s s$ of elements of S with coefficients a_s from R; with $a_s = 0$ almost everywhere. Ignoring the R-module structure on $R\langle S\rangle$, this assignment induces a functor $R\langle\_\rangle : \mathsf{Set} \to \mathsf{Set}$ with the obvious action on morphisms. For simplicity, we restrict our attention to the semiring $(\mathbb{N}, +, \times)$.

This functor is presentable: a presentation is given by the polynomial functor $\mathsf{list} : \mathsf{Set} \to \mathsf{Set}$ and the epimorphism

$$p : \mathsf{list} \longrightarrow \mathbb{N}\langle\_\rangle$$
$$p_X([x_1, \ldots, x_n]) := x_1 + \ldots + x_n \quad .$$

By Proposition 55, this yields a presentable signature, which we call $\mathbb{N}\langle\Theta\rangle$.

The Differential Lambda Calculus (DLC) [17] of Ehrhard and Regnier is a lambda calculus with operations suitable to express differential constructions. The calculus is parametrized by a semiring R; again we restrict to $R = (\mathbb{N}, +, \times)$.

DLC has a binary 'application' operator, written $(s)t$, where $s \in T$ is an element of the inductively defined set T of terms and $t \in \mathbb{N}\langle T\rangle$ is an element of the free $(\mathbb{N}, +, \times)$-module. This operator is thus specified by the presentable signature $\Theta \times \mathbb{N}\langle\Theta\rangle$.

8.2 Example: Adding a syntactic closure operator

Given a quantification construction (e.g., abstraction, universal or existential quantification), it is often useful to take the associated closure operation. One well-known example is the universal closure of a logic formula. Such a closure is invariant under permutation of the fresh variables. A closure can be syntactically encoded in a rough way by iterating the closure with respect to one variable at a time. Here our framework allows a refined syntactic encoding which we explain below.

Let us start with binding a fixed number k of fresh variables. The elementary signature $\Theta^{(k)}$ already specifies an operation that binds k variables. However, this encoding does not reflect invariance under variable permutation. To enforce this invariance, it suffices to quotient the signature $\Theta^{(k)}$ with respect to the action of the group S_k of permutations of the set k, that is, to consider the colimit of the following one-object diagram:

$$\begin{tikzcd} \Theta^{(k)} \arrow[loop above, "\Theta^{(\sigma)}"] \end{tikzcd}$$

where σ ranges over the elements of S_k. We denote by $\mathcal{S}^{(k)}\Theta$ the resulting signature presented by the projection $\Theta^{(k)} \to \mathcal{S}^{(k)}\Theta$. By universal property of the quotient, a model of it consists of a monad R with an action $m : R^{(k)} \to R$ that satisfies the required invariance.

Now, we want to specify an operation which binds an arbitrary number of fresh variables, as expected from a closure operator. One rough solution is to consider the coproduct $\coprod_k \mathcal{S}^{(k)}\Theta$. However, we encounter a similar inconvenience as for $\Theta^{(k)}$. Indeed, for each $k' > k$, each term already encoded by the signature $\mathcal{S}^{(k)}\Theta$ may be considered again, encoded (differently) through $\mathcal{S}^{(k')}\Theta$.

Fortunately, a finer encoding is provided by the following simple colimit of presentable signatures. The crucial point here is that, for each k, all natural injections from $\Theta^{(k)}$ to $\Theta^{(k+1)}$ induce the same canonical injection from $\mathcal{S}^{(k)}\Theta$ to $\mathcal{S}^{(k+1)}\Theta$. We thus have a natural colimit for the sequence $k \mapsto \mathcal{S}^{(k)}\Theta$ and thus a signature $\operatorname{colim}_k \mathcal{S}^{(k)}\Theta$ which, as a colimit of presentable signatures, is presentable (Proposition 46).

Accordingly, we define a total closure on a monad R to be an action of the signature $\operatorname{colim}_k \mathcal{S}^{(k)}\Theta$ in R. It can easily be checked that a model of this signature is a monad R together with a family of module morphisms $(e_k : R^{(k)} \to R)_{k\in\mathbb{N}}$ compatible in the sense that for each injection $i : k \to k'$ the following diagram commutes:

$$\begin{tikzcd} R^{(k)} \arrow[r, "R^{(i)}"] \arrow[dr, "e_k"'] & R^{(k')} \arrow[d, "e_{k'}"] \\ & R \end{tikzcd}$$

8.3 Example: Adding an explicit substitution

Explicit substitution was introduced by Abadi et al. [1] as a theoretical device to study the theory of substitution and to describe concrete implementations of substitution algorithms. In this section, we explain how we can extend any presentable signature with an explicit substitution construction, and we offer some refinements from a purely syntactic point of view. In fact, we will show three solutions, differing in the amount of 'coherence' which is handled at the syntactic level (e.g., invariance under permutation and weakening). We follow the approach initiated by Ghani, Uustalu, and Hamana in [24]. With respect to their work, one key difference is that our approach does not require the notion of strength.

Let R be a monad. We have already considered (see Lemma 16) the (unary) substitution $\sigma_R : R' \times R \to R$. More generally, we have the sequence of substitution operations

$$\mathsf{subst}_p : R^{(p)} \times R^p \longrightarrow R. \tag{3}$$

We say that subst_p is the p-substitution in R; it simultaneously replaces the p extra variables in its first argument with the p other arguments, respectively. (Note that subst_1 is the original σ_R).

We observe that, for fixed p, the group S_p of permutations on p elements has a natural action on $R^{(p)} \times R^p$, and that subst_p is invariant under this action.

Thus, if we fix an integer p, there are two ways to internalise subst_p in the syntax: we can choose the elementary signature $\Theta^{(p)} \times \Theta^p$, which is rough in the sense that the above invariance is not reflected; and, alternatively, if we want to reflect the permutation invariance syntactically, we can choose the quotient Q_p of the above signature by the action of S_p.

By universal property of the quotient, a model of our quotient Q_p is given by a monad R with an action $m : R^{(p)} \times R^p \to R$ satisfying the desired invariance.

Before turning to the encoding of the entire series $(\mathsf{subst}_p)_{p\in\mathbb{N}}$, we recall how, as noticed already in [24], this series enjoys further coherence. In order to explain this coherence, we start with two natural numbers p and q and the module $R^{(p)} \times R^q$. Pairs in this module are almost ready for substitution: what is missing is a map $u : p \longrightarrow q$ between the corresponding standard finite sets. But such a map can be used in two ways: letting u act covariantly on the first factor leads us into $R^{(q)} \times R^q$ where we can apply subst_q; while letting u act contravariantly on the second factor leads us into $R^{(p)} \times R^p$ where we can apply subst_p. The good news is that we obtain the same result. More precisely, the following diagram is commutative:

$$\begin{tikzcd} R^{(p)} \times R^q \arrow[r, "R^{(p)} \times R^u"] \arrow[d, "R^{(u)} \times R^q"'] & R^{(p)} \times R^p \arrow[d, "\mathsf{subst}_p"] \\ R^{(q)} \times R^q \arrow[r, "\mathsf{subst}_q"'] & R \end{tikzcd} \tag{4}$$

Note that in the case where p equals q and u is a permutation, we recover exactly the invariance by permutation considered earlier.

Abstracting over the numbers p, q and the map u, this exactly means that our series factors through the coend $\int^{p:\mathbb{N}} R^{(\underline{p})} \times R^{\overline{p}}$, where covariant (resp. contravariant) occurrences of the bifunctor have been underlined (resp. overlined), and the category $\mathbb{N}$ is the full subcategory of Set whose objects are natural numbers. Thus we have a canonical morphism

$$\mathsf{isubst}_R : \int^{p:\mathbb{N}} R^{(\underline{p})} \times R^{\overline{p}} \longrightarrow R.$$

Abstracting over R, we obtain the following:

▶ **Definition 57.** *The* integrated substitution

$$\mathsf{isubst} : \int^{p:\mathbb{N}} \Theta^{(\underline{p})} \times \Theta^{\overline{p}} \longrightarrow \Theta$$

is the signature morphism obtained by abstracting over R the linear morphisms isubst_R*.*

Thus, if we want to internalise the whole sequence $(\mathsf{subst}_p)_{p:\mathbb{N}}$ in the syntax, we have at least three solutions: we can choose the algebraic signature

$$\coprod_{p:\mathbb{N}} \Theta^{(p)} \times \Theta^p$$

which is rough in the sense that the above invariance and coherence is not reflected; we can choose the presentable signature

$$\coprod_{p:\mathbb{N}} Q_p,$$

which reflects the invariance by permutation, but not more; and finally, if we want to reflect the whole coherence syntactically, we can choose the presentable signature

$$\int^{p:\mathbb{N}} \Theta^{(\underline{p})} \times \Theta^{\overline{p}}.$$

Thus, whenever we have a presentable signature, we can safely extend it by adding one or the other of the three above signatures, for a (more or less coherent) explicit substitution.

8.4 Example: Adding a coherent fixed-point operator

In the same spirit as in the previous section, we define, in this section,

- for each $n \in \mathbb{N}$, a notion of *n-ary fixed-point operator* in a monad;
- a notion of *coherent fixed-point operator* in a monad, which assigns, in a 'coherent' way, to each $n \in \mathbb{N}$, an n-ary fixed-point operator.

We furthermore explain how to safely extend any syntax generated by a presentable signature with a syntactic coherent fixed-point operator.

There is one fundamental difference between the integrated substitution of the previous section and our coherent fixed points: while every monad has a canonical integrated substitution, this is not the case for coherent fixed-point operators.

Let us start with the unary case.

▶ **Definition 58.** *A* unary fixed-point operator for a monad R *is a module morphism* f *from* R' *to* R *that makes the following diagram commute,*

$$\begin{tikzcd} R' \arrow[rr, "(id_{R'}, f)"] \arrow[dr, "f"'] & & R' \times R \arrow[dl, "\sigma"] \\ & R & \end{tikzcd}$$

where σ *is the substitution morphism defined in Lemma 16.*

Accordingly, the signature for a syntactic unary fixpoint operator is Θ', ignoring the commutation requirement (which we plan to address in a future work by extending our framework with equations).

Let us digress here and examine what the unary fixpoint operators are for the lambda calculus, more precisely, for the monad $\mathsf{LC}_{\beta\eta}$ of the lambda-calculus modulo β- and η-equivalence. How can we relate the above notion to the classical notion of fixed-point combinator? Terms are built out of two constructions, $\mathsf{app} : \mathsf{LC}_{\beta\eta} \times \mathsf{LC}_{\beta\eta} \to \mathsf{LC}_{\beta\eta}$ and $\mathsf{abs} : \mathsf{LC}'_{\beta\eta} \to \mathsf{LC}_{\beta\eta}$. A fixed-point combinator is a term Y satisfying, for any (possibly open) term t, the equation

$$\mathsf{app}(t, \mathsf{app}(Y, t)) = \mathsf{app}(Y, t).$$

Given such a combinator Y, we define a module morphism $\hat{Y} : \mathsf{LC}'_{\beta\eta} \to \mathsf{LC}_{\beta\eta}$. It associates, to any term t depending on an additional variable $*$, the term $\hat{Y}(t) := \mathsf{app}(Y, \mathsf{abs}\ t)$. This term satisfies $t[\hat{Y}(t)/*] = \hat{Y}(t)$, which is precisely the commutativity of the diagram of Definition 58 that $\hat{Y}$ must satisfy to be a unary fixed-point operator for the monad $\mathsf{LC}_{\beta\eta}$. Conversely, we have:

▶ **Proposition 59.** *Any fixed-point combinator in* $\mathsf{LC}_{\beta\eta}$ *comes from a unique fixed-point operator.*

Proof. We construct a bijection between the set $\mathsf{LC}_{\beta\eta}(0)$ of closed terms on the one hand and the set of module morphisms from $\mathsf{LC}'_{\beta\eta}$ to $\mathsf{LC}_{\beta\eta}$ satisfying the fixed-point property on the other hand.

A closed lambda term t is mapped to the morphism $u \mapsto \hat{t}\, u := \mathsf{app}(t, \mathsf{abs}\ u)$. We have already seen that if t is a fixed-point combinator, then $\hat{t}$ is a fixed-point operator.

For the inverse function, note that a module morphism f from $\mathsf{LC}'_{\beta\eta}$ to $\mathsf{LC}_{\beta\eta}$ induces a closed term $Y_f := \mathsf{abs}(f_1(\mathsf{app}(*, **)))$ where $f_1 : \mathsf{LC}_{\beta\eta}(\{*, **\}) \to \mathsf{LC}_{\beta\eta}(\{*\})$.

A small calculation shows that $Y \mapsto \hat{Y}$ and $f \mapsto Y_f$ are inverse to each other.

It remains to be proved that if f is a fixed-point operator, then Y_f satisfies the fixed-point combinator equation. Let $t \in \mathsf{LC}_{\beta\eta}X$, then we have

$$\begin{aligned} \mathsf{app}(Y_f, t) &= \mathsf{app}(\mathsf{abs}\ f_1(\mathsf{app}(*, **)), t) && (5)\\ &= f_X(\mathsf{app}(t, **)) && (6)\\ &= \mathsf{app}(t, \mathsf{app}(Y_f, t)) && (7) \end{aligned}$$

where (6) comes from the definition of a fixed-point operator. Equality (7) follows from the equality $\mathsf{app}(Y_f, t) = f_X(\mathsf{app}(t, **))$, which is obtained by chaining the equalities from (5) to (6). This concludes the construction of the bijection. ◀

After this digression, we now turn to the n-ary case.

▶ **Definition 60.**

- *A* rough n-ary fixed-point operator *for a monad R is a module morphism $f : (R^{(n)})^n \to R^n$ making the following diagram commute:*

$$\begin{tikzcd}
(R^{(n)})^n \arrow[r, "id_{(R^{(n)})^n}, f, .., f"] \arrow[d, "f"'] & (R^{(n)})^n \times (R^n)^n \arrow[d, "\cong"] \\
R^n & (R^{(n)} \times R^n)^n \arrow[l, "(\mathsf{subst}_n)^n"]
\end{tikzcd}$$

where subst_n *is the n-substitution as in Section 8.3.*

- *An* n-ary fixed-point operator *is just a rough n-ary fixed-point operator which is furthermore invariant under the natural action of the permutation group S_n.*

The type of f above is canonically isomorphic to

$$(R^{(n)})^n + (R^{(n)})^n + \ldots + (R^{(n)})^n \to R,$$

which we abbreviate to $n \times (R^{(n)})^n \to R$.

Accordingly, a natural signature for encoding a syntactic rough n-ary fixpoint operator is $n \times (\Theta^{(n)})^n$.

Similarly, a natural signature for encoding a syntactic n-ary fixpoint operator is $(n \times (\Theta^{(n)})^n)/S_n$ obtained by quotienting the previous signature by the action of S_n.

Now we let n vary and say that a *total fixed-point operator* on a given monad R assigns to each $n \in \mathbb{N}$ an n-ary fixpoint operator on R. Obviously, the natural signature for the encoding of a syntactic total fixed-point operator is $\coprod_n (\Theta^{(n)})^n/S_n$. Alternatively, we may wish to discard those total fixed-point operators that do not satisfy some coherence conditions analogous to what we encountered in Section 8.3, which we now introduce.

Let R be a monad with a sequence of module morphisms $\mathsf{fix}_n : n \times (R^{(n)})^n \to R$. We call this family *coherent* if, for any $p, q \in \mathbb{N}$ and $u : p \to q$, the following diagram commutes:

$$\begin{tikzcd}
p \times (R^{(p)})^q \arrow[r, "p\times(R^{(p)})^u"] \arrow[d, "u\times(R^{(u)})^q"'] & p \times (R^{(p)})^p \arrow[d, "\mathsf{fix}_p"] \\
q \times (R^{(q)})^q \arrow[r, "\mathsf{fix}_q"'] & R
\end{tikzcd} \tag{8}$$

These conditions have an interpretation in terms of a coend, just as we already encountered in Section 8.3. This leads us to the following

▶ **Definition 61.** *Given a monad R, we define a* coherent fixed-point operator *on R to be a module morphism from $\int^{n:\mathbb{N}} \underline{n} \times (R^{(\underline{n})})^{\overline{n}}$ to R where, for every $n \in \mathbb{N}$, the n-th component is a (rough)*[9] *n-ary fixpoint operator.*

Thus, given a presentable signature Σ, we can safely extend it with a syntactic coherent fixed-point operator by adding the presentable signature

$$\int^{n:\mathbb{N}} \underline{n} \times (\Theta^{(\underline{n})})^{\overline{n}}$$

to Σ.

[9] As in Section 8.3, the invariance follows from the coherence.

9 Recursion

Initiality can be seen as an abstract way to present recursion. In this section, we collect several examples of recursive maps that can be derived from initiality.

9.1 Example: Translation of intuitionistic logic into linear logic

We start with an elementary example of translation of syntaxes using initiality, namely the translation of second-order intuitionistic logic into second-order linear logic [25, page 6]. The syntax of second-order intuitionistic logic can be defined with one unary operator $\neg$, three binary operators $\vee$, $\wedge$ and $\Rightarrow$, and two binding operators $\forall$ and $\exists$. The associated (algebraic) signature is $\Sigma_{LJ} = \Theta + 3 \times \Theta^2 + 2 \times \Theta'$. As for linear logic, there are four constants $\top, \bot, 0, 1$, two unary operators ! and ?, five binary operators &, ⅋, $\otimes$, $\oplus$, $\multimap$ and two binding operators $\forall$ and $\exists$. The associated (algebraic) signature is $\Sigma_{LL} = 4 \times * + 2 \times \Theta + 5 \times \Theta^2 + 2 \times \Theta'$.

By universality of the coproduct, a model of Σ_{LJ} is given by a monad R with module morphisms:

- $r_\neg : R \longrightarrow R$
- $r_\wedge, r_\vee, r_\Rightarrow : R \times R \longrightarrow R$
- $r_\forall, r_\exists : R' \longrightarrow R$

and similarly, we can decompose an action of Σ_{LL} into as many components as there are operators.

The translation will be a morphism of monads between the initial models (i.e. the syntaxes) $o : \hat{\Sigma}_{LJ} \longrightarrow \hat{\Sigma}_{LL}$ coming from the initiality of $\hat{\Sigma}_{LJ}$. Indeed, equipping $\hat{\Sigma}_{LL}$ with an action $r'_\alpha : \alpha(\hat{\Sigma}_{LL}) \longrightarrow \hat{\Sigma}_{LL}$ for each operator α of intuitionistic logic ($\neg$,$\vee$,$\wedge$,$\Rightarrow$,$\forall$ and $\exists$) yields a morphism of monads $o : \hat{\Sigma}_{LJ} \longrightarrow \hat{\Sigma}_{LL}$ such that $o(r_\alpha(t)) = r'_\alpha(\alpha(o)(t))$ for each α.

The definition of r'_α is then straightforward to devise, following the standard translation given on the right:

$$
\begin{aligned}
r'_\neg &= r_\multimap \circ (r_! \times r_0) & (\neg A)^o &:= (!A) \multimap 0\\
r'_\wedge &= r_\& & (A \wedge B)^o &:= A^o \& B^o\\
r'_\vee &= r_\oplus \circ (r_! \times r_!) & (A \vee B)^o &:= !A^o \oplus !B^o\\
r'_\Rightarrow &= r_\multimap \circ (r_! \times id) & (A \Rightarrow B)^o &:= !A^o \multimap B^o\\
r'_\exists &= r_\exists \circ r_! & (\exists x A)^o &:= \exists x !A^o\\
r'_\forall &= r_\forall & (\forall x A)^o &:= \forall x A^o
\end{aligned}
$$

The induced action of Σ_{LJ} in the monad $\hat{\Sigma}_{LL}$ yields the desired translation morphism $o : \hat{\Sigma}_{LJ} \to \hat{\Sigma}_{LL}$. Note that variables are automatically preserved by the translation because o is a monad morphism.

9.2 Example: Computing the set of free variables

As above, we denote by $\mathcal{P}X$ the powerset of X. The union gives us a composition operator $\mathcal{P}(\mathcal{P}X) \to \mathcal{P}X$ defined by $u \mapsto \bigcup_{s \in u} s$, which yields a monad structure on $\mathcal{P}$.

We now define an action of the signature of lambda calculus Σ_{LC} in the monad $\mathcal{P}$. We take the binary union operator $\cup \colon \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ as action of the application signature $\Theta \times \Theta$ in $\mathcal{P}$; this is a module morphism since binary union distributes over union of sets. Next, given $S \in \mathcal{P}(X + *)$ we define $\mathsf{Maybe}_X^{-1}(S) = S \cap X$. This defines a morphism of modules $\mathsf{Maybe}^{-1} \colon \mathcal{P}' \to \mathcal{P}$; a small calculation using a distributivity law of binary intersection over union of sets shows that this natural transformation is indeed linear. It can hence be used to model the abstraction signature Θ' in $\mathcal{P}$.

Associated to this model of Σ_{LC} in $\mathcal{P}$ we have an initial morphism $\mathsf{free}\colon \mathsf{LC} \to \mathcal{P}$. Then, for any $t \in \mathsf{LC}(X)$, the set $\mathsf{free}(t)$ is the set of free variables occurring in t.

9.3 Example: Computing the size of a term

We now consider the problem of computing the 'size' of a λ-term, that is, for any set X, a function $s_X\colon \mathsf{LC}(X) \longrightarrow \mathbb{N}$ such that

$$\begin{aligned} s_X(x) &= 0 \qquad (x \in X \text{ variable}) \\ s_X(\mathsf{abs}(t)) &= 1 + s_{X+*}(t) \\ s_X(\mathsf{app}(t,u)) &= 1 + s_X(t) + s_X(u) \end{aligned}$$

To express this map as a morphism of models, we first need to find a suitable monad underlying the target model. The first candidate, the constant functor $X \mapsto \mathbb{N}$, does not admit a monad structure; the problem lies in finding a suitable unit for the monad. (More generally, given a monad R and a set A, the functor $X \mapsto R(X) \times A$ does not admit a monad structure whenever A is not a singleton.)

This problem hints at a different approach to the original question: instead of computing the size of a term (which is 0 for a variable), we compute a generalized size gs which depends on arbitrary (formal) sizes attributed to variables. We have

$$gs : \prod_{X:\mathsf{Set}} \Big(\mathsf{LC}(X) \to (X \to \mathbb{N}) \to \mathbb{N}\Big)$$

Here, unsurprisingly, we recognize the continuation monad (see also [34] for the use of continuation for implementing complicated recursion schemes using initiality)

$$\mathsf{Cont}_{\mathbb{N}} := X \mapsto (X \to \mathbb{N}) \to \mathbb{N}$$

with multiplication $\lambda f.\lambda g.f(\lambda h.h(g))$.

Now we can define gs through initiality by endowing the monad $\mathsf{Cont}_{\mathbb{N}}$ with a structure of Σ_{LC}-model as follows.

The function $\alpha(m,n) = 1 + m + n$ induces a natural transformation

$$c_{\mathsf{app}}\colon \mathsf{Cont}_{\mathbb{N}} \times \mathsf{Cont}_{\mathbb{N}} \longrightarrow \mathsf{Cont}_{\mathbb{N}}$$

and thus an action for the application signature $\Theta \times \Theta$ in the monad $\mathsf{Cont}_{\mathbb{N}}$.

Next, given a set X and $k : X \to \mathbb{N}$, define $\hat{k} : X + \{*\} \to \mathbb{N}$ by $\hat{k}(x) = k(x)$ for all $x \in X$ and $\hat{k}(*) = 0$. This induces a function

$$\begin{array}{rcl} c_{\mathsf{abs}}(X)\colon \mathsf{Cont}'_{\mathbb{N}}(X) & \longrightarrow & \mathsf{Cont}_{\mathbb{N}}(X) \\ t & \mapsto & (k \mapsto 1 + t(\hat{k})) \end{array}$$

which is the desired action of the abstraction signature Θ'.

Altogether, the transformations c_{app} and c_{abs} form the desired action of Σ_{LC} in $\mathsf{Cont}_{\mathbb{N}}$ and thus give an initial morphism, i.e., a natural transformation $\iota\colon \mathsf{LC} \to \mathsf{Cont}_{\mathbb{N}}$ which respects the Σ_{LC}-model structure. Now let 0_X be the function that is constantly zero on X. Then the sought 'size' map $s : \prod_{X:\mathsf{Set}} \mathsf{LC}(X) \to \mathbb{N}$ is given by $s_X(t) = \iota_X(t, 0_X)$.

9.4 Example: Counting the number of redexes

We now consider a function r such that $r(t)$ is the number of redexes of the λ-term t of $\mathsf{LC}(X)$. Informally, the equations defining r are

$$\begin{aligned} r(x) &= 0, \qquad (x \text{ variable}) \\ r(\mathsf{abs}(t)) &= r(t), \\ r(\mathsf{app}(t,u)) &= r(t) + r(u) + \begin{cases} 1 & \text{if } t \text{ is an abstraction} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

In order to compute recursively the number of β-redexes in a term, we need to keep track, not only of the number of redexes in subterms, but also whether the head construction of subterms is the abstraction; in the affirmative case we use the value 1, and otherwise we use 0. Hence, we define a Σ_{LC}-action on the monad $W := \mathsf{Cont}_{\mathbb{N}\times\{0,1\}}$. We denote by π_1, π_2 the projections that access the two components of the product $\mathbb{N} \times \{0,1\}$.

For any set X and function $k : X \to \mathbb{N} \times \{0,1\}$, let us denote by $\hat{k} : X + \{*\} \to \mathbb{N} \times \{0,1\}$ the function which sends $x \in X$ to $k(x)$ and $*$ to $(0,0)$. Now, given a set X, consider the function

$$\begin{array}{rcl} c_{\mathsf{abs}}(X) \colon W'(X) & \longrightarrow & W(X) \\ t & \mapsto & (k \mapsto (\pi_1(t(\hat{k})), 1)). \end{array}$$

Then c_{abs} is an action of the abstraction signature Θ' in W.

Next, we specify an action $c_{\mathsf{app}} : W \times W \to W$ of the application signature $\Theta \times \Theta$: Given a set X, consider the function

$$\begin{array}{rcl} c_{\mathsf{app}}(X) \colon W(X) \times W(X) & \longrightarrow & W(X) \\ (t,u) & \mapsto & (k \mapsto (\pi_1(t(k)) + \pi_1(u(k)) + \pi_2(t(k)), 0)). \end{array}$$

Then c_{app} is an action of the abstraction signature $\Theta \times \Theta$ in W.

Overall we have a Σ_{LC}-action from which we get an initial morphism $\iota \colon \mathsf{LC} \to W$. If 0_X is the constant function $X \to \mathbb{N} \times \{0,1\}$ returning the pair $(0,0)$, then $\pi_1(\iota(0_X)) : \mathsf{LC}(X) \to \mathbb{N}$ is the desired function r.

10 Conclusion

We have presented notions of *signature* and *model of a signature*. A *representation of a signature* is an initial object in its category of models—a *syntax*. We have defined a class of *presentable* signatures, which contains traditional algebraic signatures, and which is closed under various operations, including colimits. One of our main results says that any presentable signature is representable.

One difference to other work on Initial Semantics, e.g., [38, 23, 18, 20], is that we do not rely on the notion of strength. However, a signature endofunctor with strength as used in the aforementioned articles can be translated to a high-level signature as presented in this work (Proposition 25).

This paper is a counterpart to Chapter 3 of Lafont's PhD thesis [35]. The other chapters there treat related topics; they provide

- a systematic approach to *equations* for our notions of signature and model of a signature, see [5] and [35, Chapter 4]; and
- two extensions of the present work accounting for operational semantics, see [6] and [35, Chapter 5], and [28] and [35, Chapter 6], respectively.

References

1 M. Abadi, L. Cardelli, P.-L. Curien, and J.-J. Levy. Explicit substitutions. In *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '90, pages 31–46, New York, NY, USA, 1990. ACM. URL: http://doi.acm.org/10.1145/96709.96712, doi:10.1145/96709.96712.

2 J. Adámek and H.-E. Porst. On tree coalgebras and coalgebra presentations. *Theor. Comput. Sci.*, 311(1-3):257–283, January 2004. URL: http://dx.doi.org/10.1016/S0304-3975(03)00378-5, doi:10.1016/S0304-3975(03)00378-5.

3 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26:3–37, 2016. doi:10.1017/S0960129514000103.

4 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. High-Level Signatures and Initial Semantics. In Dan Ghica and Achim Jung, editors, *27th EACSL Annual Conference on Computer Science Logic (CSL 2018)*, volume 119 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 4:1–4:22, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2018/9671, doi:10.4230/LIPIcs.CSL.2018.4.

5 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: https://doi.org/10.4230/LIPIcs.FSCD.2019.6, doi:10.4230/LIPIcs.FSCD.2019.6.

6 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Reduction monads and their signatures. *PACMPL*, 4(POPL):31:1–31:29, 2020. URL: https://doi.org/10.1145/3371099, doi:10.1145/3371099.

7 Benedikt Ahrens, Krzysztof Kapulkin, and Michael Shulman. Univalent categories and the rezk completion. *Mathematical Structures in Computer Science*, 25(5):1010–1039, 2015. URL: https://doi.org/10.1017/S0960129514000486, doi:10.1017/S0960129514000486.

8 Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed categories. *Logical Methods in Computer Science*, 15(1), 2019. URL: https://doi.org/10.23638/LMCS-15(1:20)2019, doi:10.23638/LMCS-15(1:20)2019.

9 Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg. From Signatures to Monads in UniMath. *Journal of Automated Reasoning*, 2018. doi:10.1007/s10817-018-9474-4.

10 Benedikt Ahrens and Julianna Zsido. Initial semantics for higher-order typed syntax in coq. *J. Formalized Reasoning*, 4(1):25–69, 2011. URL: https://doi.org/10.6092/issn.1972-5787/2066, doi:10.6092/issn.1972-5787/2066.

11 Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:3)2015.

12 Thorsten Altenkirch and Bernhard Reus. Monadic presentations of lambda terms using generalized inductive types. In Jörg Flum and Mario Rodríguez-Artalejo, editors, *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, volume 1683 of *Lecture Notes in Computer Science*, pages 453–468. Springer, 1999. doi:10.1007/3-540-48168-0_32.

13 Françoise Bellegarde and James Hook. Subsitution: A formal methods case study using monads and transformations. *Sci. Comput. Program.*, 23(2-3):287–311, 1994. URL: https://doi.org/10.1016/0167-6423(94)00022-0, doi:10.1016/0167-6423(94)00022-0.

14 Richard S. Bird and Oege de Moor. *Algebra of programming*. Prentice Hall International series in computer science. Prentice Hall, 1997.

15 Richard S. Bird and Ross Paterson. Generalised folds for nested datatypes. *Formal Asp. Comput.*, 11(2):200–222, 1999. doi:10.1007/s001650050047.

16 Martin Brandenburg. *Tensor categorical foundations of algebraic geometry*. PhD thesis, Universität Münster, 2014. arXiv:1410.1716.

17 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003. URL: https://doi.org/10.1016/S0304-3975(03)00392-X, doi:10.1016/S0304-3975(03)00392-X.

18 Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 57–68. IEEE Computer Society, 2008. URL: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=4557886, doi:10.1109/LICS.2008.38.

19 Marcelo P. Fiore and Chung-Kil Hur. Second-order equational logic (extended abstract). In Anuj Dawar and Helmut Veith, editors, *Computer Science Logic, 24th International Workshop, CSL 2010, 19th Annual Conference of the EACSL, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. URL: https://doi.org/10.1007/978-3-642-15205-4_26, doi:10.1007/978-3-642-15205-4_26.

20 Marcelo P. Fiore and Ola Mahmoud. Second-order algebraic theories - (extended abstract). In Petr Hlinený and Antonín Kucera, editors, *Mathematical Foundations of Computer Science 2010, 35th International Symposium, MFCS 2010, Brno, Czech Republic, August 23-27, 2010. Proceedings*, volume 6281 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2010. URL: https://doi.org/10.1007/978-3-642-15155-2_33, doi:10.1007/978-3-642-15155-2_33.

21 Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi. Abstract syntax and variable binding. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 193–202. IEEE Computer Society, 1999. URL: https://doi.org/10.1109/LICS.1999.782615, doi:10.1109/LICS.1999.782615.

22 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 214–224. IEEE Computer Society, 1999. URL: https://doi.org/10.1109/LICS.1999.782617, doi:10.1109/LICS.1999.782617.

23 Neil Ghani and Tarmo Uustalu. Explicit substitutions and higher-order syntax. In *Eighth ACM SIGPLAN International Conference on Functional Programming, Workshop on Mechanized reasoning about languages with variable binding, MERLIN 2003, Uppsala, Sweden, August 2003*. ACM, 2003. URL: https://doi.org/10.1145/976571.976580, doi:10.1145/976571.976580.

24 Neil Ghani, Tarmo Uustalu, and Makoto Hamana. Explicit substitutions and higher-order syntax. *Higher-Order and Symbolic Computation*, 19(2-3):263–282, 2006. doi:10.1007/s10990-006-8748-4.

25 Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, January 1987. doi:10.1016/0304-3975(87)90045-4.

26 Joseph A. Goguen, James W. Thatcher, and Eric G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, editor, *Current Trends in Programming Methodology, IV: Data Structuring*, pages 80–144. Prentice-Hall, 1978.

27 Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993. doi:10.1145/138027.138060.

28 André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. Modules over monads and operational semantics. Preprint, March 2020. URL: https://hal.archives-ouvertes.fr/hal-02338144.

29 André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In D. Leivant and R. J. G. B. de Queiroz, editors, *WoLLIC*, volume 4576 of *Lecture Notes in Computer Science*, pages 218–237. Springer, 2007. doi:10.1007/978-3-540-73445-1_16.

30 André Hirschowitz and Marco Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, May 2010. Special Issue: 14th Workshop on Logic, Language, Information and Computation (WoLLIC 2007). doi:10.1016/j.ic.2009.07.003.

31 André Hirschowitz and Marco Maggesi. Initial semantics for strengthened signatures. In Dale Miller and Zoltán Ésik, editors, *Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012, Tallinn, Estonia, 24th March 2012.*, volume 77 of *EPTCS*, pages 31–38, 2012. URL: https://doi.org/10.4204/EPTCS.77.5, doi:10.4204/EPTCS.77.5.

32 Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *Logical Methods in Computer Science*, 9(3):10, 2013. 19 pages. doi:10.2168/LMCS-9(3:10)2013.

33 Martin Hyland and John Power. The category theoretic understanding of universal algebra: Lawvere theories and monads. *Electronic Notes in Theoretical Computer Science*, 172:437–458, April 2007. doi:10.1016/j.entcs.2007.02.019.

34 Patricia Johann and Neil Ghani. Initial algebra semantics is enough! In *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, pages 207–222, 2007. doi:10.1007/978-3-540-73228-0_16.

35 Ambroise Lafont. *Signatures and models for syntax and operational semantics in the presence of variable binding*. PhD thesis, Ecole nationale superieure Mines-Telecom Atlantique Bretagne Pays de la Loire - IMT Atlantique, 2019. https://arxiv.org/abs/1910.09162.

36 Joachim Lambek. A fixed point theorem for complete categories. *Mathematische Zeitschrift*, 103:151–161, 1968.

37 Saunders Mac Lane. *Categories for the working mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1998.

38 Ralph Matthes and Tarmo Uustalu. Substitution in non-wellfounded syntax with variable binding. *Theor. Comput. Sci.*, 327(1-2):155–174, 2004. doi:10.1016/j.tcs.2004.07.025.

39 The Coq development team. *The Coq Proof Assistant, version 8.11.0*, 2020. Version 8.11. URL: http://coq.inria.fr.

40 Vladimir Voevodsky, Benedikt Ahrens, Daniel Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at https://github.com/UniMath/UniMath.