UNIVERSITY OF BIRMINGHAM
University of Birmingham Research at Birmingham

High-level signatures and initial semantics

Ahrens, Benedikt; Hirschowitz, Andre; Lafont, Ambroise; Maggesi, Marco

DOI:
10.4230/LIPIcs.CSL.2018.4

License:
Creative Commons: Attribution (CC BY)

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Ahrens, B, Hirschowitz, A, Lafont, A & Maggesi, M 2018, High-level signatures and initial semantics. in DR Ghica & A Jung (eds), *27th EACSL Annual Conference on Computer Science Logic 2018 (CSL 2018).* Leibniz International Proceedings in Informatics (LIPIcs), vol. 119, Schloss Dagstuhl, pp. 4:1 - 4:22, 27th EACSL Annual Conference on Computer Science Logic 2018 (CSL 2018), Birmingham, United Kingdom, 4/09/18. https://doi.org/10.4230/LIPIcs.CSL.2018.4

Link to publication on Research at Birmingham portal

Publisher Rights Statement:
Checked for eligibility: 10/10/2018

General rights
Unless a licence is specified above, all rights (including copyright and moral rights) in this document are retained by the authors and/or the copyright holders. The express permission of the copyright holder must be obtained for any use of this material other than for purposes permitted by law.

•Users may freely distribute the URL that is used to identify this publication.
•Users may download and/or print one copy of the publication from the University of Birmingham research portal for the purpose of private study or non-commercial research.
•User may use extracts from the document in line with the concept of 'fair dealing' under the Copyright, Designs and Patents Act 1988 (?)
•Users may not further distribute the material nor use it for the purposes of commercial gain.

Where a licence is displayed above, please note the terms and conditions of the licence govern your use of this document.

When citing, please reference the published version.

Take down policy
While the University of Birmingham exercises care and attention in making items available there are rare occasions when an item has been uploaded in error or has been deemed to be commercially or otherwise sensitive.

If you believe that this is the case for this document, please contact UBIRA@lists.bham.ac.uk providing details and we will remove access to the work immediately and investigate.

Download date: 04. May. 2024

High-Level Signatures and Initial Semantics

Benedikt Ahrens
University of Birmingham, UK
B.Ahrens@cs.bham.ac.uk
https://orcid.org/0000-0002-6786-4538

André Hirschowitz
Université Nice Sophia Antipolis, France
ah@unice.fr
https://orcid.org/0000-0003-2523-1481

Ambroise Lafont
IMT Atlantique
Inria, LS2N CNRS, France
ambroise.lafont@inria.fr
https://orcid.org/0000-0002-9299-641X

Marco Maggesi[1]
Università degli Studi di Firenze, Italy
marco.maggesi@unifi.it
https://orcid.org/0000-0003-4380-7691

Abstract

We present a device for specifying and reasoning about syntax for datatypes, programming languages, and logic calculi. More precisely, we consider a general notion of "signature" for specifying syntactic constructions. Our signatures subsume classical algebraic signatures (i.e., signatures for languages with variable binding, such as the pure lambda calculus) and extend to much more general examples.

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. Our notions of signature and syntax are suited for compositionality and provide, beyond the desired algebra of terms, a well-behaved substitution and the associated inductive/recursive principles.

Our signatures are "general" in the sense that the existence of an associated syntax is not automatically guaranteed. In this work, we identify a large and simple class of signatures which do generate a 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, signatures, syntax, monadic substitution, computer-checked proofs

Digital Object Identifier 10.4230/LIPIcs.CSL.2018.4

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

[1] Supported by GNSAGA-INdAM and MIUR.

© Benedikt Ahrens, André Hirschowitz, Marco Maggesi, and Ambroise Lafont;
licensed under Creative Commons License CC-BY
27th EACSL Annual Conference on Computer Science Logic (CSL 2018).
Editors: Dan Ghica and Achim Jung; Article No. 4; pp. 4:1–4:22
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Funding This work has partly been funded by the CoqHoTT ERC Grant 637339. This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-17-1-0363.

Acknowledgements We would like to thank the anonymous referees for their helpful comments. Their constructive criticism led us to a deep revision of our presentation.

1 Introduction

1.1 Initial Semantics

The concept of characterizing data through an initiality property is standard in computer science, where it is known under the terms *Initial Semantics* and *Algebraic Specification* [21], and has been popularized by the movement of *Algebra of Programming* [5].

This concept offers the following methodology to define a *formal language*[2]:

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

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 one hand, our signatures are fairly more general than those introduced in some of the seminal papers on this topic [10, 15, 11], 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.

The 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 7). Examples are given in Section 8.

Our notions of signature and syntax enjoy modularity in the sense introduced by [13]: 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 a previous attempt [18] 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.

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

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

1.2 Computer-checked formalization

The intricate nature of our main result made it desirable to provide a mechanically checked proof of that result, in conjunction with a human-readable summary of the proof.

Our computer-checked proof is based on the UniMath library [26], which itself is based on the proof assistant Coq [25]. 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 [2].

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

Here below, we give in `teletype font` the name of the corresponding result in the computer-checked library, when available – often in the format `filename:identifier`.

1.3 Related work

The idea that the notion of monad is suited for modeling substitution concerning syntax (and semantics) has been retained by many contributions on the subject (see e.g. [6, 13, 24, 4]).

Matthes, Uustalu [24], followed by Ghani, Uustalu, and Hamana [13], are the first to consider a form of colimits (namely coends) of signatures. Their treatment rests on the technical device of *strength*[4] and so did our preliminary version of the present work [18]. Notably, the present version simplifies the treatment by avoiding the consideration of strengths.

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

Fiore, Plotkin, Turi [10] develop a notion of substitution monoid. Following [3], this setting can be rephrased in terms of relative monads and modules over them [1]. Accordingly, our present contribution could probably be customized for this "relative" approach.

The work by Fiore with collaborators [10, 8, 9] and the work by Uustalu with collaborators [24, 13] 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 [11] 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 [20], 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 [19].

1.4 Organisation of the paper

Section 2 gives a succinct account of modules over a monad. 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 show our modularity result about merging extensions of syntax. In Section 6

[4] A (tensorial) strength for a functor $F : V \to V$ is given by a natural transformation $\beta_{v,w} : v \otimes Fw \to F(v \otimes w)$ commuting suitably with the associator and the unitor of the monoidal structure on V.

we show through examples how recursion can be recovered from initiality. Our notions of presentable signature and presentable syntax appear in Section 7. Finally, in Section 8, we give examples of presentable signatures and syntaxes.

2 Categories of modules over monads

2.1 Modules over monads

We recall only the definition and some basic facts about modules over a monad in the specific case of the category Set of sets, although most definitions are generalizable. See [17] for a more extensive introduction on this topic.

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 equations hold:

$$\mu \cdot \mu R = \mu \cdot R\mu, \qquad \mu \cdot \eta R = 1_R, \qquad \mu \cdot R\eta = 1_R.$$

Let R be a monad.

▶ **Definition 1** (Modules)**.** A left R-module is given by a functor $M\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ equipped with a natural transformation $\rho\colon M \cdot R \longrightarrow M$, called *module substitution*, which is compatible with the monad composition and identity:

$$\rho \cdot \rho R = \rho \cdot M\mu, \qquad \rho \cdot M\eta = 1_M.$$

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 2.**

- 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 evident way).
- For every set W we denote by $\underline{\mathrm{W}}\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ the constant functor $\underline{\mathrm{W}} := X \mapsto W$. Then $\underline{\mathrm{W}}$ is trivially an R-module since $\underline{\mathrm{W}} = \underline{\mathrm{W}} \cdot R$.
- Let M_1, M_2 be two R-modules. Then the product functor $M_1 \times M_2$ is an R-module (see Proposition 4 for a general statement).

▶ **Definition 3** (Linearity)**.** We say that a natural transformation of R-modules $\tau\colon M \longrightarrow N$ is *linear*[5] if it is compatible with module substitution on either side:

$$\tau \cdot \rho^M = \rho^N \cdot \tau R.$$

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 $\mathrm{Mod}(R)$.

[5] Given a monoidal category $\mathcal{C}$, there is a notion of (left or right) module over a monoid object in $\mathcal{C}$ (see `https://ncatlab.org/nlab/show/module+over+a+monoid` 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.

Limits and colimits in the category of modules can be constructed point-wise:

▶ **Proposition 4.** $\mathrm{Mod}(R)$ *is complete and cocomplete.*

See `LModule_Colims_of_shape` and `LModule_Lims_of_shape` in `Prelims/LModuleColims` for the formalized proofs.

2.2 The total category of modules

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

▶ **Definition 5** (Pullback)**.** Let $f\colon R \longrightarrow S$ be a morphism of monads[6] and M an S-module. The module substitution $M \cdot R \xrightarrow{Mf} M \cdot S \xrightarrow{\rho} M$ defines an R-module which is called *pullback* of M along f and noted f^*M.[7]

▶ **Definition 6** (The total module category)**.** We define the *total module category* $\int_R \mathrm{Mod}(R)$ as follows[8]:

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

▶ **Proposition 7.** *The forgetful functor* $\int_R \mathrm{Mod}(R) \to \mathrm{Mon}$ *given by the first projection 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 5.

The formal proof is available as `Prelims/modules:cleaving_bmod`.

▶ **Proposition 8.** *For any monad morphism* $f : R \longrightarrow S$*, the functor* f^* *preserves limits and colimits.*

See `pb_LModule_colim_iso` and `pb_LModule_lim_iso` in `Prelims/LModuleColims` for the formalized proofs.

2.3 Derivation

For our purposes, important examples of modules are given by the following general construction. Let us denote the final object of Set as $*$.

[6] An explicit definition of morphism of monads can be found in [17].
[7] The term "pullback" is standard in the terminology of Grothendieck fibrations (see Proposition 7).
[8] 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 pseudofunctor. It overlaps with the notation for categorical ends.

▶ **Definition 9** (Derivation). For any R-module M, the *derivative* of M is the functor $M' := X \mapsto M(X+*)$. It is an R-module with the substitution $\rho' \colon M' \cdot R \longrightarrow M'$ defined as in the diagram

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

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

Derivation is a cartesian 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.

▶ **Definition 10.** Given a list of non negative integers $(a) = (a_1, \ldots, a_n)$ and a left module M over a monad R, we denote by $M^{(a)} = M^{(a_1,\ldots,a_n)}$ the module $M^{(a_1)} \times \cdots \times M^{(a_n)}$. Observe that, when $(a) = ()$ is the empty list, we have $M^{()} = *$ the final module.

▶ **Proposition 11.** *Derivation yields an endofunctor of $\int_R \mathrm{Mod}(R)$ which commutes with any functor f^* induced by a monad morphism f (Proposition 7).*

See `LModule_deriv_is_functor` in `Prelims/DerivationIsFunctorial` and `pb_deriv_to_deriv_pb_iso` in `Prelims/LModPbCommute` for the formalized proofs.

We have a natural *substitution morphism* $\sigma \colon M' \times R \longrightarrow M$ defined by $\sigma_X = \rho_X \circ w_x$, where $w_X : M(X+*) \times R(X) \to M(R(X))$ is the map

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

▶ **Lemma 12.** *The transformation σ is linear.*

See `Prelims/derivadj:substitution_laws` for the formalized proof.

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 13.** *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.*

See `Prelims/derivadj:deriv_adj` for the formalized proof.

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 14.** A *signature* is a section of the forgetful functor from the category $\int_R \mathrm{Mod}(R)$ to the category Mon.

Now we give our basic examples of signatures.

▶ **Example 15.** The assignment $R \mapsto R$ is a signature, which we denote by Θ.

▶ **Example 16.** 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 $F \cdot \Sigma$.

▶ **Example 17.** The assignment $R \mapsto *_R$, where $*_R$ denotes the final module over R, is a signature which we denote by $*$.

▶ **Example 18.** Given two signatures Σ and Υ, the assignment $R \mapsto \Sigma(R) \times \Upsilon(R)$ is a signature which we denote by $\Sigma \times \Upsilon$. In particular, $\Theta^2 = \Theta \times \Theta$ is the signature of any (first-order) binary operation, and, more generally, Θ^n is the signature of n-ary operations.

▶ **Example 19.** Given two signatures Σ and Υ, the assignment $R \mapsto \Sigma(R) + \Upsilon(R)$ is a signature which we denote by $\Sigma + \Upsilon$. In particular, $\Theta^2 + \Theta^2$ is the signature of a pair of binary operations.

This 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).

▶ **Definition 20.** For each sequence of non-negative integers $s = (s_1, \ldots, s_n)$, the assignment $R \mapsto R^{(s_1)} \times \cdots \times R^{(s_n)}$ (see Definition 10) is a signature, which we denote by $\Theta^{(s)}$, 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.** A *morphism between two signatures* $\Sigma_1, \Sigma_2\colon \mathrm{Mon} \longrightarrow \int_R \mathrm{Mod}(R)$ is a natural transformation $m\colon \Sigma_1 \longrightarrow \Sigma_2$ which, post-composed with the projection $\int_R \mathrm{Mod}(R) \longrightarrow \mathrm{Mon}$, becomes the identity. Signatures form a subcategory Sig of the category of functors from Mon to $\int_R \mathrm{Mod}(R)$.

Limits and colimits of signatures can be easily constructed point-wise:

▶ **Proposition 23.** *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.*

See `Sig_Lims_of_shape` and `Sig_Colims_of_shape` in `Signatures/SignaturesColims`, and `Sig_isDistributive` in `Signatures/PresentableSignatureBinProdR` for the formalized proofs.

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

These signatures are those which appear in [10]. For instance, the algebraic signature of the lambda-calculus is $\Sigma_{\mathsf{LC}} = \Theta^2 + \Theta'$.

4 Categories of models

We define the notion of action of a signature in a monad.

▶ **Definition 25.** Given a monad R over Set, we define an *action*[9] *of the signature* Σ *in* R to be a module morphism from $\Sigma(R)$ to R.

▶ **Example 26.** The usual $\mathsf{app}\colon \mathsf{LC}^2 \longrightarrow \mathsf{LC}$ is an action of the elementary signature Θ^2 into 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' into the monad LC. Then $\mathsf{app} + \mathsf{abs}$ is an action of the algebraic signature of the lambda-calculus $\Theta^2 + \Theta'$ into the monad LC.

▶ **Definition 27.** Given a signature Σ, we build the category Mon^Σ of *models* of Σ as follows. Its objects are pairs (R, r) of a monad R equipped with an action $r : \Sigma(R) \to R$ of Σ. A morphism from (R, r) to (S, s) is a morphism of monads $m : R \to S$ compatible with the actions in the sense that the following diagram of R-modules commutes:

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

This is equivalent to asking that the square of underlying natural transformations commutes, i.e., $m \circ r = s \circ \Sigma(m)$. 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.

▶ **Proposition 28.** *These morphisms, together with the obvious composition, turn* Mon^Σ *into a category which comes equipped with a forgetful functor to the category of monads.*

In the formalization, this category is recovered as the fiber category over Σ of the displayed category [2] of models, see `Signatures/Signature:rep_disp`.

▶ **Definition 29** (Pullback)**.** Let $f\colon \Sigma \longrightarrow \Upsilon$ be a morphism of signatures and $\mathcal{R} = (R, r)$ a model of Υ. The linear morphism $\Sigma(R) \xrightarrow{f} \Upsilon(R) \xrightarrow{r} R$ defines an action of Σ in R. The induced model of Σ is called *pullback*[10] of $\mathcal{R}$ along f and noted $f^*\mathcal{R}$.

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 this object *the syntax generated by* Σ.

5.1 Representability

▶ **Definition 30.** Given a signature Σ, a *representation of* Σ is an initial object in Mon^Σ. If such an object exists, we call it *the syntax generated by* Σ and denote it by $\hat{\Sigma}$. In this case, we also say that $\hat{\Sigma}$ *represents* Σ, and we call the signature Σ *representable*[11].

▶ **Theorem 31.** *Algebraic signatures are representable.*

---

[9] This terminology is borrowed from the vocabulary of algebras over a monad: an algebra over a monad T on a category $\mathcal{C}$ is an object X of $\mathcal{C}$ with a morphism $\nu : T(X) \longrightarrow X$ that is compatible with the multiplication of the monad. This morphism is sometimes called an action.

[10] Following the terminology introduced in Definition 5, the term "pullback" is justified by Lemma 33.

[11] 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.

This result is proved in a previous work [16, Theorems 1 and 2]. The proof goes 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. As part of the present work, we provide a computer-checked proof as `algebraic_sig_representable` in the file `Signatures/BindingSig`.

In the following we present a more general representability result: Theorem 35 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 32), was stated already in the preliminary version [18, Section 6], there without proof.

Suppose that we have a pushout square of representable signatures,

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

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 total category $\int_\Sigma \mathrm{Mon}^\Sigma$ of models:

- An object of $\int_\Sigma \mathrm{Mon}^\Sigma$ 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 $\int_\Sigma \mathrm{Mon}^\Sigma$ 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 $\int_\Sigma \mathrm{Mon}^\Sigma$ into a category.

Now for each signature Σ, we have an obvious inclusion from the fiber Mon^Σ into $\int_\Sigma \mathrm{Mon}^\Sigma$, through which we may see the syntax $\hat{\Sigma}$ of any representable signature as an object in $\int_\Sigma \mathrm{Mon}^\Sigma$. 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 $\int_\Sigma \mathrm{Mon}^\Sigma$. Hence our pushout square of representable signatures as described above yields a square in $\int_\Sigma \mathrm{Mon}^\Sigma$.

▶ **Theorem 32.** *Modularity holds in $\int_\Sigma \mathrm{Mon}^\Sigma$, in the sense that given a pushout square of representable signatures as above, the associated square in $\int_\Sigma \mathrm{Mon}^\Sigma$ is a pushout again.*

In particular, the binary coproduct of two signatures Σ_1 and Σ_2 is represented by the binary coproduct of the representations of Σ_1 and Σ_2.

Our computer-checked proof of modularity is available as `pushout_in_big_rep` in the file `Signatures/Modularity`. The proof uses, in particular, the following fact:

▶ **Lemma 33.** *The projection* $\pi : \int_\Sigma \mathrm{Mon}^\Sigma \to \mathsf{Sig}$ *is a Grothendieck fibration.*

See `rep_cleaving` in `Signatures.Signature` for the formalized proof.

6 Recursion

We now show through examples how certain forms of recursion can be derived from initiality.

6.1 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 [14, 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 $\Sigma_{LK} = \Theta + (3\times\Theta^2) + (2\times\Theta')$. 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_{LK} is given by a monad R with module morphisms:

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

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) $o : \hat{\Sigma}_{LK} \longrightarrow \hat{\Sigma}_{LL}$ that further satisfies the properties of a morphism of Σ_{LK}-models, for example $o(r_\exists(t)) = r_\exists(r_!(o(t)))$. The strategy is to use the initiality of $\hat{\Sigma}_{LK}$. 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 ($\top$,$\bot$, $\vee$,$\wedge$,$\Rightarrow$,$\forall$,$\exists$, $\in$ and $=$) yields a morphism of monads $o : \hat{\Sigma}_{LK} \longrightarrow \hat{\Sigma}_{LL}$ 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 recursive clauses 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_{LK} in the monad $\hat{\Sigma}_{LL}$ yields the desired translation morphism $o : \hat{\Sigma}_{LK} \to \hat{\Sigma}_{LL}$. Note that variables are automatically preserved by the translation because o is a monad morphism.

6.2 Example: Computing the set of free variables

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

We now define an action of the signature of lambda calculus Σ_{LC} in the monad P. We take union operator $\cup\colon P \times P \to P$ as action of the application signature $\Theta \times \Theta$; this is a module morphism since binary union distributes over union of sets. Next, given $s \in P(X + *)$ we define $\mathsf{Maybe}^{-1}(s) = s \cap X$. This defines a morphism of modules $\mathsf{Maybe}^{-1}\colon P' \to P$; 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 P.

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

6.3 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

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

This problem (and many similar other ones) does not fit directly in our vision because this computation does not commute with substitution, hence does not correspond to a (potentially initial) morphism of monads.

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

$$gs : \forall X : \mathsf{Set}, \mathsf{LC}(X) \to (X \to \mathbb{N}) \to \mathbb{N}$$

Here, we recognize the continuation monad (see also [22])

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

with multiplication $\lambda f.\lambda g.f(\lambda h.h(g))$. The sets $\mathsf{Cont}_A(\emptyset)$ and A are in natural bijection and we will identify them in what follows.

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

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

$$\alpha_+\colon \mathsf{Cont}_{\mathbb{N}} \times \mathsf{Cont}_{\mathbb{N}} \longrightarrow \mathsf{Cont}_{\mathbb{N}}$$

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

Next, given $f \in \mathsf{Cont}_{\mathbb{N}}(X + *)$, define $f' \in \mathsf{Cont}_{\mathbb{N}}(X)$ by $f'(x) = 1 + f(x)$ for all $x \in X$ and $f'(*) = 0$. This induces a natural transformation

$$\begin{array}{rcl} \beta\colon \mathsf{Cont}'_{\mathbb{N}} & \longrightarrow & \mathsf{Cont}_{\mathbb{N}} \\ f & \mapsto & f' \end{array}$$

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

Altogether, we have the desired action of Σ_{LC} in $\mathsf{Cont}_{\mathbb{N}}$ and thus 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 identically zero function on X. Then the sought "size" map is given by $s_X(x) = \iota_X(x, 0_X)$.

6.4 Example: Counting the number of redexes

We now consider an example of recursive computation: 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

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

Here the (standard) recipe is to make the desired function appear as a projection of an iterative function with values in a product. Concretely, we will proceed by first defining a Σ_{LC}-action on the monad product $W := \mathsf{Cont}_{\mathbb{N}} \times \mathsf{LC}$. First, consider the linear morphism $\beta\colon \mathsf{Cont}'_{\mathbb{N}} \to \mathsf{Cont}_{\mathbb{N}}$ given by $\beta(f)(x) = f(x)$ for all $f \in \mathsf{Cont}_{\mathbb{N}}(X + *)$ and $x \in X$. Since we have $W' = \mathsf{Cont}'_{\mathbb{N}} \times \mathsf{LC}'$, the product

$$\beta \times \mathsf{abs}\colon W' \longrightarrow W$$

is an action of the abstraction signature Θ' in W.

Next we specify the action of the application signature $\Theta \times \Theta$. Given $((u,s),(v,t)) \in W(X) \times W(X)$ and $k\colon X \to A$ we define

$$c((u,s),(v,t)) := \begin{cases} (1 + u(k) + v(k))(k) & \text{if } t \text{ is an abstraction} \\ (u(k) + v(k))(k) & \text{otherwise} \end{cases}$$

and

$$a((u,s),(v,t)) := \mathsf{app}(s,t)$$

The pair map $(c,a)\colon W \times W \to W$ is our action of app in W.

From this Σ_{LC}-action, we get an initial morphism $\iota\colon \mathsf{LC} \to \mathsf{Cont}_{\mathbb{N}} \times \mathsf{LC}$. The second component of ι is nothing but the identity morphism. By taking the projection on the first component, we find a module morphism $\pi_1 \cdot \iota\colon \mathsf{LC} \to \mathsf{Cont}_{\mathbb{N}}$. Finally, if 0_X is the constant function $X \to \mathbb{N}$ returning zero, then $\pi_1(\iota(0_X)) : \mathsf{LC}(X) \to \mathbb{N}$ is the desired function r.

7 Presentable signatures and syntaxes

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

▶ **Definition 34.** A signature Σ is *presentable*[12] if there is an algebraic signature Υ and an epimorphism of signatures $p : \Upsilon \longrightarrow \Sigma$.

▶ Remark. By definition, any construction which can be encoded through a presentable signature can alternatively be encoded through the "presenting" algebraic signature. 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.

[12] 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).

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

▶ **Theorem 35.** *Any presentable signature is representable.*

A sketch of the proof is available in Appendix A.
See `PresentableisRepresentable` in `Signatures/PresentableSignature` for the formalized proof.

▶ **Definition 36.** We call a syntax *presentable* if it is generated by a presentable signature.

Next, we give important examples of presentable signatures:

▶ **Theorem 37.** *The following hold:*
1. *Any algebraic signature is presentable.*
2. *Any colimit of presentable signatures is presentable.*
3. *The product of two presentable signatures is presentable.*
   (`Signatures/PresentableSignatureBinProdR:har_binprodR_isPresentable` in the case when one of them is Θ).

Proof. Items 1–2 are easy to prove. For Item 3, 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 $\coprod_{i,j} \Upsilon_i \times \Phi_j$. ◀

▶ **Corollary 38.** *Any colimit of algebraic signatures is representable.*

8 Examples of presentable signatures

In this section we present various constructions which, thanks to Theorem 35, can be "safely" added to a presentable syntax. *Safely* here means that the resulting signature is still presentable.

8.1 Example: Adding a syntactic binary commutative operator

Here we present a signature that could be used to formalize a binary commutative operator, for example the addition of two numbers. The elementary signature $\Theta \times \Theta$ already provides a way to extend the syntax with a constructor with two arguments. By quotienting this signature, we can enforce commutativity. To this end, consider the signature $\mathcal{S}_2 \cdot \Theta$ (see Example 16) where $\mathcal{S}_2$ is the endofunctor that assigns to each set X the set of its unordered pairs. It is presentable because the epimorphism between the square endofunctor $\Delta = X \mapsto X \times X$ and $\mathcal{S}_2$ yields an epimorphism from $\Delta \cdot \Theta \cong \Theta \times \Theta$ to $\mathcal{S}_2 \cdot \Theta$. This signature could alternatively be defined as the coequalizer of the identity morphism and the signature morphism $\mathsf{swap} : \Theta \times \Theta \to \Theta \times \Theta$ that exchanges the first and the second projection.

An action of the signature $\mathcal{S}_2 \cdot \Theta$ in a monad R is given by an operation on unordered pairs of elements of $R(X)$ for any set X, or equivalently, thanks to the universal property of the quotient, by a module morphism $m : R^2 \to R$ such that, for any set X and $a, b \in R(X)$, $m_X(a, b) = m_X(b, a)$.

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:

$$\begin{tikzcd} \Theta^{(\sigma)} \arrow[d, bend left] \\ \Theta^{(k)} \arrow[u, bend left] \end{tikzcd}$$

where σ ranges over the elements of S_k. We denote by $\mathcal{S}^{(k)}\Theta$ the resulting (presentable) signature. 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 (Theorem 37, item 2).

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

In this section, we explain how we can extend any presentable signature with an *explicit substitution* construction. 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 [13].

Let R be a monad. We have already considered (see Lemma 12) the (unary) substitution $\sigma_R : R' \times R \to R$. More generally, we have the sequence of substitution operations

$$\mathsf{subst}_p : R^{(p)} \times R^p \longrightarrow R. \tag{2}$$

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 internalize 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 [13], 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 $u : I_p \longrightarrow I_q$. 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:

$$\begin{tikzcd} R^{(p)} \times R^q \arrow[r, "R^{(p)}\times R^u"] \arrow[d, "R^{(u)}\times R^p"'] & R^{(p)} \times R^p \arrow[d, "\mathsf{subst}_p"] \\ R^{(q)} \times R^q \arrow[r, "\mathsf{subst}_q"'] & R \end{tikzcd} \tag{3}$$

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

Ghani, Uustalu, and Hamana already studied this problem in [13]. Our solution proposed here does not require the consideration of a *strength*.

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 presentable syntax 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 40.** 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 12.

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 diagram of Definition 40 that $\hat{Y}$ must satisfy to be a unary fixed point operator for the monad $\mathsf{LC}_{\beta\eta}$. Conversely, we have:

▶ **Proposition 41.** *Any fixed point combinator in $\mathsf{LC}_{\beta\eta}$ comes from a unique fixed point operator.*

The proof can be found in Appendix B.

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

▶ **Definition 42.**

- 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:

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

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[13] $n \times (R^{(n)})^n \to R$.

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:

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

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 43.** 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)[14] n-ary fixpoint operator.

Now, the natural signature for a syntactic coherent fixed point operator is $\int^{n:\mathbb{N}} \underline{n} \times (\Theta^{(\underline{n})})^{\overline{n}}$. 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 Conclusions and future work

We have presented notions of *signature* and *model of a signature*. A signature is said to be *representable* when its category of models has an initial model. We have defined a class of *presentable* signatures, which contains traditional algebraic signatures, and which is closed under various operations, including colimits. Our main result says that any presentable signature is representable.

One difference to other work on Initial Semantics, e.g., [24, 12, 7, 9], 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. In future work, we will show that this translation extends faithfully to models of signatures, and preserves initiality.

[13] In the following, we similarly write n instead of I_n in order to make equations more readable.
[14] As in Section 8.3, the invariance follows from the coherence.

Furthermore, we plan to generalize our representability criterion to encompass *explicit join* (see [24]); to generalize our notions of signature and models to *(simply-)typed* syntax; and to provide a systematic approach to *equations* for our notion of signature and models.

References

1. Benedikt Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26:3–37, 2016. doi:10.1017/S0960129514000103.
2. Benedikt Ahrens and Peter LeFanu Lumsdaine. Displayed Categories. In Dale Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction*, volume 84 of *Leibniz International Proceedings in Informatics*, pages 5:1–5:16, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2017.5.
3. 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.
4. 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.
5. Richard S. Bird and Oege de Moor. *Algebra of programming.* Prentice Hall International series in computer science. Prentice Hall, 1997.
6. Richard S. Bird and Ross Paterson. Generalised folds for nested datatypes. *Formal Asp. Comput.*, 11(2):200–222, 1999. doi:10.1007/s001650050047.
7. 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. doi:10.1109/LICS.2008.38.
8. Marcelo P. Fiore and Chung-Kil Hur. Second-order equational logic (extended abstract). In Anuj Dawar and Helmut Veith, editors, *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2010. doi:10.1007/978-3-642-15205-4_26.
9. Marcelo P. Fiore and Ola Mahmoud. Second-order algebraic theories - (extended abstract). In Petr Hlinený and Antonín Kucera, editors, *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 368–380. Springer, 2010. doi:10.1007/978-3-642-15155-2_33.
10. 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, 1999. doi:10.1109/LICS.1999.782615.
11. Murdoch J. Gabbay and Andrew M. Pitts. A New Approach to Abstract Syntax Involving Binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224, Washington, DC, USA, 1999. IEEE Computer Society Press. doi:10.1109/LICS.1999.782617.
12. Neil Ghani and Tarmo Uustalu. Explicit substitutions and higher-order syntax. In *MERLIN '03: Proceedings of the 2003 ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, pages 1–7, New York, NY, USA, 2003. ACM Press.
13. 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.
14. Jean-Yves Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, 1987. doi:10.1016/0304-3975(87)90045-4.
15. Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, jan 1993. doi:10.1145/138027.138060.

16 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.

17 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.

18 André Hirschowitz and Marco Maggesi. Initial semantics for strengthened signatures. In Dale Miller and Ésik Zoltán, editors, *Proceedings of the 8th Workshop on Fixed Points in Computer Science*, pages 31–38, 2012. doi:10.4204/EPTCS.77.

19 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.

20 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.

21 J.W. Thatcher J.A. Goguen and E.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.

22 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.

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

24 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.

25 The Coq development team. *The Coq Proof Assistant, version 8.8.0*, 2018. Version 8.8. URL: http://coq.inria.fr.

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

A Proof of Theorem 35

The results of this section, as well as Theorem 35 for which these results are used, are mechanically checked in our library; the reader may thus prefer to check the formalized statements in the library rather than their proofs in this section.

The proof of Theorem 35 rests on the more technical Lemma 48 below, which requires the notion of *epi-signature*:

▶ **Definition 44.** 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 the underlying endofunctors.

▶ **Example 45.** Any algebraic signature is an epi-signature.

This example is formalized in `Signatures/BindingSig:BindingSigAreEpiSig`.

▶ **Proposition 46.** *Epimorphisms of signatures are pointwise epimorphisms.*

Proof. The proof if formalized in `Signatures/EpiArePointwise:epiSig_is_pwEpi`. In any category, a morphism $f : a \to b$ is an epimorphism if and only if the following diagram is a pushout diagram ([23, 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, 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, this is possible when any two elements in RRX that are mapped to related elements by $\pi \cdot \pi$ (the left vertical morphism) are also mapped to related elements 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 47.** *Given a monad R, 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.*

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

▶ **Lemma 48.** *Let Υ be a representable signature. 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. We denote by R 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 $R(X)$ defined as follows: for all $x, y \in R(X)$ we stipulate that $x \sim_X y$ if and only if $i_X(x) = i_X(y)$ for each (initial) morphism of Υ-models $i : R \to F^*S$ with S a Σ-model and F^*S the Υ-model induced by $F : \Upsilon \to \Sigma$.

Per Lemma 47 we obtain the quotient monad, which we call R/F, and the epimorphic projection $\pi : R \to R/F$. We now equip R/F with a Σ-action, and show that the induced model is initial, in four steps:

(i) We equip R/F with a Σ-action, i.e., with a morphism of R/F-modules $m_{R/F} : \Sigma(R/F) \to R/F$. We define $u : \Upsilon(R) \to \Sigma(R/F)$ as $u = F_{R/F} \circ \Upsilon(\pi)$. Then u is epimorphic, by composition of epimorphisms and by using Corollary 46. Let $m_R : \Upsilon(R) \to R$ be the action of the initial model of Υ. We define $m_{R/F}$ as the unique morphism making the following diagram commute in the category of endofunctors on Set:

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

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

(ii) Now, π can be seen as a morphism of Υ-models between R and F^*R/F, by naturality of F and using the previous diagram.

It remains to show that $(R/F, m_{R/F})$ is initial in the category of Σ-models.

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

(iv) We show that ι_S is the only morphism $R/F \to S$. Let g be such a morphism. Then $g \circ \pi : R \to S$ 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 $g = i'_S$. ◀

In the formalization, this result is derived from the existence of a left adjoint to the pullback functor F^* from Σ-models to Υ-models. The right adjoint is constructed in `is_right_adjoint_functor_of_reps_from_pw_epi` in `Signatures/EpiSigRepresentability`, and transfer of representability is shown in `push_initiality` in the same file.

Proof of Thm. 35. Let Σ be presentable. We need to show that Σ is representable. By hypothesis, we have a presenting algebraic signature Υ and an epimorphism of signatures $e : \Upsilon \longrightarrow \Sigma$.

As the signature Υ is algebraic, it is representable (by Theorem 31) and is an epi-signature (by Example 45). We can thus instantiate Lemma 48 to deduce representability of Σ. ◀

B Miscellanea

Proof of Prop. 41. We construct a bijection between the set $\mathsf{LC}_{\beta\eta}\emptyset$ 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. ◀