High-level signatures and initial semantics

May 6, 2019

Contents

Abstract

This thesis deals with the specification and construction of syntaxes with equations. We work with a general notion of "signature" for specifying a syntax, defined as the initial object in a suitable category of models. This characterization, in the spirit of Initial Semantics, gives a justification of the recursion principle.

Languages with variable binding, such as the pure lambda calculus, are specifiable through the classical algebraic signatures. The first extensions to syntaxes with equations that we consider are "quotients" of these algebraic signatures. They allow, for example, to specify a binary commutative operation. However, it seems too much constrained. For example, we don't know how to specify an associative operation with such quotients.

This motivates the notion of 2-signature, consisting in two parts: a specificiation of operations through a usual signature as before, and a set of equations among them. We identify the class of "algebraic 2-signatures" for which the existence of the associated syntax is guaranteed.

Formalization

The main results presented in this thesis are computer-checked within the UniMath system. This github repository is available at the following address: `https://github.com/UniMath/largecatmodules`. Throughout the chapter, statements are annotated with their corresponding identifiers in the formalization. These identifiers are also hyperlinks to the online documentation stored at `https://initialsemantics.github.io/doc/50fd617/index.html`.

Chapitre 1

Résumé long (en français)

Cette thèse s'intéresse à la mathématisation de la notion de syntaxe. Les objectifs à long terme d'une telle modélisation sont multiples :

1. remédier à la diversité des présentations de syntaxes en proposant un canevas uniforme et standard ;

2. modulariser certaines preuves, et partager certains résultats et définitions usuels concernant la syntaxe ;

3. fournir une définition rigoureuse et accessible à ceux qui n'ont pas l'expérience ou l'intuition de la notion de syntaxe.

Différents cadres mathématiques ont déjà été proposés pour rendre compte de la notion de syntaxe. Certains permettent, par exemple, d'obtenir automatiquement une opération de substitution vérifiant de bonnes propriétés, comme l'équivalence entre effectuer deux substitutions successives et une seule, bien choisie. En tout cas, il semble que le stade de consensus et de démocratisation de ces outils n'a pas encore été atteint.

Dans cette thèse, nous nous intéressons plus particulièrement à la spécification (et la construction) de syntaxes vérifiant des équations. Afin de motiver les notions mathématiques mises en jeu, nous examinons le langage de programmation fonctionnel le plus simple que l'on puisse envisager : le lambda calcul pur. Dans la section 1.1, nous donnons une première présentation de sa syntaxe, et la dotons d'une opération de substitution. Nous expliquons ensuite, dans la section 1.2, comment la notion mathématique de monade permet

d'en rendre compte, puis, dans la section 1.3, comment la notion de morphisme de modules fournit un moyen d'exprimer une propriété essentielle des constructions de la syntaxe : la préservation de la substitution.

Dans la section 1.4, nous caractérisons la syntaxe par son principe de récurrence, que nous formulons par une propriété d'initialité. Nous expliquons dans la section 1.5 que préciser cette propriété d'initialité requiert une notion de modèle adéquate, laquelle est déterminée par la *signature* spécificiant la syntaxe.

Nous terminons cette introduction par un aperçu des options explorées dans cette thèse (section 1.6) pour traiter les syntaxes vérifiant des équations.

1.1 Présentation naïve de la syntaxe du lambda calcul

Nous donnons ici une présentation, que nous qualifions de naïve, de la syntaxe du lambda calcul, de manière à en montrer les limites. Étant donné un ensemble fixé de variables $V = \{x, y, z, \ldots\}$, l'ensemble des *termes* ou expressions valides du lambda calcul peut être caractérisé récursivement ainsi :

- chaque variable x est un terme du lambda calcul,
- si t et u sont des termes, alors $t\,u$ est un terme, appelé *application* de t à u ;
- si t est un terme, alors $\lambda x.t$ est un terme, appelé *lambda abstraction* de t, où x est une variable qui peut apparaître dans t.

L'expression $\lambda x.t$ correspond à la notation mathématique $x \mapsto t$. Il s'agit de définir une fonction dépendant de la variable x, le corps de cette fonction étant donné par le terme t. L'expression $f\ t$ correspond à la notation mathématique $f(t)$. Il s'agit de l'application de la fonction f à l'argument t.

En mathématique, le nom de la variable choisie pour définir une fonction est purement conventionnel : les fonctions $x \mapsto f(x)$ et $y \mapsto f(y)$ sont identiques. Dans le langage du lambda calcul, cela signifie que nous voulons identifier le terme $\lambda x.t$ avec le terme $\lambda y.t'$, où t' est obtenu à partir du terme t en remplaçant toutes les occurrences de la variable x par la variable y. On dit que la construction $\lambda x.t$ *lie* la variable x dans t, ou encore que x est une variable liée dans t. Les variables non liées par une lambda abstraction sont

appelées variables libres de t. Les termes $\lambda x.t$ et $\lambda y.t'$ sont dits *α-équivalents*. Plus généralement, deux termes sont α-équivalents si l'on peut renommer les variables liées de l'un pour obtenir l'autre terme. La définition précise de la relation d'α-équivalence requiert quelques précautions. Par exemple, dans le cas présent, il est sous-entendu que la variable y n'apparait pas dans t ; autrement, nous identifierions (contre notre gré) les termes $\lambda x.y$ et $\lambda y.y$.

Au delà de cette notion d'α-équivalence, la *substitution* est un aspect essentiel de la syntaxe du lambda calcul : étant donné un terme t, si nous remplaçons toutes les occurrences d'une variable x par un même terme u, nous obtenons une nouvelle expression valide, que nous notons $t[x \mapsto u]$. L'opération de substitution permet d'exprimer l'intuition mathématique suivante : le résultat d'une fonction $x \mapsto t$ appliquée à un argument u est obtenu en remplaçant la variable x dans t par u. Cette affirmation se transpose, pour le lambda calcul, en l'égalité $(\lambda x.t)\ u = t[x \mapsto u]$. En tant que langage de programmation fonctionnel, l'égalité précédente est comprise comme une étape d'exécution du programme. On dit alors que $(\lambda x.t)\ u$ se *β-réduit* vers $t[x \mapsto u]$.

L'opération de substitution se définit par récurrence sur la syntaxe du lambda calcul. Terminons cette section en mentionnant une difficulté soulevée par l'α-équivalence : nous voulons que si $\lambda x.t$ et $\lambda y.t'$ sont α-équivalents, alors substituer la même variable dans chacun d'eux fournit deux termes α-équivalents. Si la variable substituée est identique à la variable abstraite, la définition naïve $(\lambda x.t)[x \mapsto u] = \lambda x.(t[x \mapsto u])$ ne satisfait pas cette exigence. Ici, considérer que les termes α-équivalents sont effectivement identiques nous permet de proposer la définition suivante : $(\lambda x.t)[x \mapsto u] = (\lambda y.t')[x \mapsto u] = \lambda y.(t'[x \mapsto u])$.

1.2 La monade du lambda calcul

Le concept de monade fournit une contrepartie mathématique de la notion intuitive de syntaxe munie d'une opération de substitution. Nous motivons cette définition par l'exemple du lambda calcul. Au lieu de considérer un ensemble unique de termes avec un ensemble de variables V fixé à l'avance, nous regroupons les termes qui utilisent les mêmes variables libres. Notons $L(X)$ l'ensemble des termes dont les variables libres sont choisies dans un ensemble X. Dans le point de vue que nous proposons ici, les termes α-équivalents sont considérés comme identiques : ainsi, $\lambda x.x = \lambda y.y \in L(\emptyset)$.

Toute variable est en particulier un terme valide ; il y a donc une inclusion $\mathsf{var}_X : X \to L(X)$ pour tout ensemble X. D'autre part, si l'on se donne un terme t dont les variables libres sont choisies dans X, ainsi que pour toute variable $x \in X$, un terme u_x dont les variables libres sont choisies dans Y, nous pouvons effectuer la substitution simultanée de toutes les variables aparaissant dans t par les termes u_x correspondant.

Cette opération de substitution simultanée, notée $t[x \mapsto u_x]_{x \in X}$ vérifie les propriétés attendues suivantes :

- chaque variable est substituée par le terme correspondant

$$x'[x \mapsto u_x]_{x \in X} = u_{x'}$$

- la substitution identité ne modifie pas le terme

$$t[x \mapsto x]_{x \in X} = t$$

- toute succession de substitutions est équivalente à une substitution bien choisie

$$t[x \mapsto u_x]_{x \in X}[y \mapsto v_y]_{y \in Y} = t[x \mapsto u_x[y \mapsto v_y]_{y \in Y}]_{x \in X}$$

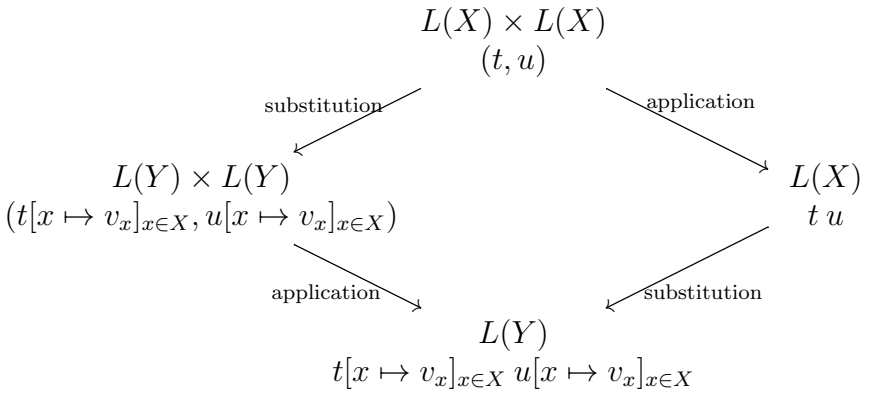
L'inclusion des variables dans les termes, l'opération de substitution simultanée satisfaisant les équations ci-dessus définissent une *monade sur la catégorie des ensembles*. Ce concept mathématique est au cœur des développements que nous exposons dans cette thèse autour de la notion de syntaxe.

1.3 Les constructions sont des morphismes de modules

Les concepts mathématiques de modules et de morphismes de modules offrent un cadre permettant d'exprimer la commutation d'une construction de la syntaxe avec l'opération de substitution. Nous illustrons cette idée pour l'application $t\ u$ du lambda calcul.

La commutation de l'application avec la substitution se traduit par l'égalité $(t\ u)[x \mapsto v_x]_{x \in X} = t[x \mapsto v_x]_{x \in X}\ u[x \mapsto v_x]_{x \in X}$. Informellement, cette équation signifie qu'il n'y a pas de différence entre effectuer la substitution

avant l'application et effectuer la substitution après l'application, comme l'exprime le diagramme commutatif suivant :



Ce constat s'appuie implicitement sur l'opération de substitution $(t, u)[x \mapsto v_x]_{x\in X} = (t[x \mapsto v_x]_{x\in X}, u[x \mapsto v_x]_{x\in X})$ dont bénéficie la collection $(L(X) \times L(X))_X$, vérifiant les propriétés attendues suivantes :

- la substitution identité ne modifie pas la paire de termes

$$(t, u)[x \mapsto x]_{x\in X} = (t, u)$$

- toute succession de substitutions est équivalente à une seule substitution bien choisie

$$(t, u)[x \mapsto u_x]_{x\in X}[y \mapsto v_y]_{y\in Y} = (t, u)[x \mapsto u_x[y \mapsto v_y]_{y\in Y}]_{x\in X}$$

À ce titre, la collection des ensembles de paires de termes définit un *module sur la monade* L, que nous notons $L \times L$.

Les définitions de monade et de module sont similaires ; d'ailleurs toute monade définit un module sur elle-même. Pour comprendre la différence, voyons comment l'on peut munir $L \times L$ d'une structure de monade. Tout d'abord, nous définissons l'inclusion des variables dans les paires de termes $x \mapsto (x, x)$. Remarquons que cette inclusion n'est pas donnée par la structure de module. Nous laissons le lecteur imaginer l'opération de substitution adéquate, étant donné une paire de termes (t, u) dont les variables libres sont choisies dans X, et pour toute variable $x \in X$, une paire de termes (v_x, w_x) dont les variables libres sont choisies dans Y. Remarquons que la substitution

donnée par la structure de module ne convient pas, puisqu'elle ne s'applique que dans le cas où l'on a associé à chaque variable un terme de la monade L, plutôt qu'une paire de termes.

L'application du lambda calcul induit une collection de fonctions $L(X) \times L(X) \to L(X)$ qui associe à toute paire (t, u) le terme $t\,u$. La propriété de commutation avec la substitution sus-mentionnée en fait un *morphisme de modules* de $L \times L$ vers L, où L est vue comme un module sur la monade homonyme.

Dans cette thèse, nous nous intéressons exclusivement à des syntaxes dont les constructions commutent avec la substitution. Par conséquent, les notions de module et de morphisme de modules y sont essentielles.

1.4 Récursion et initialité

La présentation naïve du lambda calcul induit naturellement un principe de récurrence sur la syntaxe. Supposons, par exemple, que nous voulons calculer l'ensemble des variables libres d'un terme du lambda calcul. Pour ce faire, nous raisonnons par récurrence sur la structure du terme t. Si t est une variable x, alors l'ensemble de ses variables libres est le singleton $\{x\}$. Si t est une application $u\,v$, alors l'ensemble de ses variables libres est la réunion des variables libres de u et v. Si t est une lambda abstraction $\lambda x.u$, alors n'importe quelle variable libre de u différente de x est une variable libre de t.

Dans notre cadre, nous adoptons le point de vue de la sémantique initiale : la notion de principe de récurrence est traduite par une propriété d'*initialité*. Le lambda calcul est ainsi caractérisée comme la "plus petite" monade munie d'une application et d'une lambda abstraction, dans un sens que nous allons essayer d'expliquer en reprenant l'exemple ci-dessus du calcul des variables libres (cet exemple est étudié plus formellement dans la Section 3.5.2).

Considérons la monade $\mathcal{P}$ qui associe à X l'ensemble $\mathcal{P}(X)$ de ses parties : une variable $x \in X$ induit un "terme" $\{x\} \in \mathcal{P}(X)$; la substitution $t[x \mapsto u_x]_{x \in X}$ est donnée par la réunion $\cup_{z \in t} u_z$. L'union de deux sous-ensembles fournit une opération d'application pour $\mathcal{P}$, qui au couple (t, u) associe le sous-ensemble $t \cup u$. De même, une opération adéquate d'abstraction peut être construite pour $\mathcal{P}$.

La propriété d'initialité du lambda calcul mentionnée s'instancie alors par l'existence d'une unique famille de fonctions $(\mathsf{free}_X : L(X) \to \mathcal{P}(X))_X$

vérifiant les propriétés suivantes :

- free préserve les variables

$$\mathsf{free}_X(\mathsf{var}(x)) = \{x\}$$

  (rappelons en effet que dans le cas de la monde $\mathcal{P}$, la variable x est vue comme le sous-ensemble $\{x\}$)

- free préserve la substitution

$$\mathsf{free}_Y(t[x \mapsto u_x]_{x \in X}) = \bigcup_{z \in \mathsf{free}_X(t)} \mathsf{free}_Y(u_z)$$

- free préserve l'application

$$\mathsf{free}(t\ u) = \mathsf{free}(t) \cup \mathsf{free}(u)$$

- free préserve l'abstraction

$$\mathsf{free}(\lambda x.t) = \mathsf{free}(t) \backslash \{x\}$$

Les deux premiers points caractérisent free comme un *morphisme de monades* entre L et $\mathcal{P}$. Nous expliquons dans la section suivante comment les deux derniers points en font un morphisme de modèles de la signature du lambda calcul.

1.5 Signatures et modèles

Une *signature* est une spécification d'une syntaxe. Le cadre mathématique dans lequel s'inscrit cette thèse en fournit une définition précise que nous allons motiver avec l'exemple du lambda calcul. Dans la section précédente, nous l'avons caractérisé comme la monade initiale munie d'une application et d'une lambda abstraction. Plus précisément, nous disons qu'une monade R est munie d'une application si elle est dotée d'une opération binaire, c'est-à-dire, d'un morphisme de modules $\mathsf{app}^R : R \times R \to R$.

Si l'on néglige provisoirement la partie lambda abstraction, la signature Σ_{LC} du lambda calcul associe à toute monade R le module $R \times R$ sur R. Un modèle de cette signature est une monade R munie d'un morphisme de

modules de $R \times R$ vers R, est ainsi, le lambda calcul est le modèle initial : si R est un modèle, alors il existe un morphisme de monades $f : L \to R$ qui préserve l'opération binaire.

Plus généralement, une signature Σ associe à toute monade R un module $\Sigma(R)$. Un modèle de Σ est alors une monade R munie d'un morphisme de modules $\rho : \Sigma(R) \to R$, et la syntaxe spécifiée par la signature Σ est le modèle initial $(S, \sigma^S : \Sigma(S) \to S)$ au sens suivant : étant donné un modèle $(R, \sigma^R : \Sigma(R) \to R)$, il existe un unique morphisme de monades $f : S \to R$ qui préserve σ, i.e., $f(\sigma^S(t)) = \sigma^R(f(t))$. Notons que le membre de droite requiert de donner un sens à l'expression $f(t)$ lorsque t est un élément de $\Sigma(S)_X$. Dans le cas de l'opération binaire, $\Sigma(S)_X = S(X) \times S(X)$; t est donc une paire (u, v) et l'on définit $f(u, v)$ par $(f(u), f(v))$. Dans le cas général, nous demandons que toute signature Σ vienne avec une action "fonctorielle" : tout morphisme de monades $f : R \to T$ induit un morphisme de modules[1] $\Sigma(R) \to \Sigma(T)$ sur la monade R que nous notons f, ou $\Sigma(f)$. La propriété additionnelle de fonctorialité que nous imposons signifie que cette action préserve le morphisme identité et la composition des morphismes.

Terminons cette section en mentionnant le fait que l'existence de la syntaxe associée à une signature quelconque n'est pas systématique[2]. Nous qualifions d'*effective* une telle signature. C'est le cas de toute signature que nous appelons *algébrique*, spécifiant une syntaxe avec opérations n-aires qui peuvent éventuellement lier des variables dans leurs arguments, comme par exemple le lambda calcul.

1.6 Syntaxes avec équations

Dans cette thèse, nous nous intéressons plus particulièrement à la spécification de syntaxes vérifiant des équations.

Dans le premier chapitre, nous étudions les signatures que nous appellons *présentables* : ce sont, en quelque sorte, des quotients de signatures algébriques. Nous montrons qu'elles sont effectives (Théorème 35). Ces signatures engendrent des syntaxes vérifiant des équations ; par exemple, il est

[1] Le lecteur attentif aura remarqué que $\Sigma(T)$ n'est pas un module sur R, mais sur T. Nous lui laisson le soin de vérifier qu'un morphisme de monades $R \to T$ induit une structure de module sur R pour tout module sur T.

[2] L'exemple 27 fournit un contre-exemple : il s'agit de la signature associant à toute monade R le module $(\mathcal{P}(R(X)))_X$.

possible de spécifier une opération binaire commutative (Section 3.8.1). Pour cela, il suffit de remarquer que la donnée d'une telle opération est équivalente à la donnée d'une opération prenant en argument une paire non ordonnée de termes. Donnons maintenant la signature présentable associée : il s'agit d'un quotient de la signature algébrique d'une opération binaire qui associe à toute monade R le module $R \times R$. Plus précisément, à toute monade R, la signature présentable associe le module $(R(X) \times R(X)/\sim_X)_X$, où $R(X) \times R(X)/\sim_X$ est l'ensemble des paires quotienté par la relation $(t, u) \sim (u, t)$; autrement dit, il s'agit de l'ensemble des paires non ordonnées. La syntaxe bénéficie alors d'une opération qui prend en argument un couple non ordonné de termes, comme désiré.

Néanmoins, la classe des signatures présentables paraît limitée. Par exemple, nous ne savons pas spécifier une opération binaire associative, ou bien construire la syntaxe du lambda calcul quotientée par la β-réduction. C'est pourquoi, dans un second chapitre, nous considérons une extension de la notion de signature : une 2-signature est une paire composée d'une signature (au sens du premier chapitre) et d'un ensemble d'équations. Un 2-modèle d'une 2-signature est alors un modèle de la signature sous-jacente qui vérifie toutes les équations requises. Nous montrons l'existence d'un 2-modèle initial pour la classe des 2-signatures algébriques (Théorème 83) : il s'agit de 2-signatures composées d'une signature algébrique au sens précédent, et d'un ensemble d'équations dites *élémentaires* (Définition 80). Nous ne savons pas si ce théorème permet de spécifier n'importe quelle syntaxe engendrée par une signature présentable. Néanmoins, nous avons été capable d'encoder tous les exemples de signature présentable que nous considérons dans le premier chapitre.

Chapter 2

Introduction

This introduction is largely inspired by [AHLM19a, AHLM19b].

2.1 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* [JGW78], and has been popularised by the movement of *Algebra of Programming* [BdM97].

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].

[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.

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.

Let us give a synopsis of this thesis before presenting related work.

2.2 Synopsis and published work

Chapter 3 is based on the following article: [AHLM19a] *High-level signatures and initial semantics*. The notion of signature and model that we work with is given there. The main result of this chapter is that "quotients" of "algebraic" signatures, that we call presentable signatures, have an initial model.

Then, in Chapter 4, we extend the notion of signature, so that equations can also be specified, to yield the notion of 2-signature. The main result of this chapter is that "algebraic 2-signatures" have an initial model. This is based on the following article: [AHLM19b] *Modular specification of monads through higher-order presentations* .

We suppose a certain familiarity with category theory and basic notions such as categories, functors, natural transformations, monads, limits and colimits.

2.3 Computer-checked formalization

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

Our computer-checked proof is based on the `UniMath` library [VAG$^+$], which itself is based on the proof assistant `Coq` [CoqDev]. 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 [AL17].

The formalization can be consulted on `https://github.com/UniMath/largecatmodules`. A guide is given in the README.

For the purpose of this thesis, we refer to a fixed version of our library, with the short hash 50fd617. This version compiles with version 10839ee of `UniMath`.

Throughout the thesis, statements are annotated with their corresponding identifiers in the formalization. These identifiers are also hyperlinks to the online documentation stored at `https://initialsemantics.github.io/doc/50fd617/index.html`.

2.4 Related work

2.5 Syntaxes and monads

The idea that the notion of monad is suited for modelling substitution concerning syntax (and semantics) has been retained by many contributions on the subject (see e.g. [BP99, GUH06, MU04, AR99]). In particular, Matthes and Uustalu [MU04] introduce a very general notion of signature and, subsequently, Ghani, Uustalu, and Hamana [GUH06] consider a form of colimits (namely coends) of such signatures. Their treatment rests on the technical device of *strength*[3], that we avoid here. Any signature with strength gives rise to a signature in our sense, cf. Proposition 21. Research on signatures with strength is actively developed, see also [AMM18] for a more recent account.

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

Fiore, Plotkin, and Turi [FPT99] develop a notion of substitution monoid. Following [ACU15], this setting can be rephrased in terms of relative monads and modules over them [Ahr16]. Accordingly, our present contributions could probably be customised for this "relative" approach.

The work by Fiore with collaborators [FPT99, FH10, FM10] and the work by Uustalu with collaborators [MU04, GUH06] share two traits: firstly,

[3] 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.

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 [GP99] 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 [HP07], 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 [Hir13].

2.6 Syntax with equations

Ahrens [Ahr16] introduces the notion of 2-signature which we consider here in Chapter 4, in the slightly different context of (relative) monads on preordered sets, where the preorder models the reduction relation. In some sense, Chapter 4 tackles the technical issue of quotienting the initial (relative) monad constructed in [Ahr16] by the preorder.

In a classical paper, Barr [Bar70] explained the construction of the "free monad" generated by an endofunctor[4]. In another classical paper, Kelly and Power [KP93] explained how any finitary monad can be presented as a coequalizer of free monads[5]. There, free monads correspond to our initial models of an algebraic 1-signature without any binding construction.

There also several contributions by Fiore and his collaborators that tackle the specification of syntaxes with equations:

- Our notion of equations and that of model for them seem very close to the notion of equational systems and that of algebra for them in [FH09]: in particular, the preservation of epimorphisms, which occurs in their construction of inductive free algebras for equational systems, appears here in our definition of elementary equation. It would be interesting to understand formal connections between the two approaches.

[4]Fiore and Saville [FS17] give an enlightening generalization of the construction by Barr.

[5]Their work has been applied to various more general contexts (e.g. [Sta13]).

- In [FH10], Fiore and Hur introduce a notion of equation based on syntax with *meta-variables*: essentially, a specific syntax, say, $T := T(M, X)$ considered there depends on two contexts: a meta-context M, and an object-context X. The terms of the actual syntax are then those terms $t \in T(\emptyset, X)$ in an empty meta-context. An equation for T is, simply speaking, a pair of terms in the same pair of contexts. Transferring an equation to any model of the underlying algebraic 1-signature is done by induction on the syntax with meta-variables. The authors show a monadicity theorem which straightforwardly implies an initiality result very similar to ours. That monadicity result is furthermore an instance of a more general theorem by Fiore and Mahmoud [FM10, Theorem 6.2].

- Translations between languages similar to the translation we present in Section 4.5 are also studied in [FM10]. Here again, it would be interesting to understand formal connections.

- At this stage, our work only concerns untyped syntax, but we anticipate it will generalize to the sorted setting as in [FH10] (see also the more general [FH13]).

Furthermore, Hamana [Ham03] proposes initial algebra semantics for "binding term rewriting systems", based on Fiore, Plotkin, and Turi's presheaf semantics of variable binding and Lüth and Ghani's monadic semantics of term rewriting systems [LG97].

The alternative *nominal* approach to binding syntax initiated by Gabbay and Pitts [GP99] has been actively studied[6]. We highlight some contributions:

- Clouston [Clo10] discusses signatures, structures (a.k.a. models), and equations over signatures in nominal style.

- Fernández and Gabbay [FG10] study signatures and equational theories as well as rewrite theories over signatures.

- Kurz and Petrisan [KP10] study closure properties of subcategories of algebras under quotients, subalgebras, and products. They characterize full subcategories closed under these operations as those that

[6]The approaches by Fiore and collaborators and Gabbay and Pitts [GP99] are nicely compared by Power [Pow07], who also comments on some generalization of the former approach.

are definable by equations. They also show that the signature of the lambda calculus is effective, and study the subcategory of algebras of that signature specified by the β- and η-equations.

Chapter 3

Presentable signatures

In the present chapter extracted from [AHLM19a], 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 [FPT99, HHP93, GP99], 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 chapter 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 3.6). Examples are given in Section 3.8.

Our notions of signature and syntax enjoy modularity in the sense introduced by [GUH06]: 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.

This work improves on a previous attempt [HM12] 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.

Organisation of the chapter

Section 3.1 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 3.2 and 3.3 respectively. In Section 3.4, we give our definition of a syntax, and we present our first main result, a modularity result about merging extensions of syntax. In Section 3.5, we show through examples how recursion can be recovered from initiality. Our notion of *presentation of a signature* appears in Section 3.6. There, we also state our second main result: presentable signatures generate a syntax. The proof of that result is given in Section 3.7. Finally, in Section 3.8, we give examples of presentable signatures.

3.1 Categories of modules over monads

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 definitions are generalizable. See [HM10] for a more extensive introduction on this topic.

3.1.1 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 equations hold:

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

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 $f : R \longrightarrow S$ between the underlying functors such that

$$f \circ \eta = \eta', \qquad f \circ \mu = \mu' \circ (f \cdot f) \ .$$

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 $\rho^M\colon M \cdot R \longrightarrow M$, called *module substitution*, which is compatible with the monad composition and identity:

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

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*[1] if it is compatible with module substitution on either side:

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

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 $\mathsf{Mod}(R)$.

Limits and colimits in the category of modules can be constructed pointwise:

Proposition 4 (`LModule_Colims_of_shape`, `LModule_Lims_of_shape`)**.** $\mathsf{Mod}(R)$ *is complete and cocomplete.*

[1] 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., [Bra14, 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.

3.1.2 The total category of modules

We already introduced the category $\mathsf{Mod}(R)$ of modules with fixed base R. It is 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 and M an S-module. The module substitution $M \cdot R \xrightarrow{Mf} M \cdot S \xrightarrow{\rho^M} M$ defines an R-module which is called *pullback* of M along f and noted f^*M.[2]

Definition 6 (The total module category)**.** We define the *total module category* $\int_R \mathsf{Mod}(R)$, or $\int \mathsf{Mod}$ for short, as follows[3]:

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

Proposition 7 (`cleaving_bmod`)**.** *The forgetful functor $\int \mathsf{Mod} \to \mathsf{Mon}$ is a Grothendieck fibration with fibre $\mathsf{Mod}(R)$ over a monad R. In particular, any monad morphism $f : R \longrightarrow S$ gives rise to a functor*

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

given on objects by Definition 5.

Proposition 8 (`pb_LModule_colim_iso`, `pb_LModule_lim_iso`)**.** *For any monad morphism $f : R \longrightarrow S$, the functor $f^* : \mathsf{Mod}(S) \longrightarrow \mathsf{Mod}(R)$ preserves limits and colimits.*

[2] The term "pullback" is standard in the terminology of Grothendieck fibrations (see Proposition 7).

[3] 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.

3.1.3 Derivation

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

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 $\rho^{M'} \colon M' \cdot R \longrightarrow M'$ defined as in the diagram

$$\begin{array}{ccc} M(R(X)+*) & \xrightarrow{\rho^{M'}_X} & M(X+*) \\ {\scriptstyle M(R(i_X)+\eta_{X+*}\circ \underline{*})}\downarrow & \nearrow_{\rho^M_{X+*}} & \\ M(R(X+*)) & & \end{array} \tag{3.1}$$

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

Derivation is a cartesian endofunctor on the category $\mathsf{Mod}(R)$ 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 nonnegative 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, $M^{()}$ is the final module $*$.

Definition 11. For every monad R and R-module M we have a natural *substitution morphism* $\sigma \colon M' \times R \longrightarrow M$ defined 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 12 (`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 13 (`deriv_adj`)**.** *The endofunctor of $\mathsf{Mod}(R)$ mapping M to the R-module $M \times R$ is left adjoint to the derivation endofunctor, the unit being the substitution morphism σ.*

3.2 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 $\int \mathsf{Mod}$ to the category Mon.

Now we give our basic examples of signatures.

Example 15.

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 19 for families that are not necessarily finitely indexed).

Elementary signatures are of a particularly simple shape:

Definition 16. For each sequence of nonnegative 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 17. The product of two elementary signatures is elementary.

Definition 18. A *morphism between two signatures* $\Sigma_1, \Sigma_2 \colon \mathsf{Mon} \longrightarrow \int \mathsf{Mod}$ is a natural transformation $m \colon \Sigma_1 \longrightarrow \Sigma_2$ which, post-composed with the projection $\int \mathsf{Mod} \longrightarrow \mathsf{Mon}$, becomes the identity. Signatures form a subcategory Sig of the category of functors from Mon to $\int \mathsf{Mod}$.

Limits and colimits of signatures can be easily constructed pointwise:

Proposition 19 (`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 20. An *algebraic signature* is a (possibly infinite) coproduct of elementary signatures.

These signatures are those which appear in [FPT99]. 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 [MU04] (even though they were not given an explicit name there). The relevant definitions regarding signatures with strength are summarized in [AMM18], to which we refer the interested reader.

We recall that a signature with strength [AMM18, 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 [AMM18, 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 $\mathsf{SigStrength}$ [AMM18].

Any signature with strength H gives rise to a signature $\tilde{H}$ [HM12, 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 21 (`sigWithStrength_to_sig_functor`)**.** *The maps sketched above yield a functor* $(\tilde{-}) : \mathsf{SigStrength} \longrightarrow \mathsf{Sig}$.

3.3 Categories of models

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

Definition 22 (Models and actions)**.** 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 a module morphism $r : \Sigma(R) \to R$, called *action of* Σ *in* R. In other words, a model of Σ is a monad R equipped with an action of Σ in R[4]. 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:

$$\begin{array}{ccc} \Sigma(R) & \xrightarrow{\quad r \quad} & R \\ {\scriptstyle \Sigma(m)}\downarrow & & \downarrow{\scriptstyle m} \\ m^*(\Sigma(S)) & \xrightarrow[m^*s]{} & m^*S \end{array}$$

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

Example 23. 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

[4]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 and unit of the monad. This morphism is sometimes called an action.

$\mathsf{abs}\colon \mathsf{LC}' \longrightarrow \mathsf{LC}$ 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 24. *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 formalisation, this category is recovered as the fiber category over Σ of the displayed category [AL17] 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`.

Definition 25 (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*[5] of (R, r) along f and denoted by $f^*(R, r)$.

3.4 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.

3.4.1 Representations of a signature

Definition 26. 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 *effective*[6].

In this work, we aim to identify signatures that are effective. This is not automatic: below, we give a signature that is not effective. Afterwards, we give suitable sufficient criteria for signatures to be effective.

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

[6]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.

Non-example 27. Let $\mathcal{P}$ denote the powerset functor and consider the signature $\mathcal{P} \cdot \Theta$ (see Example 15, 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 effective.

Instead of giving a direct proof of the fact that $\mathcal{P} \cdot \Theta$ is not effective, we deduce it as a simple consequence of a stronger result that we consider interesting in itself: an analogue of Lambek's Lemma, given in Lemma 30.

The following preparatory lemma explains how to construct new models of a signature Σ from old ones:

Lemma 28. *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*

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

  give the endofunctor $\Sigma(R) + \mathsf{Id}$ the structure of a monad.

- *Moreover, this monad can be given the following Σ-action:*

$$\Sigma\big(\Sigma(R) + \mathsf{Id}\big) \xrightarrow{\Sigma([r,\eta])} \Sigma(R) \cdot R \xrightarrow{\rho^{\Sigma(R)}} \Sigma(R) \longrightarrow \Sigma(R) + \mathsf{Id} \qquad (3.2)$$

- *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 (3.2) in the source and $r : \Sigma(R) \longrightarrow R$ in the target.*

Definition 29. Given a model M of Σ, we denote by $M^\sharp$ the Σ-model constructed in Lemma 28, and by $\epsilon_M : M^\sharp \longrightarrow M$ the morphism of models defined there.

Lemma 30 (`iso_mod_id_model`)**.** *If Σ is effective, then the morphism of Σ-models*

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

is an isomorphism.

We go back to considering the signature $\Sigma := \mathcal{P} \cdot \Theta$. Suppose that Σ is effective. From Lemma 30 it follows that $\mathcal{P}\hat{\Sigma}X + X \cong \hat{\Sigma}X$. In particular, we 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 effective signatures:

Theorem 31 (`algebraic_sig_effective`)**.** *Algebraic signatures are effective.*

This result is proved in a previous work [HM07, 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 [AMM18].

In Section 3.6, we show a more general effectiveness result: Theorem 35 states that *presentable* signatures, which form a superclass of algebraic signatures, are effective.

3.4.2 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 [HM12, Section 6], there without proof.

Suppose that we have a pushout square of effective signatures,

$$\begin{array}{ccc} \Sigma_0 & \longrightarrow & \Sigma_1 \\ \downarrow & & \downarrow \\ \Sigma_2 & \longrightarrow & \Sigma \end{array}$$

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,

$$\begin{array}{ccc} \hat{\Sigma}_0 & \longrightarrow & \hat{\Sigma}_1 \\ \downarrow & & \downarrow \\ \hat{\Sigma}_2 & \longrightarrow & \hat{\Sigma} \end{array}$$

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 $\int_\Sigma \mathsf{Mon}^\Sigma$, or $\int \mathsf{Mon}$ for short, of models:

- An object of $\int \mathsf{Mon}$ 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 $\int \mathsf{Mon}$ 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 $\int \mathsf{Mon}$ into a category.

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

Theorem 32 (`pushout_in_big_rep`)**.** *Modularity holds in $\int \mathsf{Mon}$, in the sense that given a pushout square of effective signatures as above, the associated square in $\int \mathsf{Mon}$ is a pushout again.*

The proof uses, in particular, the following fact:

Lemma 33 (`rep_cleaving`)**.** *The projection $\pi : \int \mathsf{Mon} \to \mathsf{Sig}$ is a Grothendieck fibration. In particular, given a morphism $f\colon \Upsilon \longrightarrow \Sigma$ of signatures, the pullback map defined in Definition 25 extends to a functor*

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

Note that Theorem 32 does *not* say that a pushout of effective signatures is effective again; it only tells us that if all of the signatures in a pushout square are effective, 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 effective signatures is effective again.

In Section 3.6 we study *presentable* signatures, which we show to be effective. The subcategory of presentable signatures is closed under colimits.

3.5 Recursion

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

3.5.1 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 [Gir87, 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 &, $\parr$, $\otimes$, $\oplus$, $\multimap$ 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 $\hat{\Sigma}_{LJ}$,

enjoying the expected equalities. 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 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_{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.

3.5.2 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 $\mathsf{Maybe}^{-1}_X(S) = S \cap X$. 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.

3.5.3 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 [JG07] 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

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

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

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

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 $s_X(t) = \iota_X(t, 0_X)$.

3.5.4 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

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

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 0 otherwise. 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, 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.

3.6 Presentations of signatures and syntaxes

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

Definition 34. Given a signature Σ, a *presentation*[7] *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*.

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

Remark. 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 effective:

Theorem 35 (`PresentableisEffective`)**.** *Any presentable signature is effective.*

The proof is discussed in Section 3.7.

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

Theorem 36 (`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 $p^* : \mathsf{Mon}^{\Sigma} \longrightarrow \mathsf{Mon}^{\Upsilon}$ has a left adjoint.*

[7] 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).

In the proof of Theorem 36, the axiom of choice is used to show that endofunctors on Set preserve epimorphisms.

Theorem 35 follows from Theorem 36 since the left adjoint $p^! : \mathsf{Mon}^{\Upsilon} \longrightarrow \mathsf{Mon}^{\Sigma}$ preserves colimits, in particular, initial objects. However, Theorem 35 is proved in Section 3.7 without appealing to 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.

Definition 37. We call a syntax *presentable* if it is generated by a presentable signature.

Next, we give important examples of presentable signatures:

Theorem 38. *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 (`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 39. *Any colimit of algebraic signatures is effective.*

3.7 Proof of Theorem 35

In this section, we prove Theorem 35. 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 35 rests on the more technical Lemma 44 below.

Proposition 40 (`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 ([ML98, Exercise III.4.4]) :

$$\begin{array}{ccc} a & \xrightarrow{f} & b \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle id} \\ b & \xrightarrow[id]{} & b \end{array}$$

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

$$\begin{array}{ccc} R \cdot R & \xrightarrow{\mu} & R \\ {\scriptstyle \pi\cdot\pi}\downarrow & & \downarrow{\scriptstyle \pi} \\ \overline{R} \cdot \overline{R} & \dashrightarrow^{\overline{\mu}} & \overline{R} \end{array}$$

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 41 (`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.*

Note that the axiom of choice implies that epimorphisms have a retraction, and thus that any endofunctor on Set preserves epimorphisms.

Definition 42. 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.

Example 43 (`BindingSigAreEpiSig`)**.** Any algebraic signature is an epi-signature.

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

Lemma 44 (`push_initiality`)**.** *Let Υ be effective, 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 effective.*

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 $F : \Upsilon \to \Sigma$.

Per Lemma 41, as $\hat{\Upsilon}$ 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 40. 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:

$$\begin{array}{ccc} \Upsilon(\hat{\Upsilon}) & \xrightarrow{m_{\hat{\Upsilon}}} & \hat{\Upsilon} \\ {\scriptstyle u}\downarrow & & \downarrow{\scriptstyle \pi} \\ \Sigma(\hat{\Upsilon}/F) & \xrightarrow[m_{\hat{\Upsilon}/F}]{} & \hat{\Upsilon}/F \end{array}$$

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 shows 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 45 (`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. 35. Let $p : \Upsilon \to \Sigma$ be a presentation of Σ. We need to construct a representation for Σ.

As the signature Υ is algebraic, it is effective (by Theorem 31) and is an epi-signature (by Example 43). We can thus instantiate Lemma 44 to see that Σ is effective, thanks to Lemma 45. □

3.8 Constructions 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 15). Thanks to Theorem 38, Item 2, direct sums (or, indeed, any colimit) of presentable signatures are presentable, hence effective by Theorem 35.

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.

3.8.1 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 46. 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 47. *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 $F \cdot \Theta \to G \cdot \Theta$. □

Proposition 48. *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 [AP04], since ω-accessible functors are exactly the finitary ones. □

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

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 47, the signature $\mathcal{S}_2 \cdot \Theta$ is presentable; it encodes a commutative binary operator.

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 47, 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.

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

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

By Proposition 47, this yields a presentable signature, which we call $\mathbb{N}\langle\Theta\rangle$.

The Differential Lambda Calculus (DLC) [ER03] 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 $\Theta \times \mathbb{N}\langle\Theta\rangle$.

3.8.2 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:

$$\begin{array}{c} \Theta^{(\sigma)} \\ \circlearrowright \\ \Theta^{(k)} \end{array}$$

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 $\mathrm{colim}_k\, \mathcal{S}^{(k)}\Theta$ which, as a colimit of presentable signatures, is presentable. (Theorem 38, Item 2).

Accordingly, we define a total closure on a monad R to be an action of the signature $\mathrm{colim}_k\, \mathcal{S}^{(k)}\Theta$ 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:

$$\begin{array}{ccc} R^{(k)} & \xrightarrow{R^{(i)}} & R^{(k')} \\ & \searrow^{e_k} & \downarrow{\scriptstyle e_{k'}} \\ & & R \end{array}$$

3.8.3 Example: Adding an explicit substitution

Explicit substitution was introduced by Abadi et al. [ACCL90] 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 [GUH06].

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

$$\mathsf{subst}_p : R^{(p)} \times R^p \longrightarrow R. \tag{3.3}$$

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 [GUH06], 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:

$$\begin{array}{ccc} R^{(p)} \times R^q & \xrightarrow{R^{(p)} \times R^u} & R^{(p)} \times R^p \\ \downarrow{\scriptstyle R^{(u)} \times R^q} & & \downarrow{\scriptstyle \mathsf{subst}_p} \\ R^{(q)} \times R^q & \xrightarrow[\mathsf{subst}_q]{} & R \end{array} \tag{3.4}$$

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 49. 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.

Ghani, Uustalu, and Hamana already studied this problem in [GUH06]. Our solution proposed here does not require the consideration of a *strength*.

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

$$\begin{array}{ccc} R' & \xrightarrow{(id_{R'}, f)} & R' \times R \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle \sigma} & \\ & R & \end{array}$$

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

Proposition 51. *Any fixed-point combinator in $\mathsf{LC}_{\beta\eta}$ comes from a unique fixed-point operator.*

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

$$\begin{aligned} \mathsf{app}(Y_f, t) &= \mathsf{app}(\mathsf{abs}\, f_1(\mathsf{app}(*, **)), t) && (3.5)\\ &= f_X(\mathsf{app}(t, **)) && (3.6)\\ &= \mathsf{app}(t, \mathsf{app}(Y_f, t)) && (3.7) \end{aligned}$$

where (3.6) comes from the definition of a fixed-point operator. Equality (3.7) follows from the equality $\mathsf{app}(Y_f, t) = f_X(\mathsf{app}(t, **))$, which is obtained by chaining the equalities from (3.5) to (3.6). This concludes the construction of the bijection. □

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

Definition 52.

- 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:

$$\begin{array}{ccc} (R^{(n)})^n & \xrightarrow{id_{(R^{(n)})^n}, f, .., f} & (R^{(n)})^n \times (R^n)^n \\ {\scriptstyle f}\downarrow & & \| \wr \\ R^n & \xleftarrow[(\mathsf{subst}_n)^n]{} & (R^{(n)} \times R^n)^n \end{array}$$

  where subst_n is the n-substitution as in Section 3.8.3.

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

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 3.8.3, which we now introduce.

[8] In the following, we similarly write n instead of I_n in order to make equations more readable.

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:

$$\begin{array}{ccc} p \times (R^{(p)})^q & \xrightarrow{p \times (R^{(p)})^u} & p \times (R^{(p)})^p \\ {\scriptstyle u \times (R^{(u)})^q}\downarrow & & \downarrow{\scriptstyle \mathsf{fix}_p} \\ q \times (R^{(q)})^q & \xrightarrow[\mathsf{fix}_q]{} & R \end{array} \tag{3.8}$$

These conditions have an interpretation in terms of a coend, just as we already encountered in Section 3.8.3. This leads us to the following

Definition 53. 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.

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.

3.9 Conclusions

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 effective.

Despite the fact that the constructions in Section 3.7 make heavy use of quotients, there is no need to appeal to the axiom of choice. While a previous version of our formalisation did use the axiom of choice to show that certain functors preserve epimorphisms, we managed subsequently to

[9] As in Section 3.8.3, the invariance follows from the coherence.

prove this without using the axiom of choice. This analysis, and subsequent reworking, of the proof was significantly helped by the formalisation.

One difference to other work on Initial Semantics, e.g., [MU04, GU03, Fio08, FM10], 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 21).

Chapter 4

Algebraic 2-signatures

This chapter is extracted from [AHLM19b].

Presentable signatures of Chapter 3 allow to specify syntaxes satisfying some equations by considering colimits of algebraic 1-signatures. However, it seems quite limited: for example, we don't know how to specify an associative operation through a presentable signature. This motivates the extension that we do in the present chapter: our signatures can now specify not only operations, but also equations among them. We refer to these enhanced signatures as 2-signatures, while signatures and models in the sense of Chapter 3 are now referred to as 1-signatures and 1-models.

In this chapter, we also adopt an alternative viewpoint on this work: our signatures give presentations of monads on the category of sets. In the following, we adopt this viewpoint and motivate our notion of 2-signature. We extend the notion of signature of the previous chapter in order to take into account more sophisticated equations in the syntax.

We identify the class of algebraic 2-signatures which generate a syntax. It is not clear if any syntax generated by a presentable signature can also be generated by an algebraic 2-signature, although we don't have any counter-example. Conversely, algebraic 2-signatures take into account operations that we don't know how to specify using a presentable signature, such as an associative operation.

4.1 Introduction

There is a well established theory of presentations of monads through generating (first-order) operations equipped with relations among the corresponding derived operations. Algebraic 1-signatures can be considered as generating monads by *binding operations*. Various algebraic structures generated by binding operations have been considered by many, going back at least to Fiore, Plotkin, and Turi [FPT99], Gabbay and Pitts [GP99], and Hofmann [Hof99].

If $p : \hat{\Sigma} \to R$ is a monad epimorphism, we understand that R is generated by a family of operations whose binding arities are given by Σ, subject to suitable identifications. In particular, for $\Sigma := ((0,0),(1))$, $\hat{\Sigma}$ may be understood as the monad LC of syntactic terms of the lambda calculus, and we have an obvious epimorphism $p : \hat{\Sigma} \to \mathsf{LC}_{\beta\eta}$, where $\mathsf{LC}_{\beta\eta}$ is the monad of lambda-terms modulo β and η. In order to manage such equalities, the approach in the first-order case suggests to identify p as the coequalizer of a double arrow from T to $\hat{\Sigma}$ where T is again a "free" monad. Let us see what comes out when we attempt to find such an encoding for the β-equality of the monad $\mathsf{LC}_{\beta\eta}$. It should say that for each set X, the following two maps from $\hat{\Sigma}(X + \{*\}) \times \hat{\Sigma}(X)$ to $\hat{\Sigma}(X)$,

- $(t, u) \mapsto \mathsf{app}(\mathsf{abs}(t), u)$
- $(t, u) \mapsto t[* \mapsto u]$

are equal. Here a problem occurs, namely that the above collections of maps, which can be understood as a morphism of functors, cannot be understood as a morphism of monads. Notably, they do not send variables to variables.

On the other hand, we observe that the members of our equations, which are not morphisms of monads, commute with substitution, and hence are more than morphisms of functors: indeed they are morphisms of *modules over* $\hat{\Sigma}$. Accordingly, a (second-order) presentation for a monad R could be a diagram

$$T \overset{f}{\rightrightarrows} \hat{\Sigma} \xrightarrow{p} R \tag{4.1}$$

where Σ is a binding signature, $\hat{\Sigma}$ is the associated free monad, T is a module over $\hat{\Sigma}$, f is a pair of morphisms of modules over $\hat{\Sigma}$, and p is a monad epimorphism. And now we are faced with the task of finding a condition

meaning something like "p is the coequalizer of f"[1]. To this end, we introduce the category Mon^{Σ} "of models of Σ", whose objects are monads "equipped with an action of Σ". Of course $\hat{\Sigma}$ is equipped with such an action which turns it into the initial object. Next, we define the full subcategory of models satisfying the equation f, and require R to be the initial object therein. Our definition is suited for the case where the equation f is parametric in the model: this means that now T and f are functions of the model S, and $f(S) = (u(S), v(S))$ is a pair of S-module morphisms from $T(S)$ to S. We say that S satisfies the equation f if $u(S) = v(S)$. Generalizing the case of one equation to the case of a family of equations yields the notion of 2-signature already introduced by Ahrens [Ahr16] in a slightly different context.

Now we are ready to formulate our main problem: given a 2-signature (Σ, E), where E is a family of parametric equations as above, does the subcategory of models of Σ satisfying the family of equations E admit an initial object?

We answer positively for a large subclass of 2-signatures which we call *algebraic* 2-signatures (see Theorem 83).

This provides a construction of a monad from an algebraic 2-signature, and we prove furthermore (see Theorem 78) that this construction is *modular*, in the sense that merging two extensions of 2-signatures corresponds to building an amalgamated sum of initial models. This is analogous to Theorem 32 for 1-signatures.

As expected, our initiality property generates a recursion principle which is a recipe allowing us to specify a morphism from the presented monad to any given other monad.

We give various examples of monads arising "in nature" that can be specified via an algebraic 2-signature (see Section 4.4), and we also show through a simple example how our recursion principle applies (see Section 4.5).

Computer-checked formalization A summary of our formalization regarding 2-signatures is available at `https://initialsemantics.github.io/doc/50fd617/Modules.SoftEquations.Summary.html`.

[1]This cannot be the case stricto sensu since f is a pair of module morphisms while p is a monad morphism.

4.2 2-Signatures and their models

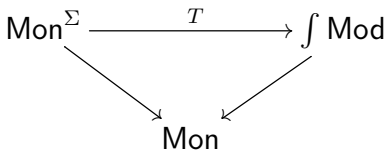
In this section we study *2-signatures* and *models of 2-signatures*. A 2-signature is a pair of a 1-signature and a family of *equations* over it.

4.2.1 Equations

Our equations are those of Ahrens [Ahr16]: they are parallel module morphisms parametrized by the models of the underlying 1-signature. The underlying notion of 1-model is essentially the same as in [Ahr16], even if, there, such equations are interpreted instead as *inequalities*.

Throughout this subsection, we fix a 1-signature Σ, that we instantiate in the examples.

Definition 54. We define a **Σ-module** to be a functor T from the category of models of Σ to the category $\int \mathsf{Mod}$ commuting with the forgetful functors to the category Mon of monads,



Example 55. To each 1-signature Ψ is associated, by precomposition with the projection from Mon^Σ to Mon, a Σ-module still denoted Ψ. All the Σ-modules occurring in this work arise in this way from 1-signatures; in other words, they do not depend on the action of the 1-model. In particular, we have the **tautological Σ-module** Θ, and, more generally, for any natural number $n \in \mathbb{N}$, a Σ-module $\Theta^{(n)}$. Also we have another fundamental Σ-module (arising in this way from) Σ itself.

Definition 56. Let S and T be Σ-modules. We define a **morphism of Σ-modules** from S to T to be a natural transformation from S to T which becomes the identity when postcomposed with the forgetful functor from the category of models of Σ to the category of monads.

Example 57. Each 1-signature morphism $\Psi \to \Phi$ upgrades into a morphism of Σ-modules. Further in that vein, there is a morphism of Σ-modules $\tau^\Sigma : \Sigma \to \Theta$. It is given, on a model (R, m) of Σ, by $m : \Sigma(R) \to R$. (Note that it does not arise from a morphism of 1-signatures.) When the context

is clear, we write simply τ for this morphism, and call it the **tautological morphism of Σ-modules**.

Proposition 58. *Our Σ-modules and their morphisms, with the obvious composition and identity, form a category.*

Definition 59. We define a Σ-equation to be a pair of parallel morphisms of Σ-modules. We also write $e_1 = e_2$ for the Σ-equation $e = (e_1, e_2)$.

Example 60 (Commutativity of a binary operation)**.** Here we instantiate our fixed 1-signature as follows: $\Sigma := \Theta \times \Theta$. In this case, we say that τ is the (tautological) binary operation. Now we can formulate the usual law of commutativity for this binary operation.

We consider the morphism of 1-signatures $\mathsf{swap} : \Theta^2 \longrightarrow \Theta^2$ that exchanges the two components of the direct product. Again by Example 57, we have an induced morphism of Σ-modules, still denoted swap.

Then, the Σ-equation for commutativity is given by the two morphisms of Σ-modules

$$\Theta^2 \xrightarrow{\mathsf{swap}} \Theta^2 \xrightarrow{\tau} \Theta$$
$$\Theta^2 \xrightarrow[\tau]{} \Theta$$

See also Section 4.4.1 where we explain in detail the case of monoids.

For the example of the lambda calculus with β- and η-equality (given in Example 62), we need to introduce *currying*:

Definition 61. By abstracting over the base monad R the adjunction in the category of R-modules of Proposition 13, we can perform **currying** of morphisms of 1-signatures: given a morphism of signatures $\Sigma_1 \times \Theta \to \Sigma_2$ it produces a new morphism $\Sigma_1 \to \Sigma_2'$. By Example 55, currying acts also on morphisms of Σ-modules.

Conversely, given a morphism of 1-signatures (resp. Σ-modules) $\Sigma_1 \to \Sigma_2'$, we can define the **uncurryied** map $\Sigma_1 \times \Theta \to \Sigma_2$.

Example 62 (β- and η-conversions)**.** Here we instantiate our fixed 1-signature as follows: $\Sigma_{\mathsf{LC}} := \Theta \times \Theta + \Theta'$. This is the 1-signature of the lambda calculus. We break the tautological Σ-module morphism into its two pieces, namely $\mathsf{app} := \tau \circ \mathsf{inl} : \Theta \times \Theta \longrightarrow \Theta$ and $\mathsf{abs} := \tau \circ \mathsf{inr} : \Theta' \longrightarrow \Theta$. Applying currying to app yields the morphism $\mathsf{app}_1 : \Theta \longrightarrow \Theta'$ of Σ_{LC}-modules. The usual β

and η relations are implemented in our formalism by two Σ_{LC}-equations that we call e_β and e_η respectively:

$$e_\beta : \begin{array}{c} \Theta' \xrightarrow{\mathsf{abs}} \Theta \xrightarrow{\mathsf{app}_1} \Theta' \\ \Theta' \xrightarrow[1]{\qquad\qquad} \Theta' \end{array} \qquad \text{and} \qquad e_\eta : \begin{array}{c} \Theta \xrightarrow{\mathsf{app}_1} \Theta' \xrightarrow{\mathsf{abs}} \Theta \\ \Theta \xrightarrow[1]{\qquad\qquad} \Theta \end{array}$$

4.2.2 2-signatures and their models

Definition 63. A **2-signature** is a pair (Σ, E) of a 1-signature Σ and a family E of Σ-equations.

Example 64. The 2-signature for a commutative binary operation is $(\Theta^2, \tau \circ \mathsf{swap} = \tau)$ (cf. Example 60).

Example 65. The 2-signature of the lambda calculus modulo β- and η-equality is $\Upsilon_{\mathsf{LC}_{\beta\eta}} = (\Theta \times \Theta + \Theta', \{e_\beta, e_\eta\})$, where e_β, e_η are the Σ_{LC}-equations defined in Example 62.

Definition 66 (`satisfies_equation`)**.** We say that a model M of Σ **satisfies the Σ-equation** $e = (e_1, e_2)$ if $e_1(M) = e_2(M)$. If E is a family of Σ-equations, we say that a model M of Σ **satisfies** E if M satisfies each Σ-equation in E.

Definition 67. Given a monad R and a 2-signature $\Upsilon = (\Sigma, E)$, an **action of Υ in** R is an action of Σ in R such that the induced 1-model satisfies all the equations in E.

Definition 68 (`category_model_equations`)**.** For a 2-signature (Σ, E), we define the **category $\mathsf{Mon}^{(\Sigma,E)}$ of models of** (Σ, E) to be the full subcategory of the category of models of Σ whose objects are models of Σ satisfying E, or equivalently, monads equipped with an action of (Σ, E).

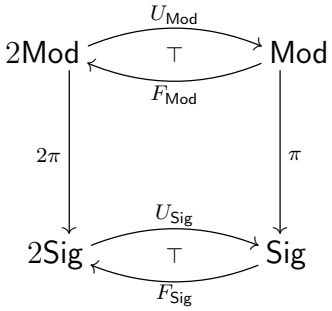
Example 69. A model of the 2-signature $\Upsilon_{\mathsf{LC}_{\beta\eta}} = (\Theta \times \Theta + \Theta', \{e_\beta, e_\eta\})$ is given by a model $(R, \mathsf{app}^R : R \times R \to R, \mathsf{abs}^R : R' \to R)$ of the 1-signature Σ_{LC} such that $\mathsf{app}_1^R \cdot \mathsf{abs}^R = 1_{R'}$ and $\mathsf{abs}^R \cdot \mathsf{app}_1^R = 1_R$ (see Example 62).

Definition 70. A 2-signature (Σ, E) is said to be **effective** if its category of models $\mathsf{Mon}^{(\Sigma,E)}$ has an initial object, denoted $\widehat{(\Sigma, E)}$.

In Section 4.2.4, we aim to find sufficient conditions for a 2-signature (Σ, E) to be effective.

4.2.3 Modularity for 2-signatures

In this section, we define the category 2Sig of 2-signatures and the category 2Mod of models of 2-signatures, together with functors that relate them with the categories of 1-signatures and 1-models. The situation is summarized in the commutative diagram of functors



where

- 2π is a Grothendieck fibration;
- π is the Grothendieck fibration defined in Section 3.4.2;
- U_{Sig} is a coreflection and preserves colimits; and
- U_{Mod} is a coreflection.

As a simple consequence of this data, we obtain, in Theorem 78, a *modularity* result in the sense of Ghani, Uustalu, and Hamana [GUH06]: it explains how the initial model of an amalgamated sum of 2-signatures is the amalgamation of the initial model of the summands[2].

We start by defining the category 2Sig of 2-signatures:

Definition 71 (`TwoSig_category`)**.** Given 2-signatures (Σ_1, E_1) and (Σ_2, E_2), a **morphism of 2-signatures from** (Σ_1, E_1) **to** (Σ_2, E_2) is a morphism of 1-signatures $m : \Sigma_1 \to \Sigma_2$ such that for any model M of Σ_2 satisfying E_2, the Σ_1-model m^*M satisfies E_1.

These morphisms, together with composition and identity inherited from 1-signatures, form the category 2Sig.

[2]This definition of "modularity" does not seem related to the specific meaning it has in the rewriting community (see, for example, [Gra12]).

We now study the existence of colimits in 2Sig. We know that Sig is cocomplete, and we use this knowledge in our study of 2Sig, by relating the two categories:

Let $F_{\mathsf{Sig}} : \mathsf{Sig} \to 2\mathsf{Sig}$ be the functor which associates to any 1-signature Σ the empty family of equations, $F_{\mathsf{Sig}}(\Sigma) := (\Sigma, \emptyset)$. Call $U_{\mathsf{Sig}} : 2\mathsf{Sig} \to \mathsf{Sig}$ the forgetful functor defined on objects as $U_{\mathsf{Sig}}(\Sigma, E) := \Sigma$.

Lemma 72 (`TwoSig_OneSig_is_right_adjoint`, `OneSig_TwoSig_fully_faithful`)**.** *We have $F_{\mathsf{Sig}} \dashv U_{\mathsf{Sig}}$. Furthermore, U_{Sig} is a coreflection.*

We are interested in specifying new languages by "gluing together" simpler ones. On the level of 2-signatures, this is done by taking the coproduct, or, more generally, the pushout of 2-signatures:

Theorem 73 (`TwoSig_PushoutsSET`)**.** *The category* 2Sig *has pushouts.*

Coproducts are computed by taking the union of the equations and the coproducts of the underlying 1-signatures. Coequalizers are computed by keeping the equations of the codomain and taking the coequalizer of the underlying 1-signatures. Thus, by decomposing any colimit into coequalizers and coproducts, we have this more general result:

Proposition 74. *The category* 2Sig *is cocomplete and U_{Sig} preserves colimits.*

We now turn to our modularity result, which states that the initial model of a coproduct of two 2-signatures is the coproduct of the initial models of each 2-signature. More generally, the two languages can be amalgamated along a common "core language", by considering a pushout rather than a coproduct.

For a precise statement of that result, we define a "total category of models of 2-signatures":

Definition 75. The category $\int_{(\Sigma,E)} \mathsf{Mon}^{(\Sigma,E)}$, or 2Mod for short, has, as objects, pairs $((\Sigma, E), M)$ of a 2-signature (Σ, E) and a model M of (Σ, E).

A morphism from $((\Sigma_1, E_1), M_1)$ to $((\Sigma_2, E_2), M_2)$ is a pair (m, f) consisting of a morphism $m : (\Sigma_1, E_1) \to (\Sigma_2, E_2)$ of 2-signatures and a morphism $f : M_1 \to m^* M_2$ of (Σ_1, E_1)-models (or, equivalently, of Σ_1-models).

This category of models of 2-signatures contains the models of 1-signatures as a coreflective subcategory. Let $F_{\mathsf{Mod}} : \mathsf{Mod} \to 2\mathsf{Mod}$ be the functor which associates to any 1-model (Σ, M) the empty family of equations, $F_{\mathsf{Mod}}(\Sigma, M) := (F_{\mathsf{Sig}}(\Sigma), M)$. Conversely, the forgetful functor $U_{\mathsf{Mod}} : 2\mathsf{Mod} \to \mathsf{Mod}$ maps $((\Sigma, E), M)$ to (Σ, M).

Lemma 76 (`TwoMod_OneMod_is_right_adjoint`, `OneMod_TwoMod_fully_faithful`)**.** *We have* $F_{\mathsf{Mod}} \dashv U_{\mathsf{Mod}}$*. Furthermore,* U_{Mod} *is a coreflection.*

The modularity result is a consequence of the following technical result:

Proposition 77 (`TwoMod_cleaving`)**.** *The forgetful functor* $2\pi : 2\mathsf{Mod} \to 2\mathsf{Sig}$ *is a Grothendieck fibration.*

The *modularity result* below is analogous to the modularity result for 1-signatures (Theorem 32):

Theorem 78 (Modularity for 2-signatures, `pushout_in_big_rep`)**.** *Suppose we have a pushout diagram of effective 2-signatures, as on the left below. This pushout gives rise to a commutative square of morphisms of models in* 2Mod *as on the right below, where we only write the second components, omitting the (morphisms of) signatures. This square is a pushout square.*

$$\begin{array}{ccc} \Upsilon_0 & \longrightarrow & \Upsilon_1 \\ \downarrow & & \downarrow \\ \Upsilon_2 & \longrightarrow & \Upsilon \end{array} \qquad \begin{array}{ccc} \widehat{\Upsilon}_0 & \longrightarrow & \widehat{\Upsilon}_1 \\ \downarrow & & \downarrow \\ \widehat{\Upsilon}_2 & \longrightarrow & \widehat{\Upsilon} \end{array}$$

Intuitively, the 2-signatures Υ_1 and Υ_2 specify two extensions of the 2-signature Υ_0, and Υ is the smallest extension containing both these extensions. By Theorem 78 the initial model of Υ is the "smallest model containing both the languages generated by Υ_1 and Υ_2".

4.2.4 Initial Semantics for 2-Signatures

We now turn to the problem of constructing the initial model of a 2-signature (Σ, E). More specifically, we identify sufficient conditions for (Σ, E) to admit an initial object $\widehat{(\Sigma, E)}$ in the category of models. Our approach is very straightforward: we seek to construct $\widehat{(\Sigma, E)}$ by applying a suitable quotient construction to the initial object $\hat{\Sigma}$ of Mon^{Σ}.

This leads immediately to our first requirement on (Σ, E), which is that Σ must be an effective 1-signature. (For instance, we can assume that Σ is an algebraic 1-signature, see Theorem 31.) This is a very natural hypothesis, since in the case where E is the empty family of Σ-equations, it is obviously a necessary and sufficient condition.

Some Σ-equations are never satisfied. In that case, the category $\mathsf{Mon}^{(\Sigma,E)}$ is empty. For example, given any 1-signature Σ, consider the Σ-equation $\mathsf{inl}, \mathsf{inr} : \Theta \rightrightarrows \Theta + \Theta$ given by the left and right inclusion. This is obviously an unsatisfiable Σ-equation. We have to find suitable hypotheses to rule out such unsatisfiable Σ-equations. This motivates the notion of *elementary* equations.

Definition 79. Given a 1-signature Σ, a Σ-module S is **nice** if S sends pointwise epimorphic Σ-model morphisms to pointwise epimorphic module morphisms.

Definition 80 (`elementary_equation`)**.** Given a 1-signature Σ, an **elementary Σ-equation** is a Σ-equation such that

- the target is a finite derivative of the tautological 2-signature Θ, i.e., of the form $\Theta^{(n)}$ for some $n \in \mathbb{N}$, and
- the source is a nice Σ-module.

Example 81. Any algebraic 1-signature is nice (Example 43). Thus, any Σ-equation between an algebraic 1-signature and $\Theta^{(n)}$, for some natural number n, is elementary.

Definition 82. A 2-signature (Σ, E) is said **algebraic** if Σ is algebraic and E is a family of elementary equations.

Theorem 83 (`elementary_equations_on_alg_preserve_initiality`)**.** *Any algebraic 2-signature has an initial model.*

The proof of Theorem 83 is given in Section 4.3.

Example 84. The 2-signature of lambda calculus modulo β and η equations given in Example 65 is algebraic. Its initial model is precisely the monad $\mathsf{LC}_{\beta\eta}$ of lambda calculus modulo $\beta\eta$ equations.

The instantiation of the formalized Theorem 83 to this 2-signature is done in `LCBetaEta`[3].

Let us mention finally that, using the axiom of choice, we can take a similar quotient on all the 1-models of Σ:

[3]An initiality result for this particular case was also previously discussed and proved formally in the Coq proof assistant in [HM10].

Proposition 85 (`ModEq_Mod_is_right_adjoint`, `ModEq_Mod_fully_faithful`)**.** *Here we assume the axiom of choice. The forgetful functor from the category* $\mathsf{Mon}^{(\Sigma,E)}$ *of 2-models of* (Σ, E) *to the category* Mon^{Σ} *of* Σ*-models has a left adjoint. Moreover, the left adjoint is a reflector.*

4.3 Proof of Theorem 83

Our main technical result on effectiveness is the following Lemma 86. In Theorem 83, we give a much simpler criterion that encompasses all the examples we give.

Lemma 86 (`elementary_equations_preserve_initiality`)**.** *Let* (Σ, E) *be a 2-signature such that:*

1. Σ *sends epimorphic natural transformations to epimorphic natural transformations,*
2. E *is a family of elementary equations,*
3. *the initial 1-model of* Σ *exists,*
4. *the initial 1-model of* Σ *preserves epimorphisms,*
5. *the image by* Σ *of the initial 1-model of* Σ *preserves epimorphisms.*

Then, the category of 2-models of (Σ, E) *has an initial object.*

Before tackling the proof of Lemma 86, we discuss how to derive Theorem 83 from it, and we prove some auxiliary results.

The "epimorphism" hypotheses of Lemma 86 are used to transfer structure from the initial model $\hat{\Sigma}$ of the 1-signature Σ onto a suitable quotient. There are different ways to prove these hypotheses:

- The axiom of choice implies conditions 4 and 5 since, in this case, any epimorphism in Set is split and thus preserved by any functor.
- Condition 5 is a consequence of condition 4 if Σ sends monads preserving epimorphisms to modules preserving epimorphisms.
- If Σ is algebraic, then conditions 1, 3, 4 and 5 are satisfied (Example 43 and Lemma 44).

From the remarks above, we derive the simpler and weaker statement of Theorem 83 that covers all our examples, which are algebraic.

This section is dedicated to the proof of the main technical result, Lemma 86. The reader inclined to do so may safely skip this section, and rely on the correctness of the machine-checked proof instead.

The proof of Lemma 86 uses some quotient constructions that we present now:

Proposition 87 (`u_monad_def`)**.** *Given a monad R preserving epimorphisms and a collection of monad morphisms $(f_i : R \to S_i)_{i\in I}$, there exists a* quotient *monad $R/(f_i)$ together with a* projection $p^R\colon R \longrightarrow R/(f_i)$*, which is a morphism of monads such that each f_i factors through p.*

Proof. The set $R/(f_i)(X)$ is computed as the quotient of $R(X)$ with respect to the relation $x \sim y$ if and only if $f_i(x) = f_i(y)$ for each $i \in I$. This is a straightforward adaptation of Lemma 41. □

Note that the epimorphism preservation is implied by the axiom of choice, but can be proven for the monad underlying the initial model $\hat{\Sigma}$ of an algebraic 1-signature Σ even without resorting to the axiom of choice.

The above construction can be transported on Σ-models:

Proposition 88 (`u_rep_def`)**.** *Let Σ be a 1-signature sending epimorphic natural transformations to epimorphic natural transformations, and let R be a Σ-model such that R and $\Sigma(R)$ preserve epimorphisms. Let $(f_i : R \to S_i)_{i\in I}$ be a collection of Σ-model morphisms. Then the monad $R/(f_i)$ has a natural structure of Σ-model and the quotient map $p^R\colon R \longrightarrow R/(f_i)$ is a morphism of Σ-models. Any morphism f_i factors through p^R in the category of Σ-models.*

The fact that R and $\Sigma(R)$ preserve epimorphisms is implied by the axiom of choice. The proof follows the same line of reasoning as the proof of Proposition 87.

Now we are ready to prove the main technical lemma:

Proof of Lemma 86. Let Σ be an effective 1-signature, and let E be a set of elementary Σ-equations. The plan of the proof is as follows:

1. Start with the initial model $(\hat{\Sigma}, \sigma)$, with $\sigma : \Sigma(\hat{\Sigma}) \to \hat{\Sigma}$.

2. Construct the quotient model $\hat{\Sigma}/(f_i)$ according to Proposition 88 where $(f_i : \hat{\Sigma} \to S_i)_i$ is the collection of all initial Σ-morphisms from $\hat{\Sigma}$ to any Σ-model satisfying the equations. We denote by $\sigma/(f_i) : \Sigma(\hat{\Sigma}/(f_i)) \to \hat{\Sigma}/(f_i)$ the action of the quotient model.

3. Given a model M of the 2-signature (Σ, E), we obtain a morphism $i_M : \hat{\Sigma}/(f_i) \to M$ from Proposition 88. Uniqueness of i_M is shown using epimorphicity of the projection $p : \hat{\Sigma} \to \hat{\Sigma}/(f_i)$. For this, it suffices to show uniqueness of the composition $i_M \circ p : \hat{\Sigma} \to M$ in the category of 1-models of Σ, which follows from initiality of $\hat{\Sigma}$.

4. The verification that $\big(\hat{\Sigma}/(f_i), \sigma/(f_i)\big)$ satisfies the equations is given below. Actually, it follows the same line of reasoning as in the proof of Proposition 87 that $\hat{\Sigma}/(f_i)$ satisfies the monad equations.

Let $e = (e_1, e_2) : U \to \Theta^{(n)}$ be an elementary equation of E. We want to prove that the two arrows

$$e_{1,\hat{\Sigma}/(f_i)}, e_{2,\hat{\Sigma}/(f_i)} \colon U(\hat{\Sigma}/(f_i)) \longrightarrow (\hat{\Sigma}/(f_i))^{(n)}$$

are equal. As p is an epimorphic natural transformation, $U(p)$ also is by definition of an elementary equation. It is thus sufficient to prove that

$$e_{1,\hat{\Sigma}/(f_i)} \circ U(p) = e_{2,\hat{\Sigma}/(f_i)} \circ U(p) \ ,$$

which, by naturality of e_1 and e_2, is equivalent to $p^{(n)} \circ e_{1,\hat{\Sigma}} = p^{(n)} \circ e_{2,\hat{\Sigma}}$.

Let x be an element of $U(\hat{\Sigma})$ and let us show that $p^{(n)}(e_{1,\hat{\Sigma}}(x)) = p^{(n)}(e_{2,\hat{\Sigma}}(x))$. By definition of $\hat{\Sigma}/(f_i)$ as a pointwise quotient (see Proposition 87), it is enough to show that for any j, the equality $f_j^{(n)}(e_{1,\hat{\Sigma}}(x)) = f_j^{(n)}(e_{2,\hat{\Sigma}}(x))$ is satisfied. Now, by naturality of e_1 and e_2, this equation is equivalent to $e_{1,S_j}(U(f_j)(x))) = e_{2,S_j}(U(f_j)(x)))$ which is true since S_j satisfies the equation $e_1 = e_2$. □

4.4 Examples of algebraic 2-signatures

We already illustrated our theory by looking at the paradigmatic case of lambda calculus modulo β- and η-equations (Examples 62 and 84). This section collects further examples of application of our results.

In our framework, complex signatures can be built out of simpler ones by taking their coproducts. Note that the class of algebraic 2-signatures encompasses the algebraic 1-signatures and is closed under arbitrary coproducts: the prototypical examples of algebraic 2-signatures given in this section can be combined with any other algebraic 2-signature, yielding an effective 2-signature thanks to Theorem 83.

4.4.1 Monoids

We begin with an example of monad for a first-order syntax with equations. Given a set X, we denote by $M(X)$ the free monoid built over X. This is a classical example of monad over the category of (small) sets. The monoid structure gives us, for each set X, two maps $m_X \colon M(X) \times M(X) \longrightarrow M(X)$ and $e_X \colon 1 \longrightarrow M(X)$ given by the product and the identity respectively. It can be easily verified that $m \colon M^2 \longrightarrow M$ and $e \colon 1 \longrightarrow M$ are M-module morphisms. In other words, $(M, \rho) = (M, [m, e])$ is a model of the 1-signature $\Sigma = \Theta \times \Theta + 1$.

We break the tautological morphism of Σ-modules (cf. Example 57) into constituent pieces, defining $\mathsf{m} := \tau \circ \mathsf{inl} : \Theta \times \Theta \to \Theta$ and $\mathsf{e} := \tau \circ \mathsf{inr} : 1 \to \Theta$.

Over the 1-signature Σ we specify equations postulating *associativity* and *left and right unitality* as follows:

$$\begin{array}{ccc}
\Theta^3 \xrightarrow{\Theta\times\mathsf{m}} \Theta^2 \xrightarrow{\mathsf{m}} \Theta & \qquad \Theta \xrightarrow{\mathsf{e}\times\Theta} \Theta^2 \xrightarrow{\mathsf{m}} \Theta & \qquad \Theta \xrightarrow{\Theta\times\mathsf{e}} \Theta^2 \xrightarrow{\mathsf{m}} \Theta \\
\Theta^3 \xrightarrow[\mathsf{m}\times\Theta]{} \Theta^2 \xrightarrow[\mathsf{m}]{} \Theta & \qquad \Theta \xrightarrow[1]{} \Theta & \qquad \Theta \xrightarrow[1]{} \Theta
\end{array}$$

and we denote by E the family consisting of these three Σ-equations. All are elementary since their codomain is Θ, and their domain a product of Θs.

One checks easily that $(M, [m, e])$ is the initial model of (Σ, E).

Several other classical (equational) algebraic theories, such as groups and rings, can be treated similarly, see Section 4.4.3 below. However, at the present state we cannot model theories with partial construction (e.g., fields).

4.4.2 Colimits of algebraic 2-signatures

In this section, we argue that our framework encompasses any colimit of algebraic 2-signatures.

Actually, the class of algebraic 2-signatures is not stable under colimits, as this is not even the case for algebraic 1-signatures. However, we can weaken this statement as follows:

Proposition 89. *Given any colimit of algebraic 2-signatures, there is an algebraic 2-signature yielding an isomorphic category of models.*

Proof. As the class of algebraic 2-signatures is closed under arbitrary coproducts, using the decomposition of colimits into coproducts and coequalizers, any colimit Ξ of algebraic 2-signatures can be expressed as a coequalizer of two morphisms f, g between some algebraic 2-signatures (Σ_1, E_1) and (Σ_2, E_2),

$$(\Sigma_1, E_1) \underset{g}{\overset{f}{\rightrightarrows}} (\Sigma_2, E_2) \xrightarrow{p} \Xi = (\Sigma_3, E_2) \ .$$

where Σ_3 is the coequalizer of the 1-signatures morphisms f and g. Note that the set of equations of Ξ is E_2, by definition of the coequalizer in the category of 2-signatures. Now, consider the algebraic 2-signature $\Xi' = (\Sigma_2, E_2 + (4.2))$ consisting of the 1-signature Σ_2 and the equations of E_2 plus the following elementary equation (see Example 81):

$$\begin{aligned} \Sigma_1 &\xrightarrow{f} \Sigma_2 \xrightarrow{\tau^{\Sigma_2}} \Theta \\ \Sigma_1 &\xrightarrow[g]{} \Sigma_2 \xrightarrow[\tau^{\Sigma_2}]{} \Theta \end{aligned} \tag{4.2}$$

We show that Mon^{Ξ} and $\mathsf{Mon}^{\Xi'}$ are isomorphic. A model of Ξ' is a monad R together with an R-module morphism $r : \Sigma_2(R) \to R$ such that $r \circ f_R = r \circ g_R$ and that the equations of E_2 are satisfied. By universal property of the coequalizer, this is exactly the same as giving an R-module morphism $\Sigma_3(R) \to R$ satisfying the equations of E_2, i.e., giving R an action of $\Xi = (\Sigma_3, E_2)$.

It is straightforward to check that this correspondence yields an isomorphism between the category of models of Ξ and the category of models of Ξ'. □

This proposition, together with the following corollary, allow us to recover all the examples presented in Chapter chap:csl, as colimits of algebraic 1-signatures: syntactic commutative binary operator, maximum operator, application à la differential lambda calculus, syntactic closure operator, integrated substitution operator, coherent fixpoint operator.

Corollary 90. *If F is a finitary endofunctor on* Set*, then there is an algebraic 2-signature whose category of models is isomorphic to the category of 1-models of the 1-signature $F \cdot \Theta$.*

Proof. It is enough to prove that $F \cdot \Theta$ is a colimit of algebraic 1-signatures.

As F is finitary, it is isomorphic to the coend $\int^{n \in \mathbb{N}} F(n) \times \_^n$ where $\mathbb{N}$ is the full subcategory of Set of finite ordinals (see, e.g., [VK11, Example 3.19]). As colimits are computed pointwise, the 1-signature $F \cdot \Theta$ is the coend $\int^{n \in \mathbb{N}} F(n) \times \Theta^n$, and as such, it is a colimit of algebraic 2-signatures. □

However, we do not know whether we can recover Theorem 35 stating that any presentable 1-signature is effective.

4.4.3 Algebraic theories

From the categorical point of view, several fundamental algebraic structures in mathematics can be conveniently and elegantly described using finitary monads. For instance, the category of monoids can be seen as the category of Eilenberg–Moore algebras of the monad of lists. Other important examples, like groups and rings, can be treated analogously. A classical reference on the subject is the work of Manes, where such monads are significantly called *finitary algebraic theories* [Man76, Def. 3.17].

We want to show that such "algebraic theories" fit in our framework, in the sense that they can be incorporated into an algebraic 2-signature, with the effect of enriching the initial model with the operations of the algebraic theory, subject to the axioms of the algebraic theory.

For a finitary monad T, Corollary 90 says how to encode the 1-signature $T \cdot \Theta$ as an algebraic 2-signature (Σ_T, E_T). Models are monads R together with an R-linear morphism $r : T \cdot R \to R$.

Now, for any model (R, m) of $T \cdot \Theta$, we would like to enforce the usual T-algebra equations on the action m. This is done thanks to the following equations, where τ denotes the tautological morphism of $T \cdot \Theta$-modules:

$$\begin{array}{ll} \Theta \xrightarrow{\eta_T \cdot \Theta} T \cdot \Theta \xrightarrow{\tau} \Theta & T \cdot T \cdot \Theta \xrightarrow{\mu_T \cdot \Theta} T \cdot \Theta \xrightarrow{\tau} \Theta \\ \Theta \xrightarrow[1]{} \Theta & T \cdot T \cdot \Theta \xrightarrow[T\tau]{} T \cdot \Theta \xrightarrow{\tau} \Theta \end{array} \tag{4.3}$$

The first equation is clearly elementary. The second one is elementary thanks to the following lemma:

Lemma 91. *Let F be a finitary endofunctor on* Set*. Then F preserves epimorphisms.*

Proof. This is a consequence of the axiom of choice, because then any epimorphism in the category of Set is split, and thus preserved by any functor. Here we provide an alternative proof which does not rely on the axiom of choice. (However, it may require the excluded middle, depending on the chosen definition of finitary functor.)

As F is finitary, it is isomorphic to the coend $\int^{n\in\mathbb{N}} F(n) \times \_^n$ [VK11, Example 3.19]. By decomposing it as a coequalizer of coproducts, we get an epimorphism $\alpha : \coprod_{n\in\mathbb{N}} F(n) \times \_^n \to F$. Now, let $f : X \to Y$ be a surjective function between two sets. We show that $F(f)$ is epimorphic. By naturality, the following diagram commutes:

$$\begin{array}{ccc} \coprod_{n\in\mathbb{N}} F(n) \times X^n & \xrightarrow{F(n)\times f^n} & \coprod_{n\in\mathbb{N}} F(n) \times Y^n \\ \downarrow{\scriptstyle \alpha_X} & & \downarrow{\scriptstyle \alpha_Y} \\ F(X) & \xrightarrow[F(f)]{} & F(Y) \end{array}$$

The composition along the top-right is epimorphic by composition of epimorphisms. Thus, the bottom left is also epimorphic, and so is $F(f)$ as the last morphism of this composition.

□

In conclusion, we have exhibited the algebraic 2-signature (Σ_T, E'_T), where E'_T extends the family E_T with the two elementary equations of Diagram 4.3. This signature allows to enrich any other algebraic 2-signature with the operations of the algebraic theory T, subject to the relevant equations.

4.4.4 Fixpoint operator

Here, we show the algebraic 2-signature corresponding to a fixpoint operator. In Section 3.8.4, we studied fixpoint operators in the context of 1-signatures. In that setting, we treated a *syntactic* fixpoint operator called *coherent* fixpoint operator, somehow reminiscent of mutual letrec. We were able to impose many natural equations to this operator but we were not able to enforce the fixpoint equation. In this section, we show how a fixpoint operator can be fully specified by an algebraic 2-signature. We restrict our discussion to

the unary case; the coherent family of multi-ary fixpoint operators presented in Section 3.8.4, now including the fixpoint equations, can also be specified, in an analogous way, via an algebraic 2-signature.

Let us start by recalling Definition 50: a **unary fixpoint operator for a monad** R is a module morphism f from R' to R that makes the following diagram commute, where σ is the substitution morphism defined as the uncurrying (see Definition 61) of the identity morphism on Θ':

$$\begin{array}{ccc} R' & \xrightarrow{(id_{R'},f)} & R' \times R \\ & {\scriptstyle f}\searrow \quad \swarrow{\scriptstyle \sigma_R} & \\ & R & \end{array}$$

In order to rephrase this definition, we introduce the obviously algebraic 2-signature Υ_{fix} consisting of the 1-signature $\Sigma_{\mathsf{fix}} = \Theta'$ and the family E_{fix} consisting of the single following Σ_{fix}-equation:

$$e_{\mathsf{fix}} : \begin{array}{l} \Theta' \xrightarrow{\langle 1,\tau\rangle} \Theta' \times \Theta \xrightarrow{\sigma} \Theta \\ \Theta' \xrightarrow[\tau]{} \Theta \end{array} \tag{4.4}$$

This allows us to rephrase the previous definition as follows: a unary fixpoint operator for a monad R is just an action of the 2-signature Υ_{fix} in R.

4.5 Recursion

In this section, we explain how a recursion principle can be derived from our initiality result, and give an example of a morphism—a *translation*—between monads defined via the recursion principle.

4.5.1 Principle of recursion

In our context, the recursion principle is a recipe for constructing a morphism from the monad underlying the initial model of a 2-signature to an arbitrary monad.

Proposition 92 (Recursion principle)**.** *Let S be the monad underlying the initial model of the 2-signature Υ. To any action a of Υ in T is associated a monad morphism $\hat{a} : S \to T$.*

Proof. The action a defines a 2-model M of Υ, and $\hat{a}$ is the monad morphism underlying the initial morphism to M. □

Hence the recipe consists in the following two steps:

1. give T an action of the 1-signature Σ;
2. check that all the equations in E are satisfied for the induced model.

In the next section, we illustrate this principle.

4.5.2 Translation of lambda calculus with fixpoint to lambda calculus

In this section, we consider the 2-signature $\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}} := \Upsilon_{\mathsf{LC}_{\beta\eta}} + \Upsilon_{\mathsf{fix}}$ where the two components have been introduced above (see Example 69 and Section 4.4.4).

As a coproduct of algebraic 2-signatures, $\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}}$ is itself algebraic, and thus the initial model exists. The underlying monad $\mathsf{LC}_{\beta\eta,\mathsf{fix}}$ of the initial model can be understood as the monad of lambda calculus modulo β and η enriched with an *explicit* fixpoint operator $\mathsf{fix} : \mathsf{LC}'_{\beta\eta,\mathsf{fix}} \longrightarrow \mathsf{LC}_{\beta\eta,\mathsf{fix}}$. Now we build by recursion a monad morphism from this monad to the "bare" monad $\mathsf{LC}_{\beta\eta}$ of lambda calculus modulo β and η.

As explained in Section 4.5.1, we need to define an action of $\Upsilon_{\mathsf{LC}_{\beta\eta,\mathsf{fix}}}$ in $\mathsf{LC}_{\beta\eta}$, that is to say an action of $\Upsilon_{\mathsf{LC}_{\beta\eta}}$ plus an action of Υ_{fix}. For the action of $\Upsilon_{\mathsf{LC}_{\beta\eta}}$, we take the one yielding the initial model.

Now, in order to find an action of Υ_{fix} in $\mathsf{LC}_{\beta\eta}$, we choose a fixpoint combinator Y (say the one of Curry) and take the action $\hat{Y}$ as defined at the end of Section 4.4.4.

In more concrete terms, our translation is a kind of compilation which replaces each occurrence of the explicit fixpoint operator $\mathsf{fix}(t)$ with $\mathsf{app}(Y, \mathsf{abs}\, t)$.

Bibliography

[ACCL90] 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. doi:10.1145/96709.96712.

[ACU15] T. Altenkirch, J. Chapman, and T. Uustalu. Monads need not be endofunctors. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:3)2015.

[AHLM19a] B. Ahrens, A. Hirschowitz, A. Lafont, and M. Maggesi. High-level signatures and initial semantics. *CoRR*, 2019, 1805.03740v2. Extended version of publication at CSL 2018 (doi).

[AHLM19b] B. Ahrens, A. Hirschowitz, A. Lafont, and M. Maggesi. Modular specification of monads through higher-order presentations, 2019, 1903.00922.

[Ahr16] B. Ahrens. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science*, 26:3–37, 2016. doi:10.1017/S0960129514000103.

[AL17] B. Ahrens and P. L. Lumsdaine. Displayed Categories. In D. 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.

[AMM18] B. Ahrens, R. Matthes, and A. Mörtberg. From Signatures to Monads in UniMath. *Journal of Automated Reasoning*, 2018. doi:10.1007/s10817-018-9474-4.

[AP04] J. Adámek and H.-E. Porst. On tree coalgebras and coalgebra presentations. *Theor. Comput. Sci.*, 311(1-3):257–283, Jan. 2004. doi:10.1016/S0304-3975(03)00378-5.

[AR99] T. Altenkirch and B. Reus. Monadic presentations of lambda terms using generalized inductive types. In J. Flum and M. 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.

[Bar70] M. Barr. Coequalizers and free triples. *Mathematische Zeitschrift*, 116(4):307–322, Dec 1970. doi:10.1007/BF01111838.

[BdM97] R. S. Bird and O. de Moor. *Algebra of programming*. Prentice Hall International series in computer science. Prentice Hall, 1997.

[BP99] R. S. Bird and R. Paterson. Generalised folds for nested datatypes. *Formal Asp. Comput.*, 11(2):200–222, 1999. doi:10.1007/s001650050047.

[Bra14] M. Brandenburg. *Tensor categorical foundations of algebraic geometry*. PhD thesis, Universität Münster, 2014, 1410.1716.

[Clo10] R. Clouston. Binding in nominal equational logic. *Electr. Notes Theor. Comput. Sci.*, 265:259–276, 2010. doi:10.1016/j.entcs.2010.08.016.

[CoqDev] The Coq development team. *The Coq Proof Assistant, version 8.9*, 2019. URL `http://coq.inria.fr`. Version 8.9.

[ER03] T. Ehrhard and L. Regnier. The differential lambda-calculus. *Theor. Comput. Sci.*, 309(1-3):1–41, 2003. doi:10.1016/S0304-3975(03)00392-X.

[FG10] M. Fernández and M. J. Gabbay. Closed nominal rewriting and efficiently computable nominal algebra equality. In K. Crary and M. Miculan, editors, *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010.*, volume 34 of *EPTCS*, pages 37–51, 2010. doi:10.4204/EPTCS.34.5.

[FH09] M. P. Fiore and C. Hur. On the construction of free algebras for equational systems. *Theor. Comput. Sci.*, 410(18):1704–1729, 2009. doi:10.1016/j.tcs.2008.12.052.

[FH10] M. P. Fiore and C.-K. Hur. Second-order equational logic (extended abstract). In A. Dawar and H. 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.

[FH13] M. P. Fiore and M. Hamana. Multiversal Polymorphic Algebraic Theories: Syntax, Semantics, Translations, and Equational Logic. In *28th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2013, New Orleans, LA, USA, June 25-28, 2013*, pages 520–529. IEEE Computer Society, 2013. doi:10.1109/LICS.2013.59.

[Fio08] M. P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science*, pages 57–68. IEEE Computer Society, 2008. doi:10.1109/LICS.2008.38.

[FM10] M. P. Fiore and O. Mahmoud. Second-Order Algebraic Theories (Extended Abstract). In P. Hlinený and A. 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.

[FPT99] M. P. Fiore, G. D. Plotkin, and D. 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.

[FS17] M. P. Fiore and P. Saville. List Objects with Algebraic Structure. In D. Miller, editor, *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, volume 84 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:18, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.FSCD.2017.16.

[Gir87] J.-Y. Girard. Linear logic. *Theor. Comput. Sci.*, 50(1):1–102, Jan. 1987. doi:10.1016/0304-3975(87)90045-4.

[GP99] M. J. Gabbay and A. 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.

[Gra12] B. Gramlich. Modularity in term rewriting revisited. *Theoretical Computer Science*, 464:3 – 19, 2012. doi:10.1016/j.tcs.2012.09.008. New Directions in Rewriting (Honoring the 60th Birthday of Yoshihito Toyama).

[GU03] N. Ghani and T. 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.

[GUH06] N. Ghani, T. Uustalu, and M. 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.

[Ham03] M. Hamana. Term rewriting with variable binding: An initial algebra approach. In *Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declaritive Programming*, PPDP '03, pages 148–159, New York, NY, USA, 2003. ACM. doi:10.1145/888251.888266.

[HHP93] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, Jan. 1993. doi:10.1145/138027.138060.

[Hir13] T. Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *Logical Methods in Computer Science*, 9(3):10, 2013. doi:10.2168/LMCS-9(3:10)2013. 19 pages.

[HM07] A. Hirschowitz and M. 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.

[HM10] A. Hirschowitz and M. Maggesi. Modules over monads and initial semantics. *Information and Computation*, 208(5):545–564, May 2010. doi:10.1016/j.ic.2009.07.003. Special Issue: 14th Workshop on Logic, Language, Information and Computation (WoLLIC 2007).

[HM12] A. Hirschowitz and M. Maggesi. Initial semantics for strengthened signatures. In D. Miller and Z. 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. doi:10.4204/EPTCS.77.5.

[Hof99] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 204–213. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782616.

[HP07] M. Hyland and J. 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.

[JG07] P. Johann and N. 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.

[JGW78] J. T. J.A. Goguen and E. 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.

[KP93] G. M. Kelly and A. J. Power. Adjunctions whose counits are coequalizers, and presentations of finitary enriched monads. *Journal of Pure and Applied Algebra*, 89(1):163–179, 1993. doi:10.1016/0022-4049(93)90092-8.

[KP10] A. Kurz and D. Petrisan. On universal algebra over nominal sets. *Mathematical Structures in Computer Science*, 20(2):285–318, 2010. doi:10.1017/S0960129509990399.

[LG97] C. Lüth and N. Ghani. Monads and modular term rewriting. In E. Moggi and G. Rosolini, editors, *Category Theory and Computer Science, 7th International Conference, CTCS '97*, volume 1290 of *Lecture Notes in Computer Science*, pages 69–86. Springer, 1997. doi:10.1007/BFb0026982.

[Man76] E. Manes. *Algebraic Theories*, volume 26 of *Graduate Texts in Mathematics*. Springer, 1976.

[ML98] S. Mac Lane. *Categories for the working mathematician*, volume 5 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, second edition, 1998.

[MU04] R. Matthes and T. 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.

[Pow07] A. J. Power. Abstract syntax: Substitution and binders: Invited address. *Electr. Notes Theor. Comput. Sci.*, 173:3–16, 2007. doi:10.1016/j.entcs.2007.02.024.

[Sta13] S. Staton. An Algebraic Presentation of Predicate Logic (Extended Abstract). In F. Pfenning, editor, *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013*, volume 7794 of *Lecture Notes in Computer Science*, pages 401–417. Springer, 2013. doi:10.1007/978-3-642-37075-5_26.

[VAG+] V. Voevodsky, B. Ahrens, D. Grayson, et al. UniMath — a computer-checked library of univalent mathematics. Available at `https://github.com/UniMath/UniMath`.

[VK11] J. Velebil and A. Kurz. Equational presentations of functors and monads. *Mathematical Structures in Computer Science*, 21(2):363–381, 2011. doi:10.1017/S0960129510000575.