UNIVERSITE BRETAGNE LOIRE / MATHSTIC

UNIVERSITÉ SAVOIE MONT BLANC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPERIEURE MINES-TELECOM ATLANTIQUE BRETAGNE PAYS DE LA LOIRE - IMT ATLANTIQUE
COMUE UNIVERSITÉ BRETAGNE LOIRE

ÉCOLE DOCTORALE N° 601
Mathématiques et Sciences et Technologies de l'Information et de la Communication
Spécialité : Informatique

Par

Ambroise LAFONT

Towards an unbiased approach to specify, implement, and prove properties on programming languages

Thèse présentée et soutenue à Nantes , le jour adéquat
Unité de recherche : LS2N

Rapporteurs avant soutenance :

Marcelo FIORE	Professor, University of Cambridge
Peter LEFANU LUMSDAINE	Assistant Professor, CAS Oslo

Composition du Jury :

Attention, en cas d'absence d'un des membres du Jury le jour de la soutenance, la composition du Jury ne comprend que les membres présents

Président :	Prénom Nom	Fonction et établissement d'exercice *(à préciser après la soutenance)*
Examinateurs :	Prénom Nom	Fonction et établissement d'exercice
	Prénom Nom	Fonction et établissement d'exercice
	Prénom Nom	Fonction et établissement d'exercice
	Prénom Nom	Fonction et établissement d'exercice
Dir. de thèse :	Nicolas TABAREAU	Directeur de Recherche, INRIA Rennes
Co-dir. de thèse :	Tom HIRSCHOWITZ	Chargé de Recherche, Université Savoie Mont Blanc

Invité(s) :

Prénom Nom	Fonction et établissement d'exercice

TABLE OF CONTENTS

CHAPITRE 1

INTRODUCTION

1.1 Résumé long (en français)

Cette thèse s'intéresse à la mathématisation de la notion de *langage de programmation*, en portant une attention particulière à la prise en compte de la notion de *substitution*.

La recherche dans le domaine des langages de programmation s'appuie traditionnellement sur une définition de *syntaxe* modulo renommage des variables liées, avec sa *sémantique opérationnelle* associée. Nous nous intéressons à des outils mathématiques permettant de générer automatiquement la syntaxe et la sémantique à partir de données élémentaires.

La spécification de structures algébriques avec variables liées est un enjeu majeur lorsqu'il s'agit de mathématiser la notion de langage de programmation. Deux lignes principales de recherche sont en concurrence : les *ensembles nominaux* [GP99] et les *algèbres de substitution* [FPT99]. Dans cette thèse, nous explorons une voie alternative, proche des algèbres de substitution, proposée par [HM07 ; HM10], qui s'appuie sur la notion de *module sur une monade*.

Dans le chapitre 4, nous aboutissons à la définition de *monade de réduction*, formalisant celle de langage de programmation, tel que le *lambda calcul* muni de la règle de calcul de β*-réduction*. Nous proposons alors un protocole sur la base d'une notion de signature à trois niveaux pour spécifier une certaine classe de langages de programmation :

1. spécification des constructions, par exemple une opération binaire $+$;
2. spécification des équations, par exemple $a + b = b + a$ (commutativité de l'opération binaire $+$) ;
3. spécification des réductions entre termes, par exemple $0 + a \rightsquigarrow a$.

Les deux premiers points définissent ce que nous appelons la *syntaxe* du langage de programmation, tandis que le dernier point concerne la *sémantique* du langage : une réduction entre deux termes correspond à un chemin d'exécution du programme évoluant du premier terme vers le second. A titre d'exemple de langage intégrant les trois niveaux, nous proposons dans la section 4.6 une spécification du lambda calcul avec substitution explicite décrit dans [Kes09].

Afin de motiver les notions mathématiques mises en jeu, nous examinons dans ce résumé le langage de programmation fonctionnel le plus simple que l'on puisse envisager : le lambda calcul pur. Dans la section 1.1.1, nous donnons une première présentation de sa syntaxe, et la dotons d'une opération de substitution. Nous expliquons ensuite, dans la section 1.1.2, comment la notion mathématique de monade permet d'en rendre compte, puis, dans la section 1.1.3, comment la notion de morphisme de modules fournit un moyen d'exprimer une propriété essentielle des constructions de la syntaxe : la *compatibilité à la substitution*. Dans la section 1.1.4, nous caractérisons la syntaxe par son principe de récurrence, que nous formulons par une propriété d'*initialité*. Nous expliquons dans la section 1.1.5 que préciser cette propriété d'initialité requiert une notion de modèle adéquate, laquelle est déterminée par la *signature* : c'est l'occasion de présenter la méthodologie générale de la *sémantique initiale*. Nous examinons ensuite le cas de syntaxes vérifiant des équations (section 1.1.6), avant d'aborder, dans la section 1.1.7, la spécification de la sémantique, c'est-à-dire, dans notre cadre, d'un ensemble de réductions entre termes.

1.1.1 Présentation naïve de la syntaxe du lambda calcul

Nous donnons ici une présentation de la syntaxe du lambda calcul, ainsi qu'un aperçu de quelques difficultés habituellement associées à une telle présentation. On fixe un ensemble infini V de variables, et l'on caractérise récursivement l'ensemble des *termes* ou expressions valides du lambda calcul :

- chaque variable $x \in V$ 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 $f(t)$: c'est 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. Transposons cette identification dans le langage du lambda calcul : nous voulons égaliser 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. Dans cette situation, on dit que x est une *variable liée* dans $\lambda x.t$, et les occurences de x dans t sont alors qualifiées de liées. Les occurences de variables qui ne sont pas liées sont dites *libres*[1].

Ici, 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 précédent, 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 $\lambda y.y$.

La *substitution* est un autre aspect essentiel de la syntaxe du lambda calcul : étant donné un terme t, si l'on remplace toutes les occurrences (libres) 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'équation précédente est comprise comme une étape d'exécution du programme. On dit alors que $(\lambda x.t)\,u$ se *β-réduit* vers $t[x \mapsto u]$. Dans la suite, lorsque nous parlons du lambda calcul sans plus de précision, nous faisons référence à la syntaxe qui ne réalise pas cette identification.

Terminons cette section en soulignant la nécessité que la substitution ne remplace que les occurences libres d'une variable, afin de préserver la propriété suivante : étant donné deux termes $\lambda x.t$ et $\lambda y.t'$ supposés α-équivalents, substituer la même variable dans chacun d'eux fournit deux termes α-équivalents. Si la variable substituée est

1. Une variable peut avoir une occurence liée et une occurence libre dans le même terme : par exemple, x dans $(\lambda x.x)\,x$ est liée dans $\lambda x.x$, mais apparaît librement à droite.

identique à la variable abstraite, le terme $(\lambda x.t)[x \mapsto u]$ est donc tout simplement égal à $\lambda x.t$: par exemple $(\lambda x.x)[x \mapsto u] = \lambda x.x$ est bien α-équivalent à $(\lambda y.y)[x \mapsto u] = \lambda y.y$.

1.1.2 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. Dans le point de vue que nous adoptons ici, les termes α-équivalents sont considérés comme identiques : ainsi, $\lambda x.x = \lambda y.y$.

Au lieu de considérer un ensemble unique de termes avec un ensemble de variables V fixé à l'avance, nous définissons des classes de termes qui utilisent les mêmes variables libres. Notons $L(X)$ l'ensemble des termes dont les variables libres sont choisies dans un ensemble X. Remarquons qu'un terme $t \in L(X)$ se retrouve également dans $L(Y)$ pour tout inclusion $X \subset Y$: en effet, si les variables libres sont choisies parmi les éléments d'un ensemble X, elles sont en particulier choisies parmi les élements de n'importe quel ensemble Y qui contient X.

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 apparaissant dans t par les termes u_x correspondants.

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 adéquat :

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

- la substitution identité est neutre :

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

- toute succession de substitutions est équivalente à une substitution composée :

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

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

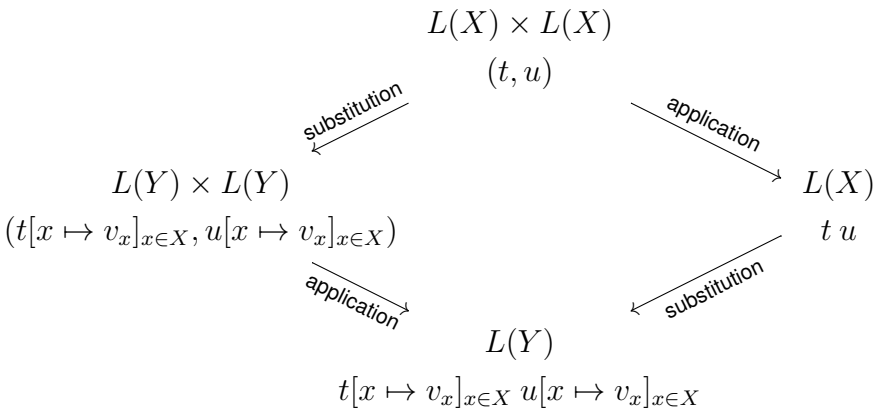
1.1.3 Les constructions sont des morphismes de modules

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 ceci avec 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 suivante dont bénéficie la collection d'ensembles $(L(X) \times L(X))_X$:

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

Cette substitution vérifie les propriétés suivantes :

- la substitution identité est neutre :

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

- toute succession de substitutions est équivalente à une seule substitution composée :

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

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. En fait, $L \times L$ définit aussi une monade, mais la substitution associée ne dérive pas de sa structure de module. 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, cette substitution monadique fournit une paire de termes dont les variables libres sont dans Y. 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 associent 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 vu comme un module sur la monade homonyme.

De même, l'abstraction du lambda calcul induit une collection de fonctions $L(X \amalg \{\star\}) \to L(X)$ qui, à tout terme t dont les variables libres sont choisies dans l'ensemble X étendu avec un nouvel élément $\star$, associe le terme $\lambda \star .t$. La famille $L(X \amalg \{\star\})_X$ est canoniquement munie d'une opération de substitution et définit donc un module. La collection des fonctions d'abstraction induit alors un morphisme de modules, en raison de la commutation avec la substitution :

$$(\lambda \star .t)[x \mapsto t_x]_{x \in X} = \lambda \star . \left(t \left[x \mapsto \begin{cases} \star & \text{si } x = \star \\ t_x & \text{sinon.} \end{cases} \right]_{x \in X \amalg \{\star\}} \right)$$

La construction de module que nous rencontrons ici se généralise à n'importe quel module M sur une monade R : le *module dérivé* M' se définit comme la collection d'ensembles $(M(X \amalg \{\star\}))_X$ munie d'une opération de substitution canonique.

Dans cette thèse, nous nous intéressons exclusivement à des langages de programmation dont les constructions et les réductions commutent avec la substitution, d'où notre intérêt pour les notions de module et de morphisme de modules.

1.1.4 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 t du lambda calcul. Pour ce faire, nous raisonnons par récurrence sur la structure du terme. Si t est une variable x, alors le singleton $\{x\}$ constitue l'ensemble de ses variables libres. 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 : le principe de récurrence est alors une conséquence d'une propriété d'*initialité*. Le lambda calcul est ainsi caractérisé comme la monade "minimale" munie d'une application et d'une lambda abstraction, dans un sens que nous allons illustrer par l'exemple du calcul des variables libres (cet exemple est étudié plus formellement dans la Section 2.5.2).

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 calculée par la réunion $\cup_{z\in t} u_z$. L'union de deux sous-ensembles fournit une opération binaire pour $\mathcal{P}$, que nous assimilons à une "opération d'application" par analogie avec l'opération binaire d'application du lambda calcul. Cette opération associe le sous-ensemble $t \cup u$ au couple (t, u). De même, une opération adéquate d'abstraction $\mathcal{P}(X \amalg \{\star\}) \to \mathcal{P}(X)$ est donnée par $t \mapsto t \cap X$.

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 que pour la monade $\mathcal{P}$, 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 $\mathcal{P}$. La section suivante définit la notion de modèle de sorte que L et $\mathcal{P}$, munies leurs opérations respectives d'application et d'abstraction, sont des modèles de la signature du lambda calcul. Le morphisme de monades free est alors un *morphisme de modèles* de cette signature, grâce aux deux derniers points.

1.1.5 Signatures et modèles

Caractériser un objet d'un certain type par une propriété d'initialité constitue le fondement de ce que l'on appelle *sémantique initiale* ou *spécification algébrique* [JGW78], popularisés par [BM97]. L'objectif est de définir un *langage formel*[2] selon les étapes suivantes :

1. Introduire une notion de signature.

2. Ici, le mot "langage" englobe les types de donnée, les langages de programmation et les calculs logiques, ainsi que les langages pour structures algébriques considérés en algèbre universelle.

2. Construire une notion de modèle associée, s'organisant en une catégorie munie d'un foncteur vers la catégorie d'objets du même type que celui que l'on veut spécifier.

3. Definir l'objet spécifié par la signature comme le modèle initial, s'il existe.

4. Trouver une condition suffisante pour que ce modèle initial existe[3].

Les modèles d'une signature constituent le domaine d'interprétation de l'objet spécifié par la signature : l'initialité induit en effet un principe de récurrence.

Dans cette thèse, nous adoptons cette méthodologie pour spécifier un langage de programmation, depuis sa syntaxe jusqu'à sa sémantique. Concentrons-nous sur l'aspect syntaxique, mathématiquement représenté par le concept de monade. Nous allons donc donner une notion de signature permettant de spécifier une monade : sa catégorie de modèles sera alors équippé d'un foncteur vers la catégorie des monades. Considérons l'exemple de la syntaxe du lambda calcul. Dans la section précédente, nous l'avons caractérisé comme la monade initiale munie d'une application et d'une abstraction. Plus précisément, en généralisant l'exemple de la monade des parties $\mathcal{P}$ étudiée dans la section précédente, nous disons qu'une monade R est munie d'une application et d'une abstraction si elle est dotée à la fois d'une opération binaire, c'est-à-dire d'un morphisme de modules $\mathsf{app}^R : R \times R \to R$, et d'un morphisme de modules $\mathsf{abs}^R : R' \to R$. De manière équivalente, c'est une monade R avec un morphisme de modules de $(R \times R) \amalg R'$ vers R.

La signature Σ_{LC} du lambda calcul associe à toute monade R le module $(R \times R) \amalg R'$ sur R. Un modèle de cette signature est une monade R munie d'une opération binaire, en tant que morphisme de modules de $R \times R$ vers R, ainsi que d'une opération d'abstraction, en tant que morphisme de modules de R' vers R. 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 et l'abstraction.

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

3. Dans la littérature, le mot signature est souvent réservé au cas où une telle condition suffisante est automatiquement satisfaite.

morphisme de monades $f : S \to R$ qui préserve σ, i.e., vérifiant

$$f(\sigma^S(t)) = \sigma^R(f(t)) \tag{1.1}$$

Un tel morphisme de monades définit ce que nous appelons un *morphisme de modèles* entre (R, σ^R) et (S, σ^S). Notons que le membre de droite de l'équation 1.1 nécessite 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[4] $\Sigma(R) \to \Sigma(T)$ 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.

L'existence de la syntaxe associée à une signature quelconque n'est pas systématique[5]. Nous qualifions d'*effective* une telle signature. C'est le cas de toute signature que nous appelons *algébrique*, généralisant l'exemple du lambda calcul. Une telle signature spécifie une syntaxe disposant d'un ensemble d'opérations n-aires, dont certaines lient des variables dans leurs arguments. Ainsi, l'abstraction du lambda calcul est une opération unaire liant une seule variable dans son unique argument.

1.1.6 Syntaxes avec équations (chapitres 2 et 3)

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

Dans le chapitre 2, 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). Les syntaxes associées peuvent être vues comme des syntaxes provenant de signatures algébriques, mais de plus vérifiant certaines équations. Par exemple, il est possible de spécifier une opération binaire commutative (Section 2.8.1). 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. Décrivons maintenant la signature présentable $\Sigma_{\mathsf{comm-bin}}$ as-

4. En réalité, $\Sigma(T)$ n'est pas un module sur R, mais sur T. Cependant, tout morphisme de monades $R \to T$ induit sur n'importe quel module sur T une structure de module sur R.

5. La signature du contre-exemple 27 associe à toute monade R le module $(\mathcal{P}(R(X)))_X$.

sociée : il s'agit d'un quotient de la signature algébrique Σ_{bin} d'une opération binaire qui associe à toute monade R le module $R \times R$. Plus précisément, à toute monade R, la signature $\Sigma_{\mathsf{comm-bin}}$ associe le module $(\mathcal{S}^2 R(X)))_X$, où $\mathcal{S}^2 R(X)$ est l'ensemble des paires d'éléments de $R(X) \times R(X)$ quotienté par la relation $(t, u) \sim (u, t)$, c'est-à-dire, 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. Considérons en effet l'exemple d'une opération binaire associative : nous ne savons pas en donner une signature, présentable ou non. Remarquons que nous pouvons malgré cela donner une définition intuitive de modèle dans ce cas particulier : il s'agit d'une monade R munie d'une opération binaire $b : R \times R \to R$ telle que pour tous x, y, z dans $R(X)$ les expressions $b(b(x, y), z)$ et $b(x, b(y, z))$ sont égales. En d'autres termes, il s'agit d'un modèle (R, b) de la signature d'une opération binaire, tel que les deux morphismes de modules de $R \times R \times R$ vers R, associant à tout triplet (x, y, z) les deux expressions envisagées, sont égales.

Dans le chapitre 3, nous donnons une définition d'équation généralisant cet exemple pour les modèles d'une signature Σ quelconque : il s'agit de la donnée,

- pour chaque modèle R de la signature Σ, de deux morphismes de modules e_R, e'_R de même domaine A_R et codomaine B_R (dans l'exemple ci-dessus, $e_R, e'_R : R \times R \times R \to R$ associent respectivement $b(b(t, u), v)$ et $b(t, b(u, v))$) à un même triplet $(t, u, v) \in R^3(X)$) ;
- pour tout morphisme de modèles $f : R \to S$, de deux morphismes de modules $A_f : A_R \to A_S$ et $B_f : B_R \to B_S$ tels que les diagrammes suivants commutent :

$$\begin{array}{ccc} A_R & \xrightarrow{e_R} & B_R \\ {\scriptstyle A_f}\downarrow & & \downarrow{\scriptstyle B_f} \\ A_S & \xrightarrow[e'_S]{} & B_S \end{array} \qquad \begin{array}{ccc} A_R & \xrightarrow{e'_R} & B_R \\ {\scriptstyle A_f}\downarrow & & \downarrow{\scriptstyle B_f} \\ A_S & \xrightarrow[e'_S]{} & B_S \end{array}$$

Ces données sont soumises comme d'habitude à une condition additionnelle de fonctorialité. On dit qu'un modèle R de la signature Σ vérifie l'équation lorsque $e_R = e'_R$.

Nous considérons alors une extension de la notion de signature : une *2-signature* est une paire composée d'une signature (au sens précédent) et d'un ensemble d'équations associées. Un modèle (ou 2-modèle) d'une 2-signature est un modèle R de

la signature sous-jacente qui vérifie toutes les équations. 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). Techniquement, il s'agit d'équations dont l'action fonctorielle du domaine envoie des morphismes surjectifs de monades sur des morphismes surjectifs de modules, et dont le codomaine est de la forme $R \mapsto R'^{\cdots\prime}$. Tout exemple de signature présentable que nous considérons dans le chapitre 2 peut être reformulée en tant que 2-signature algébriques, induisant la même catégorie de modèles (à isomorphisme près).

1.1.7 Sémantique (chapitre 4)

Il est possible de spécifier par une 2-signature la syntaxe du lambda calcul quotientée par la β-réduction que nous avons mentionnée dans la section 1.1.1 : les termes $(\lambda x.t)\, u$ et $t[x \mapsto u]$ sont ainsi égalisés. Cependant, cette β-réduction fait habituellement partie de la sémantique plutôt que de la syntaxe : c'est le graphe de réductions entre les termes du lambda calcul non quotienté qui est parfois considéré. Nous proposons dans le chapitre 4 la notion de *monade de réduction*, étendant celle de monade, pour rendre compte de la structure additionnelle de réduction. Intuitivement, une monade de réduction est une monade R munie, pour chaque paire de termes $(t, u) \in R(X)$, d'un ensemble de réductions entre t et u que l'on note $t \blacktriangleright u$, et d'une opération de substitution associée : pour toute famille de termes $(v_x)_{x\in X}$ avec $v_x \in R(Y)$, pour toute réduction m entre t et u, il existe une réduction $m[x \mapsto v_x]_{x\in X}$ entre $t[x \mapsto v_x]_{x\in X}$ et $u[x \mapsto v_x]_{x\in X}$. Des équations analogues à celles intervenant dans la définition de module sont requises.

Dans ce contexte, une *signature de réduction* consiste en une signature pour monades, c'est-à-dire une 2-signature, et une famille de *règles de réductions*. Par exemple, la règle de réduction pour la congruence de l'application du lambda calcul s'exprime informellement ainsi, en désignant explicitement l'application du lambda calcul par la construction app :

$$\frac{T \rightsquigarrow T' \qquad U \rightsquigarrow U'}{\mathsf{app}(T, U) \rightsquigarrow \mathsf{app}(T', U')}$$

Cette règle se décompose en trois paires de termes : les hypothèses (T, T') et (U, U'),

et la conclusion $(\mathsf{app}(T,U),\mathsf{app}(T',U'))$, construits à partir des métavariables T, T', U et U'. Ces paires paramétrées induisent des paires de morphismes $(h_{1,1}(R),h_{1,2}(R))$, $(h_{2,1}(R),h_{2,2}(R))$ et $(c_1(R),c_2(R))$ de R-modules entre R^4 et R pour n'importe quel modèle R de la signature du lambda calcul (en temps que monade). De plus, cette construction est fonctorielle. Plus précisément, si $f : R \to S$ est un morphisme de modèles, alors les diagrammes suivants sont commutatifs :

$$\begin{array}{ccc} R^4 & \xrightarrow{h_{i,j}(R)} & R \\ {\scriptstyle f^4}\downarrow & & \downarrow{\scriptstyle f} \\ S^4 & \xrightarrow[h_{i,j}(S)]{} & S \end{array} \qquad \begin{array}{ccc} R^4 & \xrightarrow{c_i(R)} & R \\ {\scriptstyle f^4}\downarrow & & \downarrow{\scriptstyle f} \\ S^4 & \xrightarrow[c_i(S)]{} & S \end{array}$$

Ainsi formulée, cette règle de congruence entre dans notre définition de *règle de réduction* détaillée en Section 4.3. Une *action* de cette règle dans une monade de réduction R munie d'une opération binaire app est la donnée d'une réduction $\mathsf{app\text{-}cong}(m_T,m_U)$ entre $\mathsf{app}(T,T')$ et $\mathsf{app}(U,U')$ pour tout $(T,T',U,U') \in R^4(X)$, toute réduction m_T entre T et T', et toute réduction m_U entre U et U'. Il faut de plus que app-cong commute avec la substitution, c'est-à-dire que l'équation suivante soit vérifiée :

$$\mathsf{app\text{-}cong}(m_T,m_U)[x \mapsto v_x]_{x\in X} = \mathsf{app\text{-}cong}(m_T[x \mapsto v_x]_{x\in X}, m_U[x \mapsto v_x]_{x\in X})$$

Le théorème principal (Théorème 128) du chapitre 4 affirme l'existence d'un modèle initial d'une signature de réduction composée d'une 2-signature effective et de n'importe quelle famille de règles de réduction. Par modèle, on entend ici une monade de réduction dont on s'est donné une structure de modèle de la 2-signature pour la monade sous-jacente, ainsi qu'une validation de toute les règles de réduction. Le modèle initial est construit à partir du modèle initial de la 2-signature, et la structure additionnelle de réductions est construite inductivement à partir des règles de réduction. Outre quelques variantes de signatures de réduction pour le lambda calcul avec β-reduction, nous proposons, dans la section 4.6, une signature pour le lambda calcul avec substitution explicite, tel que décrit dans [Kes09] par un ensemble de constructions soumis à une équation syntaxique, auquel s'ajoute une notion de réductions entre termes. Cette spécification se fait en trois étapes, conformément au protocole que nous avons annoncé au début de cette introduction : une 1-signature pour les opérations du langage, une 2-signature prenant en compte l'équation syntaxique, et une signature de

réduction spécifiant les réductions adéquates.

1.1.8 Formalisation

Les preuves des résultats principaux des chapitres 2 et 3 ont été vérifiées à l'aide de l'assistant de preuve Coq. Il s'agit d'un logiciel dans lequel il est possible de reproduire (ou *formaliser*) des définitions mathématiques ainsi que des démonstrations qui, si elles sont validées par le logiciel, sont incontestables. Cette garantie théorique est en pratique à nuancer pour les raisons suivantes :

1. la théorie des types dépendantes sur laquelle se fonde le logiciel est complexe, ainsi que sa preuve de correction, laquelle justifie notre certitude de principe ;

2. comme tout projet informatique, le logiciel n'est pas à l'abri d'erreurs de développements qui peuvent le conduire à accepter des démonstrations erronées ;

3. même si la démonstration d'un théorème est validée, c'est au mathématicien de s'assurer que l'énoncé mathématique qu'il a formalisé correspond effectivement à ce qu'il a écrit sur le papier.

Le premier point est l'occasion de remarquer que la formalisation des démonstrations mathématiques limite la charge mentale du lecteur soupçonneux, qui n'a plus qu'à se poser la question de la validité de la preuve de correction de la théorie des types, et n'est plus contraint de vérifier chaque démonstration, tâche mécanique (si la preuve est suffisamment détaillée) parfaitement adaptée à l'outil informatique. Néanmoins, cette vision souffre du défaut du porte-clefs, ou encore, de l'Anneau Unique : si l'idée d'un anneau pour les rassembler tous paraît séduisante, en cas de perte, le préjudice est considérable. C'est le même problème que l'on retrouve chez toute théorie qui prétend fonder les mathématiques : si celle-ci s'avère incohérente, toute démonstration dans ce cadre devient potentiellement inutilisable à jamais.

Jusqu'à présent, aucun souci sérieux n'a remis en cause la théorie des types sur laquelle se fonde Coq. En revanche, comme le suggère le deuxième point, des contradictions sont régulièrement exhibées en exploitant des défauts d'implémentation du logiciel, qui sont alors immédiatement corrigés. Mais en réalité, le troisième point est le plus problématique : en effet, il est possible de se tromper dans la formalisation d'une définition, car le logiciel Coq ne nous est d'aucune aide dans cette étape. Par suite,

un énoncé évoquant cette définition n'a pas la signification qu'on lui prête a priori, et pourrait se révéler faux dans le sens qu'on lui voudrait, puisqu'une démonstration formalisée utiliserait la mauvaise définition. Un exemple extrême consisterait à définir par des moyens élaborés un certain ensemble qui se révélerait être l'ensemble vide, à cause d'une erreur dans une étape intermédiaire de la définition. Notons que l'ensemble vide a ceci de particulier que ses éléments vérifient n'importe quelle propriété. Le lecteur d'une formalisation peut donc certes se dispenser de vérifier les preuves, mais il doit porter une attention particulière aux définitions.

Pour nos développements, nous avons choisi de nous appuyer sur la bibliothèque `UniMath` de Coq. Celle-ci comporte en effet quelques avantages, malgré un problème de "taille" (que nous mentionnerons brièvement) :

i elle comporte un certains nombre de définitions et de résultats que nous avons pu exploiter ;

ii elle utilise un minimum de fonctionnalités du langage ;

iii elle intègre l'*axiome d'univalence* et un certains nombre de ses conséquences.

Concernant le point i, `UniMath` propose, entre autres, une implémentation élaborée de la théorie des catégories, et des ensembles quotients.

Le point ii permet de limiter la complexité de la théorie des types en laquelle il nous faut faire confiance (point 1 mentionné plus haut), bien que cet argument soit très contestable au regard du problème de taille dont il est question plus loin. Il réduit également le risque d'être confronté à des erreurs d'implémentation (point 2 susmentionné), en se restreignant à des fonctionnalités éprouvées de Coq.

L'axiome d'univalence mentionné par le point iii en tant que tel n'est pas utilisé de manière crucial. Il a cependant l'avantage de rassembler en ce seul axiome un certain nombre de conséquences utiles, par exemple l'extensionnalité fonctionnelle (deux fonctions sont égales si elles envoient chaque élément sur la même image), ou encore l'extensionnalité propositionnelle (deux propositions logiquement équivalentes sont égales) qui permet de manipuler convenablement la notion de sous-ensemble.

Pour finir, la bibliothèque `UniMath` présente un important défaut dû à un problème de "taille". En effet, elle nécessite une option de compilation qui rend le système contradictoire, en rendant possible l'implémentation d'une variante du paradoxe de Russel. Cette option est en pratique utilisée dans la définition (imprédicative) des quotients.

Nous avons ignoré l'existence de cette incohérence particulière au système dans les raisonnements que nous avons formalisés, et c'est pourquoi nous espérons que nos preuves n'exploitent pas cette faille.

1.2 Long summary

TODO traduire la l'introduction française longue

1.3 Initial semantics

TODO enlever cette partie qui fait peut etre partie du résumé long finalement 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* [BM97].

This concept offers the following methodology to define a *formal language*[6]:

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[7].

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.

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

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

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

Chapter 2 is based on the following article: [Ahr+19a] *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 3, 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: [Ahr+19b] *Modular specification of monads through higher-order presentations* .

Finally, in Chapter 4, we define reduction monads, and an associated notion of signatures and models. The main result of this chapter is a natural condition on a signature for its initial model to exist.

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

1.4 Computer-checked formalization

The intricate nature of our main results made it desirable to provide a mechanically checked proof of these results. We achieved this work for Chapters 2 and 3.

Our computer-checked proof is based on the `UniMath` library [VAG+], which itself is based on the proof assistant Coq [CoqDev19]. 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`.

1.5 Related work

1.5.1 Syntax and monads

TODO parler de list objects ici 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*[8], 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, 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. 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].

8. 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.

1.5.2 Syntax with equations

Ahrens [Ahr16] introduces a notion of 2-signature in the slightly different context of (relative) monads into preordered sets, where the preorder models the reduction relation. In Chapter 3, we consider an analogous notion of 2-signature based on monads on sets. In some sense, Chapter 3 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[9]. In another classical paper, Kelly and Power [KP93] explained how any finitary monad can be presented as a coequalizer of free monads[10]. There, free monads correspond to our initial models of an algebraic 1-signature without any binding construction.

As mentioned above, the present work is also closely related to that of Fiore and collaborators:

- 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.

- 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 Sec-

9. Fiore and Saville [FS17] give an enlightening generalization of the construction by Barr.
10. Their work has been applied to various more general contexts (e.g. [Sta13]).

tion 4.7 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[11]. 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 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.

1.5.3 Syntax with reductions

Reduction signatures of Chapter 4 allow for the specification of

1. terms constructions, including constructions that bind variables, e.g., abstraction;

2. syntactic equalities between terms; and

3. reduction rules, including reduction rules with hypotheses, e.g., congruence rules for the term constructions.

11. 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.

Ahrens [Ahr16] gives a notion of signature that allows for the specification of syntax with binding operations, as well as reduction rules on that syntax. The format for reduction rules considered there does not allow expressing rules with hypotheses, e.g., the aforementioned congruence rules. Instead, the congruence rules are hard-coded in [Ahr16], so that the head-β-reduction (and other limited variants) cannot be specified by that formalism. The constructors are modelled by morphisms of modules between modules into preordered sets, i.e., by families of preorder-preserving maps.

Signatures for rewriting systems, and initial semantics for them, are given by Hamana [Ham03] under the name “binding term rewriting system (BTRS)”. Hamana considers preorder-valued functors. There, signatures for rewrite rules allow for rules without hypotheses only, though some rules with hypotheses, in particular, congruence rules, seem to be hard-coded in Hamana’s framework (see [Ham03, Figure 3]).

CHAPTER 2

PRESENTABLE SIGNATURES

In the present chapter extracted from [Ahr+19a], 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 algebraic signatures mentioned above. Specifically, our signatures form a cocomplete category and our condition is preserved by colimits (Section 2.6). Examples are given in Section 2.8.

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 a pushout.

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 2.1 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 2.2 and 2.3 respectively. In Section 2.4, we give our definition of a syntax, and we present our first main result, a modularity result about merging extensions of syntax. In Section 2.5, we show through examples how recur-

sion can be recovered from initiality. Our notion of *presentation of a signature* appears in Section 2.6. There, we also state our second main result: presentable signatures generate a syntax. The proof of that result is given in Section 2.7. Finally, in Section 2.8, we give examples of presentable signatures.

2.1 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.

2.1.1 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* $f\colon R \to R'$ *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 thesis. 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 $\underline{\mathsf{W}}\colon \mathsf{Set} \longrightarrow \mathsf{Set}$ the constant functor $\underline{\mathsf{W}} := X \mapsto W$. Then $\underline{\mathsf{W}}$ is trivially an R-module since $\underline{\mathsf{W}} = \underline{\mathsf{W}} \cdot R$.
- 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 $\mathsf{Mod}(R)$.

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.*

2.1.2 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]

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

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.

Composition and identity morphisms are the expected ones. 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.*

2.1.3 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

$$\begin{array}{ccc} M(R(X)+*) & \xrightarrow{\rho^{M'}_X} & M(X+*) \\ {\scriptstyle [M(R(i_X),\eta_{X+*}\circ \underline{*})]}\downarrow & \nearrow_{\rho^{M}_{X+*}} & \\ M(R(X+*)) & & \end{array} \tag{2.1}$$

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

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.

Derivation is a continuous and cocontinuous endofunctor on the category $\mathrm{Mod}(R)$ of modules over a fixed monad R. In particular, derivation can be iterated: we denote by $M^{(k)}$ the k-th derivative of M.

Definition 10. Given a list of 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 counit of the following adjunction:

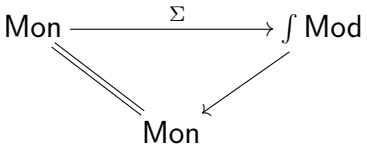
Proposition 13 (`deriv_adj`)**.** *The endofunctor of* $\mathrm{Mod}(R)$ *mapping M to the R-module $M \times R$ is left adjoint to the derivation endofunctor, the counit being the substitution morphism σ.*

2.2 The category of signatures

In this section, we give our notion of signature. The purpose of a signature is to act on monads. An action of a signature Σ in a monad R should be a morphism from some 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 $\int \mathrm{Mod}$ to the category Mon, that is, a functor $\Sigma : \int \mathrm{Mod} \to \mathrm{Mon}$ making the following diagram

commute:



Now we give some 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 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* $\tilde{(-)} : \mathsf{SigStrength} \longrightarrow \mathsf{Sig}$*.*

2.3 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 $\mathsf{abs}\colon \mathsf{LC}' \longrightarrow \mathsf{LC}$ 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`.

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.

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 $f^*(R, r)$.

2.4 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.

2.4.1 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*[6] *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*.

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.

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 $\Sigma(R)$.*

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.

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

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

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

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

$$\Sigma\Big(\Sigma(R)+\mathsf{Id}\Big) \xrightarrow{\Sigma([r,\eta])} \Sigma(R)\cdot R \xrightarrow{\rho^{\Sigma(R)}} \Sigma(R) \longrightarrow \Sigma(R)+\mathsf{Id} \tag{2.2}$$

- *The natural transformation* $[r, \eta] : \Sigma(R) + \mathsf{Id} \to R$ *is a model morphism, that is, it commutes suitably with the* Σ*-actions of Diagram* (2.2) *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 2.6, we show a more general effectiveness result: Theorem 35 states that *presentable* signatures, which form a superclass of algebraic signatures, are effective.

2.4.2 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 $\int^{\Sigma} \mathsf{Mon}^{\Sigma}$ of models:

- An object of $\int^{\Sigma} \mathsf{Mon}^{\Sigma}$ is a triple (Σ, R, r) where Σ is a signature, R is a monad, and r is an action of Σ in R.
- A morphism in $\int^{\Sigma} \mathsf{Mon}^{\Sigma}$ from (Σ_1, R_1, r_1) to (Σ_2, R_2, r_2) consists of a pair (i, m) of a signature morphism $i : \Sigma_1 \longrightarrow \Sigma_2$ and a morphism m of Σ_1-models from (R_1, r_1) to $(R_2, i^*(r_2))$.

- It is easily checked that the obvious composition turns $\int^{\Sigma} \mathsf{Mon}^{\Sigma}$ into a category.

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

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

The proof uses, in particular, the following fact:

Lemma 33 (`rep_cleaving`)**.** *The projection $\pi : \int^{\Sigma} \mathsf{Mon}^{\Sigma} \to \mathsf{Sig}$ 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*

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

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 2.6 we study *presentable* signatures, which we show to be effective and closed under colimits.

2.5 Recursion

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

2.5.1 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 universal property of 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 $\hat{\Sigma}_{LJ}$, satisfying the expected equations. 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.

2.5.2 Example: Computing the set of free variables

As above, we denote by $\mathcal{P}X$ the powerset of X. Union gives 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.

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

2.5.4 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.

2.6 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.

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).

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 2.7.

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.*

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 2.7 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 (in the case when one of them is Θ, see `har_ binprodR_ isPresentable`)*

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.*

Proof. A colimit of an algebraic is presentable, by Theorem 38, hence effective, by Theorem 35 □

2.7 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 $X : \mathrm{Set}_0$, $\sim_X$ 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

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

with a suitable $\overline{\mu} : \overline{R} \cdot \overline{R} \longrightarrow \overline{R}$ satisfying the monad laws. But π is epi, and hence so is $\pi \cdot \pi = \pi\overline{R} \circ R\pi$ since epis are closed under composition and R preserves epimorphisms. Thus, 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 sections, 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.*

Proof sketch. As before, we denote by $\hat{\Upsilon}$ the initial Υ-model, as well as—by abuse of notation—its underlying monad. For each set X, we consider the equivalence relation $\sim_X$ on $\hat{\Upsilon}(X)$ defined as follows: for all $x, y \in \hat{\Upsilon}(X)$ we stipulate that $x \sim_X y$ if and only

if $i_X(x) = i_X(y)$ for each (initial) morphism of Υ-models $i : \hat{\Upsilon} \to F^*S$ with S a Σ-model and F^*S the Υ-model induced by $F : \Upsilon \to \Sigma$.

By 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 Proposition 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 follows from 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 Theorem 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. □

2.8 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.

2.8.1 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 standard integer addition.

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 $\Theta \times \mathbb{N}\langle\Theta\rangle$.

2.8.2 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 $\mathfrak{S}_k$ of permutations of the set k, that is, to consider the colimit of the following one-object diagram:

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

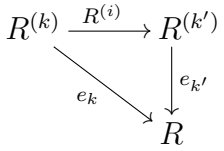
where σ ranges over the elements of $\mathfrak{S}_k$. We denote by $\mathcal{S}^{(k)}\Theta$ the resulting signature presented by the projection $\Theta^{(k)} \to \mathcal{S}^{(k)}\Theta$. By universal property of the quotient, a model of it consists of a monad R with an action $m : R^{(k)} \to R$ that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

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

Fortunately, a finer encoding is provided by the following simple colimit of presentable signatures. The crucial point here is that, for each k, all natural injections from $\Theta^{(k)}$ to $\Theta^{(k+1)}$ induce the same canonical injection from $\mathcal{S}^{(k)}\Theta$ to $\mathcal{S}^{(k+1)}\Theta$. We thus have a natural colimit for the sequence $k \mapsto \mathcal{S}^{(k)}\Theta$ and thus a signature $\operatorname{colim}_k \mathcal{S}^{(k)}\Theta$ which, as a colimit of presentable signatures, is presentable (Theorem 38, Item 2).

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

that for each injection $i : k \to k'$ the following diagram commutes:



2.8.3 Example: Adding an explicit substitution

Explicit substitution was introduced by Abadi et al. [Aba+90] 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

$$\mathsf{subst}_p : R^{(p)} \times R^p \longrightarrow R. \tag{2.3}$$

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 $\mathfrak{S}_p$ 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 $\mathfrak{S}_p$.

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 $u : I_p \longrightarrow I_q$, where I_n denotes the set $\{1, \ldots, p\}$. 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:

$$\begin{array}{ccc} R^{(p)} \times R^q & \xrightarrow{R^{(p)} \times R^u} & R^{(p)} \times R^p \\ {\scriptstyle R^{(u)} \times R^q}\downarrow & & \downarrow{\scriptstyle \mathsf{subst}_p} \\ R^{(q)} \times R^q & \xrightarrow[\mathsf{subst}_q]{} & R \end{array} \tag{2.4}$$

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 $\int^{p:\mathbb{F}} R^{(\underline{p})} \times R^{\overline{p}}$, where covariant (resp. contravariant) occurrences of the bifunctor have been underlined (resp. overlined), and the category $\mathbb{F}$ is the full subcategory of Set whose objects are natural numbers. Thus we have a canonical morphism

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

Abstracting over R, we obtain the following:

Definition 49. *Integrated substitution*

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

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

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

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*.

2.8.4 Example: Adding a coherent fixed-point operator

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

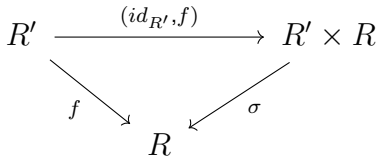
- for each $n \in \mathbb{N}$, a notion of *n-ary fixed-point operator* in a monad;
- a notion of *coherent fixed-point operator* in a monad, which assigns, in a "coherent" way, to each $n \in \mathbb{N}$, an n-ary fixed-point operator.

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,



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 address later in Section 3.4.4, after extending our notion of signature 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 for a unary fixed-point operator. Thus, $\hat{Y}$ is 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 subset of $\mathsf{LC}_{\beta\eta}\emptyset$ consisting of (closed) fixed-point combinator 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

$$\begin{aligned} \mathsf{app}(Y_f, t) &= \mathsf{app}(\mathsf{abs}\ f_X(\mathsf{app}(*, **)), t) && (2.5)\\ &= f_X(\mathsf{app}(t, **)) && (2.6)\\ &= \mathsf{app}(t, f_X(\mathsf{app}(t, **))) && (2.7)\\ &= \mathsf{app}(t, \mathsf{app}(Y_f, t)) && (2.8) \end{aligned}$$

where (2.5) comes from the definition of Y_f (and naturality of f). Equality (2.6) follows from β-reduction, Equality 2.7 from the definition of a fixed-point operator. Finally, Equality 2.8 comes from the equality $\mathsf{app}(Y_f, t) = f_X(\mathsf{app}(t, **))$, which is obtained by chaining the equalities from (2.5) to (2.6). 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 2.8.3.

- 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 $\mathfrak{S}_n$.

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*[9] rough n-ary fixpoint operator is $n \times (\Theta^{(n)})^n$.

8. In the following, we similarly write n instead of I_n in order to make equations more readable.
9. The adjective *syntactic* means here that we do not deal with the equation.

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

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 $\coprod_n (\Theta^{(n)})^n/\mathfrak{S}_n$. 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 2.8.3, which we now introduce.

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

$$\begin{array}{ccc} p \times (R^{(p)})^q & \xrightarrow{p \times (R^{(p)})^u} & p \times (R^{(p)})^p \\ {\scriptstyle u \times (R^{(u)})^q}\downarrow & & \downarrow{\scriptstyle \mathsf{fix}_p} \\ q \times (R^{(q)})^q & \xrightarrow[\mathsf{fix}_q]{} & R \end{array} \tag{2.9}$$

These conditions have an interpretation in terms of a coend, just as we already encountered in Section 2.8.3. 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 $\int^{n:\mathbb{F}} \underline{n} \times (R^{(\underline{n})})^{\overline{n}}$ to R where, for every $n \in \mathbb{N}$, the n-th component is a (rough)[10] n-ary fixpoint operator.

Now, the natural signature for a syntactic coherent fixed-point operator is $\int^{n:\mathbb{F}} \underline{n} \times (\Theta^{(\underline{n})})^{\overline{n}}$. Thus, given a presentable signature Σ, we can safely extend it with a syntactic coherent fixed-point operator by adding the presentable signature

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

to Σ.

10. As in Section 2.8.3, invariance follows from coherence.

CHAPTER 3

ALGEBRAIC 2-SIGNATURES

This chapter is extracted from [Ahr+19b].

The presentable signatures of Chapter 2 allow to specify syntaxes satisfying some equations by considering colimits of algebraic signatures. However, it seems quite limited: for example, we don't know how to specify an associative operation by a presentable signature. This motivates the work of the present chapter: our extended notion of signature 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 2 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 do not know of any counter-example. Conversely, algebraic 2-signatures take into account operations that we do not know how to specify using a presentable signature, such as an associative operation.

3.1 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 $\Sigma := \Theta \times \Theta + \Theta'$, $\hat{\Sigma}$ may be understood as the monad LC of syntactic terms of the lambda calculus (see Section 2.2), 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 cover such equations, the approach in the first-order case suggests to identify p as the coequalizer of a parallel pair of arrows 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-equation 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 mere natural transformations, cannot be understood as morphisms 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 natural transformations: indeed they are morphisms of *modules over* $\hat{\Sigma}$. Accordingly, a (second-order) presentation for a monad R could be a diagram

$$T \overset{f}{\rightrightarrows} \hat{\Sigma} \xrightarrow{p} R \tag{3.1}$$

where Σ is an algebraic 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, recall that we introduced the category Mon^{Σ} "of models of Σ", whose objects are monads "equipped with an action of Σ" (Proposition 24). Of course $\hat{\Sigma}$ is equipped with such an action which turns it into the initial object. Now, we define the full subcategory of models satisfying the equation f, and require R to be the initial object therein. Our definition is suited to the case where the equation f is

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

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 our notion of 2-signature, which is similar to that 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 3.4), and we also show through a simple example how our recursion principle applies (see Section 3.5).

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

3.2 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.

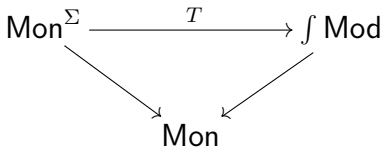
3.2.1 Equations

Our equations are analogous to those considered by Ahrens in [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 $\int \mathsf{Mod} \to \mathsf{Mon}$.

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

$$\begin{array}{l} \Theta^2 \xrightarrow{\mathsf{swap}} \Theta^2 \xrightarrow{\tau} \Theta \ . \\ \Theta^2 \xrightarrow[\tau]{\qquad\qquad} \Theta \end{array}$$

See also Section 3.4.1 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:

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

3.2.2 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).

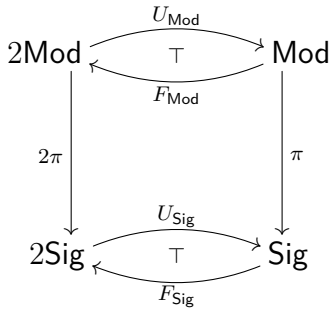
Definition 70. A 2-signature (Σ, E) is said to be **effective** if its category of models $\mathsf{Mon}^{(\Sigma,E)}$ has an initial object, denoted $\widehat{(\Sigma, E)}$.

In Section 4.4, we aim to find sufficient conditions for a 2-signature (Σ, E) to be effective.

3.2.3 Modularity for 2-signatures

In this section, we define the category $2\mathrm{Sig}$ of 2-signatures and the category $2\mathrm{Mod}$ 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 2.4.2;
- 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 models of the summands[2].

We start by defining the category $2\mathsf{Sig}$ 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 $2\mathsf{Sig}$.

We now study the existence of colimits in $2\mathsf{Sig}$. We know that Sig is cocomplete, and we use this knowledge in our study of $2\mathsf{Sig}$, 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 $U_{\mathsf{Sig}}(\Sigma, E) := \Sigma$.

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

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 $2\mathrm{Sig}$ 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 $2\mathrm{Sig}$ 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 summands. 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 $2\mathsf{Mod}$ 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* 2π *from* $2\mathrm{Mod}$ *to* $2\mathrm{Sig}$ *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* $2\mathrm{Mod}$ *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".

3.2.4 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 3.3.

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[3] in `LCBetaEta`.

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

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.*

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

3.3 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 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 to 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 $\left(\hat{\Sigma}/(f_i), \sigma/(f_i)\right)$ 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 $e_1 = e_2$. □

3.4 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.

3.4.1 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 3.4.3 below. However, at the present state we cannot model theories with partial construction (e.g., fields).

3.4.2 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-signature morphisms f and g. Note that the set of equations of Ξ is E_2, by construction of the coequalizer in the category of 2-signatures. Now, consider the algebraic 2-signature $\Xi' = (\Sigma_2, E_2 + (3.2))$ consisting of the 1-signature Σ_2 and the equations of E_2 plus the following elementary equation (see Examples 57 and 81):

$$\begin{aligned} &\Sigma_1 \xrightarrow{f} \Sigma_2 \xrightarrow{\tau^{\Sigma_2}} \Theta \ . \\ &\Sigma_1 \xrightarrow[g]{} \Sigma_2 \xrightarrow[\tau^{\Sigma_2}]{} \Theta \end{aligned} \tag{3.2}$$

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 2, 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 $\int^{n \in \mathbb{F}} F(n) \times \_^n$ where $\mathbb{F}$ 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 $\int^{n \in \mathbb{F}} F(n) \times \Theta^n$, 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.

3.4.3 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, Definition 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:

$$\begin{array}{ll} \Theta \xrightarrow{\eta_T \cdot \Theta} T \cdot \Theta \xrightarrow{\tau} \Theta & \qquad T \cdot T \cdot \Theta \xrightarrow{\mu_T \cdot \Theta} T \cdot \Theta \xrightarrow{\tau} \Theta \\ \Theta \xrightarrow[1]{} \Theta & \qquad T \cdot T \cdot \Theta \xrightarrow[T\tau]{} T \cdot \Theta \xrightarrow[\tau]{} \Theta \end{array} \tag{3.3}$$

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 $\int^{n \in \mathbb{F}} F(n) \times \_^n$ [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 top-right composite is epimorphic by composition of epimorphisms. Thus, the bottom-left composite is also epimorphic, hence so is $F(f)$ as the last morphism of this composition.

□

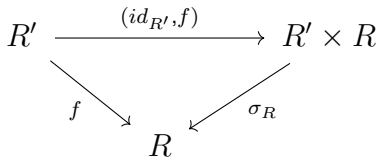
In conclusion, we have exhibited the algebraic 2-signature (Σ_T, E'_T), where E'_T extends the family E_T with the two elementary equations of Diagram 3.3. This signature allows to enrich any other algebraic 2-signature with the operations of the algebraic theory T, subject to the relevant equations.

3.4.4 Fixpoint operator

Here, we show the algebraic 2-signature corresponding to a fixpoint operator. In Section 2.8.4, 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 2.8.4, 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':



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

$$e_{\text{fix}} : \begin{array}{l} \Theta' \xrightarrow{\langle 1,\tau \rangle} \Theta' \times \Theta \xrightarrow{\sigma} \Theta \\ \Theta' \xrightarrow[\tau]{} \Theta \end{array} \tag{3.4}$$

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.

3.5 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.

3.5.1 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 $\Upsilon = (\Sigma, E)$ to an arbitrary monad T.

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.

3.5.2 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 3.4.4).

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 4.7.1, 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 3.4.4.

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

CHAPTER 4

REDUCTION MONADS AND THEIR SIGNATURES

TODO uniformiser la notation $t\{x := *\}$ et $t\{f\}$ avec les autres chapitres ? In this chapter, we study *reduction monads*, which are essentially the same as monads relative to the free functor from sets into graphs. The statements here are not formalized. As for the other chapters, this is a product of a collaboration with Benedikt Ahrens, André Hirschowitz and Marco Maggesi.

Reduction monads abstract two aspects of the lambda calculus: on the one hand, in the monadic viewpoint, it is an object equipped with a well-behaved substitution; on the other hand, in the graphical viewpoint, it is an oriented graph whose vertices are terms and whose edges witness the reductions between two terms.

We study presentations of reduction monads. To this end, we propose a notion of *reduction signature*. As usual, such a signature plays the rôle of a virtual presentation, and specifies arities for generating operations—possibly subject to equations—together with arities for generating reduction rules. For each such signature, we define a category of models. Any model is, in particular, a reduction monad, and, in the spirit of Initial Semantics, we define the reduction monad presented (or specified) by the given reduction signature to be, if it exists, the initial object of this category of models.

The main result of this chapter identifies a class of reduction signatures which specify a reduction monad in the above sense. In particular, the lambda calculus is naturally specified by such a signature.

4.1 Introduction

The lambda calculus has been a central object in theoretical computer science for decades. However, the corresponding mathematical structure does not seem to have

been identified once and for all. In particular, two complementary viewpoints on the (pure untyped) lambda calculus have been widespread: some consider it as a graph (or a preorder, or a category), while others view it as a monad (on the category of sets). The first account incorporates the β-reduction, while the second addresses substitution but incorporates only the β-equality. Merging these two perspectives led Lüth and Ghani [LG97] to consider monads on the category of preordered sets, and Ahrens [Ahr16] to consider monads relative to the free functor from sets into preorders. In the present chapter, we propose a variant of their approaches. Here we call *reduction monad* a monad relative to the natural injection of sets in graphs, and of course the lambda-calculus yields such a reduction monad. Our main contribution concerns the generation of reduction monads by syntactic (possibly binding) operations (possibly subject to equations) and reduction rules. As is common in similar contexts, we propose a notion of signature for reduction monads, which we call "reduction signatures". Each reduction signature comes equipped with the category of its models: such a model is a reduction monad "acted upon" by the signature. A reduction signature may be understood as a virtual presentation: when an initial model exists, it inherits a kind of presentation given by the action of the signature, and we say that the signature is *effective*. We identify a natural criterion for a signature to be effective. As should be expected, we give an effective reduction signature specifying the lambda calculus with its reduction rules, which yields a new, high-level definition of the (pure untyped) lambda calculus.

4.1.1 Terminology and notations

In this section, we set up some terminology and notations that we use in this chapter.

Signature for monads By **signatures for monads**, we mean 2-signatures of Chapter 3. Sometimes they also refer to 1-signatures of Chapter 2, as particular cases of 2-signatures without equations. The letter Σ is usually associated with a signature for monads.

Models of signature for monads In order to avoid any confusion with the notion of model that we introduce here, we refer to models of a signature Σ for monads as **Σ-monads**.

Substitution Let R be a monad, M be a R-module, t an element of $M(X)$ and f a function from X to $R(Y)$. Then, we denote by $t\{f\} \in R(Y)$ the substitution of all the variables in t by the corresponding term given by f.

Unary substitution We abbreviate $t\{x \mapsto \text{if } x = y \text{ then } u \text{ else } x\}$ as $t\{y := u\}$.

4.1.2 Plan of the chapter

In Section 4.2, we define reduction monads. In Section 4.3, we present our take on reduction rules. This enables us to define signatures for reduction monads—*reduction signatures*—in Section 4.4. Section 4.5 is devoted to the proof of our theorem of effectivity for these signatures. Then, in Section 4.6, we give a detailed example of a reduction signature specifying the lambda calculus with explicit substitutions of [Kes09]. Finally, in Section 4.7, we explain the recursion principle which, as usual, can be derived from initiality in our categories of models.

4.2 Reduction monads

Here below, we define *the category of reduction monads* in Section 4.2.1. We also consider some examples of reduction monads, in Section 4.2.2.

4.2.1 The category of reduction monads

Definition 93. A **reduction monad** R is given by:

1. a monad on sets (the monad of *terms*), that we still denote by R, or by $\underline{R}$ when we want to be explicit;
2. an R-module $\mathsf{Red}(R)$ (the module of *reductions*);
3. a morphism of R-modules $\mathsf{red}_R : \mathsf{Red}(R) \to R \times R$ (*source* and *target* of rules).

We set $\mathsf{source}_R := \pi_1 \circ \mathsf{red}_R : \mathsf{Red}(R) \to R$, and $\mathsf{target}_R := \pi_2 \circ \mathsf{red}_R : \mathsf{Red}(R) \to R$

For a reduction monad R, a set X, and elements $s, t \in R(X)$, we think of the fiber $\mathsf{red}_R(X)^{-1}(s,t)$ as the set of "reductions from s to t". We sometimes write $m : s \blacktriangleright t : R(X)$, or even $m : s \blacktriangleright t$ when there is no ambiguity, instead of $m \in \mathsf{red}_R(X)^{-1}(s,t)$.

Remark 94. Note that for a given reduction monad R, set X, and $s, t : R(X)$, there can be multiple reductions from s to t, that is, the fibre $s \blacktriangleright t$ is not necessarily a subsingleton.

Remark 95. Let R be a reduction monad, X and Y two sets, $f : X \to R(Y)$ a substitution, and u and v two elements of $R(X)$ related by $m : u \blacktriangleright v$. The module structure on $\mathsf{Red}(R)$ yields a reduction denoted $m\{f\}$ between $u\{f\}$ and $v\{f\}$.

However, if we are given two substitutions f and g, and for all $x \in X$, a reduction $m_x : f(x) \blacktriangleright g(x)$, then it does not follow that there is a reduction between $u\{f\}$ and $u\{g\}$. This leaves the door open for non-congruent reductions.

Our main examples of reduction monads are given by variants of the lambda calculus. We have collected these examples in Section 4.2.2.

Definition 96. A **morphism of reduction monads** from R to S is given by a pair (f, α) of

1. a monad morphism $f : R \to S$, and
2. a natural transformation $\alpha : \mathsf{Red}(R) \to \mathsf{Red}(S)$

satisfying the following two conditions:

3. α is an R-module morphism between $\mathsf{Red}(R)$ and $f^*\mathsf{Red}(S)$, and
4. the square

$$\begin{array}{ccc} \mathsf{Red}(R) & \xrightarrow{\alpha} & \mathsf{Red}(S) \\ {\scriptstyle \mathsf{red}_R}\downarrow & & \downarrow{\scriptstyle \mathsf{red}_S} \\ R \times R & \xrightarrow[f \times f]{} & S \times S \end{array}$$

commutes in the category of natural transformations.

In Section 4.7 we specify morphisms of reduction monads via a recursion principle.

Intuitively, a morphism (f, α) as above maps terms of R to terms of S via f, and reductions of R to reductions of S via α. Condition 3 states compatibility of the map of reductions with substitution. Condition 4 states preservation of source and target by the map of reductions: a reduction $m : u \blacktriangleright v$ is mapped by α to a reduction $\alpha(m) : f(u) \blacktriangleright f(v)$.

Proposition 97 (Category of reduction monads)**.** *Reduction monads and their morphisms, with the obvious composition and identity, form a category* RedMon*, equipped with a forgetful functor to the category of monads.*

It turns out that reduction monads are the same as monads relative to the free functor from sets to graphs (for the definition of relative monads, see [ACU15, Definition 2.1]):

Theorem 98. *The category of reduction monads is isomorphic to the category of monads relative to the functor mapping a set to its discrete graph.*

Proof. This is obvious after unfolding the definitions. □

4.2.2 Examples of reduction monads

We are interested in reduction monads with underlying monad LC, the monad of syntactic lambda terms specified by the 1-signature $\Sigma_{\mathsf{LC}} = \Theta \times \Theta + \Theta'$.

Example 99 (Lambda calculus with head-β-reduction)**.** Consider the reduction monad $\mathsf{LC}_{\text{head-}\beta}$ given as follows:

1. the underlying monad is LC;
2. $\mathsf{Red}(R)$ is the module $\mathsf{LC}' \times \mathsf{LC}$;
3. $\mathsf{red}_R(X)$ is the morphism $(u, v) \mapsto \Big(\mathsf{app}(\mathsf{abs}(u), v), u\{* := v\}\Big)$.

This reduction monad deserves to be called *the reduction monad of the lambda calculus with head-β-reduction*.

Example 100 (Lambda calculus with head-η-expansion)**.** Consider the reduction monad $\mathsf{LC}_{\text{head-}\eta}$ given as follows:

1. the underlying monad is LC;
2. $\mathsf{Red}(R)$ is the module LC;
3. $\mathsf{red}_R(X)$ is the morphism $t \mapsto \Big(t, \mathsf{abs}(\mathsf{app}(\iota t, *))\Big)$, where $\iota : \mathsf{LC}(X) \to \mathsf{LC}'(X)$ is the natural injection (weakening).

This reduction monad deserves to be called *the reduction monad of the lambda calculus with head-η-expansion*.

Obviously, to get the analogous reduction rule for the η-contraction, it is enough to swap, in the previous example, the components of the output of the morphism red_R.

Given two reduction monads with the same underlying monad on sets, we define the *amalgamation* of the reduction monads as follows:

Definition 101. Given reduction monads R and S with the same underlying monad on sets, we define the reduction monad $R \amalg S$ as follows:

1. the underlying monad on sets is still R (or, equivalently, S);
2. the R-module $\mathsf{Red}(R \amalg S)$ is the coproduct $\mathsf{Red}(R) \amalg \mathsf{Red}(S)$.
3. the module morphism $\mathsf{red}_{R \amalg S}$ is induced by red_R and red_S.

This is the pushout, in the category of reduction monads, of $uR \to R$ and $uR \to S$, with the reduction monad $uR := (\underline{R}, 0, !)$.

Example 102. The reduction monad $\mathsf{LC}_{\text{head-}\beta/\eta} := \mathsf{LC}_{\text{head-}\beta} \amalg \mathsf{LC}_{\text{head-}\eta}$ has, as reductions, β-reductions and η-expansions at the head of lambda terms.

So far we have only considered reductions at the root of a lambda term. The following construction allows us to propagate reductions into subterms.

Definition 103. Let R be a reduction monad over the monad LC on sets. We define the reduction monad R^{cong} as follows:

1. the underlying monad is LC;
2. $\mathsf{Red}(R^{\mathsf{cong}})(X)$ is generated by the following constructions:
   (a) for $m : u \rightsquigarrow v$ in $\mathsf{Red}(R)$, m is also in $\mathsf{Red}(R^{\mathsf{cong}})$
   (b) for $m : u \rightsquigarrow v : \mathsf{LC}(X)$ in $\mathsf{Red}(R^{\mathsf{cong}})$ and $t \in \mathsf{LC}(X)$, we have $\mathsf{app\text{-}cong}_1(m, t) : \mathsf{app}(u, t) \rightsquigarrow \mathsf{app}(v, t)$
   (c) for $m : u \rightsquigarrow v : \mathsf{LC}(X)$ in $\mathsf{Red}(R^{\mathsf{cong}})$ and $t \in \mathsf{LC}(X)$, we have $\mathsf{app\text{-}cong}_2(t, m) : \mathsf{app}(t, u) \rightsquigarrow \mathsf{app}(t, v)$
   (d) for $m : u \rightsquigarrow v : \mathsf{LC}'(X)$ in $\mathsf{Red}(R^{\mathsf{cong}})$ we have $\mathsf{abs\text{-}cong}(m) : \mathsf{abs}(u) \rightsquigarrow \mathsf{abs}(v)$

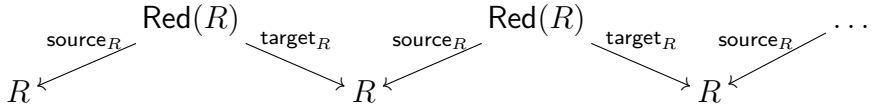
3. $\mathsf{red}_{R^{\mathsf{cong}}}$ is obvious.

Example 104. A reduction in the reduction monad $\mathsf{LC}_{\beta/\eta} := (\mathsf{LC}_{\mathsf{head}\text{-}\beta/\eta})^{\mathsf{cong}}$ is "one step" of β-reduction or η-expansion, anywhere in the source term.

The *closure under identity and composition of reductions* of a reduction monad is defined as follows:

Definition 105. Given a reduction monad R, we define the reduction monad R^* as follows:

1. the underlying monad on sets is still R;
2. the R-module $\mathsf{Red}(R^*)$ is defined as follows. For $n \in \mathbb{N}$ we define the module $\mathsf{Red}(R)^n$ of "n composable reductions", namely as the limit of the diagram



   with n copies of $\mathsf{Red}(R)$ (and hence $n+1$ copies of R). We obtain $n+1$ projections $\pi_i : \mathsf{Red}(R)^n \to R$, and we call $p_n := (\pi_0, \pi_n) : \mathsf{Red}(R)^n \to R \times R$. We set $\mathsf{Red}(R^*) := \coprod_n \mathsf{Red}(R)^n$.
3. the module morphism is $\mathsf{red}_{R^*} := [p_n]_{n\in\mathbb{N}} : \coprod_n \mathsf{Red}(R)^n \to R \times R$ the universal morphism induced by the family $(p_n)_{n\in N}$.

Example 106 (The reduction monad of the lambda calculus)**.** The *reduction monad of the lambda calculus* is defined to be the reduction monad $\mathsf{LC}^*_{\beta/\eta}$.

In Section 4.4 we introduce signatures that allow for the specification of reduction monads. The signature specifying the reduction monad of lambda calculus of Example 104 is given in Example 130.

4.3 Reduction rules

In this section, we define an abstract notion of reduction rule over a signature for monads Σ (Section 4.3.2). We first focus, in Section 4.3.1, on the example of the congruence rule for the application construction in the signature Σ_{LC} for the monad of the

lambda calculus, in order to motivate the definitions. The destiny of a reduction rule over Σ is to be "validated" in a reduction monad equipped with an action of Σ (this is what we will call a reduction Σ-monad in Section 4.3.3). We make this notion of validation precise in Section 4.3.4, as an action of the reduction rule in the reduction Σ-monad. Finally, we give examples of reduction rules in Section 4.3.5.

4.3.1 Example: congruence rule for application

We give some intuitions of the definition of reduction rule with the example of the congruence rule for application, given, e.g., in Selinger's lecture notes [Sel08], as follows:

$$\frac{T \rightsquigarrow T' \qquad U \rightsquigarrow U'}{\mathsf{app}(T,U) \rightsquigarrow \mathsf{app}(T',U')}$$

This rule is parameterized by four *metavariables*: T, T', U, and U'. The conclusion and the hypotheses are given by pairs of terms built out of these metavariables.

We formalize this rule as follows: for any monad R equipped with an application operation $\mathsf{app} : R \times R \to R$, we associate a module of metavariables $\mathcal{V}(R) = R \times R \times R \times R$, one factor for each of the metavariables T, T', U, and U'. Each hypothesis or conclusion is described by a parallel pair of morphisms from $\mathcal{V}(R)$ to R: for example, the conclusion $c_R : \mathcal{V}(R) \to R$ maps a set X and a quadruple (T, T', U, U') to the pair $(\mathsf{app}(T,U), \mathsf{app}(T',U'))$. These assignments are actually functorial in R, and abstracting over R yields our notion of *term-pair* over the Σ-module $\mathcal{V}$, as morphisms from $\mathcal{V}$ to $\Theta \times \Theta$, where Σ is any signature including a single first-order binary operation app (for example, Σ_{LC}). The three term-pairs, one for each hypothesis and one for the conclusion, define the desired reduction rule.

Now, we explain in which sense such a rule can be validated in a reduction monad R: intuitively, it means that for any set X, any quadrupe $(T, T', U, U') \in \mathcal{V}(R)$, any reductions $s : T \blacktriangleright T'$ and $t : U \blacktriangleright U'$, there is a reduction $\mathsf{app\text{-}cong}(s,t) : \mathsf{app}(T,U) \blacktriangleright \mathsf{app}(T',U')$. Of course, this only makes sense if the monad $\underline{R}$ underlying the reduction monad is equipped with an application operation, that is, with an operation of Σ_{app}. We call such a structure a *reduction Σ_{app}-monad*.

4.3.2 Definition of reduction rules

In this subsection, Σ is a signature for monads. We present our notion of *reduction rule over* Σ, from which we build *reduction signatures* in Section 4.4.

We begin with the definition of *term-pair*, alluded to already in Section 4.3.1:

Definition 107. Given a Σ-module $\mathcal{V}$, a **term-pair from** $\mathcal{V}$ is a pair (n, p) of a natural number n and a morphism of Σ-modules $p : \mathcal{V} \to \Theta^{(n)} \times \Theta^{(n)}$.

Many term-pairs are of a particularly simple form, namely a pair of projections, which intuitively picks two among the available metavariables. Because of their ubiquity, we introduce the following notation:

Definition 108. Let $n_1, \ldots, n_p$ be a list of natural numbers. For $i, j \in \{1, \ldots p\}$, we define the pair projection $\pi_{i,j}$ and the projection π_i as the following Σ-module morphisms, for any signature Σ:

$$\begin{array}{rll} \pi_{i,j} : & \Theta^{(n_1)} \times \ldots \Theta^{(n_p)} \to \Theta^{(n_i)} \times \Theta^{(n_j)} & \pi_{i,j,R,X}(T_1, \ldots, T_p) = \quad (T_i, T_j) \\ \pi_i : & \Theta^{(n_1)} \times \ldots \Theta^{(n_p)} \to \Theta^{(n_i)} & \pi_{i,j,R,X}(T_1, \ldots, T_p) = \quad T_i \end{array}$$

Some term-pairs, such as the conclusions of the congruence rules for application and abstraction, are more complicated: intuitively, they are constructed from term constructions applied to metavariables.

Example 109 (term-pair of the conclusion of the congruence for application)**.** The term-pair corresponding to the conclusion $\mathsf{app}(T, U) \rightsquigarrow \mathsf{app}(T', U')$ of congruence for application (Section 4.3.1) is given by $(0, c)$, on the Σ_{LC}-module Θ^4. Here, we have

$$\begin{aligned} &c : \mathcal{V} \to \Theta \times \Theta \\ &c_{R,X}(T, T', U, U') := \Big(\mathsf{app}_{R,X}(T, U), \mathsf{app}_{R,X}(T', U')\Big) \end{aligned}$$

More schematically:

$$c : \Theta^4 \xrightarrow{\mathsf{app}\circ\pi_{1,3}, \mathsf{app}\circ\pi_{2,4}} \Theta \times \Theta$$

We now give our definition of *reduction rule*, making precise the intuition developed in Section 4.3.1.

Definition 110. A **reduction rule** $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$ **over** Σ is given by:

- Metavariables: a Σ-module $\mathcal{V}$ of metavariables, that we sometimes denote by $\mathsf{MVar}_{\mathcal{A}}$;
- Hypotheses: a finite family of term-pairs $(n_i, h_i)_{i\in I}$ from $\mathcal{V}$;
- Conclusion: a term-pair (n, c) from $\mathcal{V}$.

Example 111 (Reduction rule for congruence of application)**.** The reduction rule $\mathcal{A}_{\mathsf{app\text{-}cong}}$ for congruence of application (Section 4.3.1) is defined as follows:

- Metavariables: $\mathcal{V} = \Theta^4$ for the four metavariables T, T', U, and U';
- Hypotheses: Given by two term-pairs $(0, h_1)$ and $(0, h_2)$:

$$h_1 : \Theta^4 \xrightarrow{\pi_{1,2}} \Theta \times \Theta \qquad h_2 : \Theta^4 \xrightarrow{\pi_{3,4}} \Theta \times \Theta$$

- Conclusion: Given by the term-pair $(0, c)$ of Example 109.

More examples of reduction rules are given in Section 4.3.5.

4.3.3 Reduction Σ-monads

As already said, the destiny of a reduction rule is to be validated in a reduction monad R. However, as the hypotheses or the conclusion of the reduction rule may refer to some operations specified by a signature Σ for monads, this reduction monad R must be equipped with an action of Σ, hence the following definition:

Definition 112. Let Σ be a signature for monads. The **category** RedMon^{Σ} **of reduction** Σ**-monads** is defined as the following pullback:

$$\begin{array}{ccc} \mathsf{RedMon}^{\Sigma} & \longrightarrow & \mathsf{RedMon} \\ \downarrow & \lrcorner & \downarrow \\ \mathsf{Mon}^{\Sigma} & \longrightarrow & \mathsf{Mon} \end{array}$$

More concretely,

- a **reduction** Σ**-monad** is a reduction monad R equipped with an **action** ρ of Σ in R, thus inducing a Σ-monad that we denote also by R, or by $\underline{R}$ when we want to be explicit;

- a **morphism of reduction Σ-monads** $R \to S$ is a morphism $f : R \to S$ of reduction monads which commutes with the action of Σ, i.e, whose underlying monad morphism is a Σ-monad morphism.

4.3.4 Action of a reduction rule

Let Σ be a signature for monads. In this section, we introduce the notion of *action of a reduction rule over Σ in a reduction Σ-monad*. Intuitively, such an action is a "map from the hypotheses to the conclusion" of the reduction rule. To make this precise, we need to first take the product of the hypotheses; this product is, more correctly, a *fibred* product.

Definition 113. Let (n, p) be a term-pair from a Σ-module $\mathcal{V}$, and R be a reduction Σ-monad. We denote by $p^*(\mathsf{Red}(R)^{(n)})$ the pullback of $\mathsf{red}_R^{(n)} : \mathsf{Red}(R)^{(n)} \to R^{(n)} \times R^{(n)}$ along $p_R : \mathcal{V}(R) \to R^{(n)} \times R^{(n)}$:

$$\begin{array}{ccc} p^*(\mathsf{Red}(R)^{(n)}) & \longrightarrow & \mathsf{Red}(R)^{(n)} \\ \downarrow & \lrcorner & \downarrow{\scriptstyle \mathsf{red}_R^{(n)}} \\ \mathcal{V}(R) & \xrightarrow[p_R]{} & R^{(n)} \times R^{(n)} \end{array}$$

We denote by $p^*(\mathsf{red}_R^{(n)}) : p^*(\mathsf{Red}(R)^{(n)}) \to \mathcal{V}(R)$ the projection morphism on the left.

Definition 114. Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i\in I}, (n, c))$ be a reduction rule.

The R**-module** $\mathsf{Hyp}_{\mathcal{A}}(R)$ **of hypotheses of** $\mathcal{A}$ is $\prod_{i\in I/\mathcal{V}(R)} h_i^*\mathsf{Red}(R)^{(n_i)}$ the fiber product of all the R-modules $h_i^*\mathsf{Red}(R)^{(n_i)}$ along their projection to $\mathcal{V}(R)$. It thus comes with a projection $\mathsf{hyp}_{\mathcal{A}}(R) : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathcal{V}(R)$

The R**-module** $\mathsf{Con}_{\mathcal{A}}(R)$ **of conclusion of** $\mathcal{A}$ is $c^*\mathsf{Red}(R)^{(n)}$, and comes with a projection $\mathsf{con}_{\mathcal{A}}(R) : \mathsf{Con}_{\mathcal{A}}(R) \to \mathcal{V}(R)$.

Example 115. Let R be a reduction Σ_{LC}-monad. The R-module of conclusion of the congruence reduction rule for application (Example 111) maps a set X to the set of quintuples (T, T', U, U', m) where $(T, T', U, U') \in R^4(X)$ and m is a reduction $m : \mathsf{app}(T, U) \blacktriangleright \mathsf{app}(T', U')$.

The R-module of hypotheses of this reduction rule maps a set X to the set of sextuples (T, T', U, U', m, n) where $(T, T', U, U') \in R^4(X)$, $m : T \blacktriangleright T'$, and $n : U \blacktriangleright U'$.

Definition 116. Let $\mathcal{A}$ be a reduction rule over Σ. An **action of $\mathcal{A}$ in a reduction Σ-monad** R is a morphism between $\mathsf{hyp}_{\mathcal{A}}(R)$ and $\mathsf{con}_{\mathcal{A}}(R)$ in the slice category $\mathsf{Mod}(R)/\mathsf{MVar}_{\mathcal{A}}$, that is, a morphism of R-modules

$$\tau : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathsf{Con}_{\mathcal{A}}(R)$$

making the following diagram commute:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\quad\tau\quad} & \mathsf{Con}_{\mathcal{A}}(R) \\ & \searrow \quad \swarrow & \\ & \mathsf{MVar}_{\mathcal{A}}(R) & \end{array} \tag{4.1}$$

Example 117 (Action of the congruence rule for application)**.** Consider the reduction rule of the congruence for application of Example 111. Let R be a reduction Σ_{LC}-monad R. An action in R is a R-module morphism mapping, for each set X, a sextuple (T, T', U, U', r, s) with $r : T \blacktriangleright T'$ and $s : U \blacktriangleright U'$ to a quintuple (A, A', B, B', m) with $m : \mathsf{app}(A, B) \blacktriangleright \mathsf{app}(A', B')$. The commutation of the triangle (4.1) ensures that $(A, A', B, B') = (T, T', U, U')$.

Alternatively (as justified formally by Lemma 134), an action is a morphism mapping the same sextuple to a reduction $m : \mathsf{app}(T, U) \blacktriangleright \mathsf{app}(T', U')$.

4.3.5 Examples of reduction rules

This section collects a list of motivating examples of reduction rules.

We adopt the following schematic presentation of a reduction rule over a signature Σ:

$$\frac{s_1(T_1, \ldots, T_q) \rightsquigarrow t_1(T_1, \ldots, T_q) \qquad \ldots \qquad s_n(T_1, \ldots, T_q) \rightsquigarrow t_n(T_1, \ldots, T_q)}{s_0(T_1, \ldots, T_q) \rightsquigarrow t_0(T_1, \ldots, T_q)}$$

where s_i and t_i are expressions depending on metavariables T_1, ..., T_q. Each pair (s_i, t_i) defines a term-pair as follows:

$$\begin{aligned} p_i : M_1 \times \cdots \times M_q &\to \Theta^{(m_i)} \times \Theta^{(m_i)} \\ p_{i,R,X}(T_1, \ldots, T_q) &:= (s_i(T_1, \ldots, T_q), t_i(T_1, \ldots, T_q)) \end{aligned} \tag{4.2}$$

$$\frac{}{T \rightsquigarrow T}\ \text{Refl} \qquad \frac{T \rightsquigarrow U \qquad U \rightsquigarrow W}{T \rightsquigarrow W}\ \text{Trans} \qquad \frac{T \rightsquigarrow U}{\mathsf{abs}(T) \rightsquigarrow \mathsf{abs}(U)}\ \text{abs-Cong}$$

$$\frac{}{\mathsf{app}(\mathsf{abs}(T), U) \rightsquigarrow T\{* := U\}}\ \beta\text{-Red} \qquad \frac{}{T \rightsquigarrow \mathsf{abs}(\iota(T))}\ \eta\text{-Exp} \qquad \frac{}{\mathsf{fix}(T) \rightsquigarrow T\{* := \mathsf{fix}(T)\}}\ \text{fix-Exp}$$

Figure 4.1: Examples of reduction rules.

where the Σ-modules $M_1, \ldots, M_q$, and the natural numbers $m_0, \ldots, m_n$ are inferred for Equation (4.2) to be well defined for all $i \in \{0, \ldots, n\}$.

The induced reduction rule is:

- Metavariables: the Σ-module of metavariables is $\mathcal{V} = M_1 \times \cdots \times M_q$;
- Hypotheses: the hypotheses are the term-pairs $(m_i, p_i)_{i \in \{1,\ldots,n\}}$;
- Conclusion: the conclusion is the term-pair (m_0, p_0).

Typically, $M_i = \Theta^{(n_i)}$ for some natural number n_i, as in the examples that we consider in this section. In practice, there are several choices for building the reduction rule out of such a schematic presentation, depending on the order in which the metavariables are picked. This order is irrelevant: the different possible versions of reduction rules are all equivalent, in the sense that taking one or the other as part of a reduction signature yields isomorphic category of models.

For the rest of this section, we assume that we have fixed a signature for monads Σ. Figure 4.1 shows some notable examples of reduction rules. In order, they are: reflexivity, transitivity, congrucence for abs, β-reduction, η-expansion, and expansion of the fixpoint operator.

For the example of the fixpoint operator (rule fix-Exp), we consider the 1-signature Σ_{fix}, as described in Section 3.4.4. (without enforcing the fixpoint equation, which is replaced here by the reduction rule under consideration). A Σ_{fix}-monad is a monad R equipped with an R-module morphism $\mathsf{fix} : R' \to R$.

Figure 4.2 lists the modules and term pairs for hypothesis and conclusion of each of these reduction rules. There, $\pi_{i,j}$ designates the pair projection described in Definition 108.

Rule	Signature	Metavariables	Hypothesis	Conclusion
Refl	any	Θ		$(0, (\mathtt{id}, \mathtt{id}))$
Trans	any	$\Theta \times \Theta \times \Theta$ for (T, U, W)	$(0, \pi_{1,2})$, $(0, \pi_{2,3})$	$(0, \pi_{1,3})$
abs-Cong	Σ_{LC}	$\Theta' \times \Theta'$ for (T, U)	$(1, \mathtt{id})$	$(0, \mathsf{abs} \times \mathsf{abs})$
β-Red	Σ_{LC}	$\Theta' \times \Theta$ for (T, U)		$(0, c_{\beta\text{-Red}})$
η-Exp	Σ_{LC}	Θ		$(0, (\mathtt{id}, \mathsf{abs} \circ \iota))$
fix-Exp	Σ_{fix}	Θ'		$(0, c_{\mathsf{fix}\text{-Exp}})$

$$c_{\beta\text{-Red},R,X}(T, U) = (\mathsf{app}(\mathsf{abs}(T), U), T\{* := U\})$$
$$c_{\mathsf{fix}\text{-Exp},R,X}(T) = (\mathsf{fix}(T), T\{* := \mathsf{fix}(T)\})$$

Figure 4.2: Modules and term pairs relative to the reduction rules of Figure 4.1.

4.4 Signatures for reduction monads and Initiality

In this section, we define the notion of *reduction signature*, consisting of a signature for monads Σ and a family of reduction rules over Σ (see Section 4.4.1). As usual, we assign to each such signature a *category of models*. We call a reduction signature *effective* if the associated category of models has an initial object. Our main result, Theorem 128 (see Section 4.4.3), states that a reduction signature is effective as soon as its underlying signature for monads is effective.

4.4.1 Signatures and their models

We define here *reduction signatures* and their *models*.

Definition 118. A **reduction signature** is a pair $(\Sigma, \mathfrak{R})$ of a signature Σ for monads and a family $\mathfrak{R}$ of reduction rules over Σ.

Definition 119. Given a reduction monad R and a reduction signature $\mathcal{S} = (\Sigma, \mathfrak{R})$, an **action of $\mathcal{S}$ in** R consists of an action of Σ in its underlying monad $\underline{R}$ and an action of each reduction rule of $\mathfrak{R}$ in R.

Definition 120. Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a reduction signature. A **model of** $\mathcal{S}$ is a reduction monad equipped with an action of $\mathcal{S}$, or equivalently, a reduction Σ-monad equipped with an action of each reduction rule of $\mathfrak{R}$.

Remark 121 (Continuation of Remark 94)**.** Just as our reduction monads are "proof-relevant" (cf. Remark 94), our notion of reduction signature allows for the specification

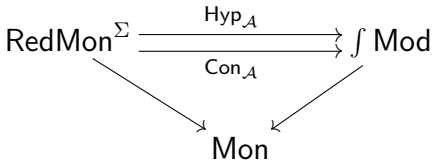
of multiple reductions between terms. As a trivial example , duplicating the β-rule in the signature $\mathcal{S}_{\mathsf{LC}}$ yields two distinct β-reductions in the initial model.

4.4.2 The functors $\mathsf{Hyp}_{\mathcal{A}}$ and $\mathsf{Con}_{\mathcal{A}}$

The definition of morphism between models of a reduction signature relies on the functoriality of the assignments $R \mapsto \mathsf{Hyp}_{\mathcal{A}}(R)$ and $R \mapsto \mathsf{Con}_{\mathcal{A}}(R)$, for a given reduction rule $\mathcal{A}$ on a signature Σ for monads.

Definition 122. Let Σ be a signature for monads, and $\mathcal{A}$ be a reduction rule over Σ. Definition 114 assigns to each Σ-monad R the R-modules $\mathsf{Hyp}_{\mathcal{A}}(R)$ and $\mathsf{Con}_{\mathcal{A}}(R)$. These assignments extend to functors $\mathsf{Hyp}_{\mathcal{A}}, \mathsf{Con}_{\mathcal{A}} : \mathsf{RedMon}^{\Sigma} \to \mathsf{Mon}^{\Sigma}$.

Proposition 123. *Given the same data, the functors* $\mathsf{Hyp}_{\mathcal{A}}$ *and* $\mathsf{Con}_{\mathcal{A}}$ *commute with the forgetful functors to* Mon*:*



4.4.3 The main result

For a reduction signature $\mathcal{S}$, we define here the notion of $\mathcal{S}$-model morphism, inducing a **category of models of** $\mathcal{S}$. We then state our main result, Theorem 128, which gives a sufficient condition for $\mathcal{S}$ to admit an initial model.

Definition 124. Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a reduction signature. A morphism between models R and T of $\mathcal{S}$ is a morphism f of reduction Σ-monads commuting with the action of any reduction rule, in the sense that for any reduction rule $\mathcal{A} \in \mathfrak{R}$, the following diagram of natural transformations commutes:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \longrightarrow & \mathsf{Con}_{\mathcal{A}}(R) \\ {\scriptstyle \mathsf{Hyp}_{\mathcal{A}}(f)}\downarrow & & \downarrow{\scriptstyle \mathsf{Con}_{\mathcal{A}}(f)} \\ \mathsf{Hyp}_{\mathcal{A}}(T) & \longrightarrow & \mathsf{Con}_{\mathcal{A}}(T) \end{array}$$

Example 125 (Example 117 continued)**.** Consider the reduction signature consisting of the signature Σ_{app} of a binary operation app and a single reduction rule of congruence for application (Example 111).

Let R and T be models for this signature: they are reduction Σ_{app}-monads equipped with an action ρ and τ, in the alternative sense of Example 117. A Σ_{app}-monad morphism (f, α) between R and T is a model morphism if, for any set X, any sextuple (A, A', B, B', m, n) where $(A, A', B, B') \in R^4(X)$, $m : A \blacktriangleright A'$, and $n : B \blacktriangleright B'$, the reduction $\rho(A, A', B, B') : \mathsf{app}(A, B) \blacktriangleright \mathsf{app}(A', B')$ is mapped to the reduction $\tau(f(A), f(A'), f(B), f(B'))$ by $\alpha : \mathsf{Red}(R) \to \mathsf{Red}(T)$.

Proposition 126. *Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a reduction signature. Models of $\mathcal{S}$ and their morphisms, with the obvious composition and identity, define a category that we denote by $\mathsf{RedMon}^{\mathcal{S}}$, equipped with a forgetful functor to RedMon^{Σ}.*

Definition 127. A reduction signature $\mathcal{S}$ is said to be **effective** if its category of models $\mathsf{RedMon}^{\mathcal{S}}$ has an initial object, denoted $\widehat{\mathcal{S}}$. In this case, we say that $\widehat{\mathcal{S}}$ is **generated (or specified) by** $\mathcal{S}$.

We now have all the ingredients required to state our main result:

Theorem 128. *Let $(\Sigma, \mathfrak{R})$ be a reduction signature. If Σ is effective, then so is $(\Sigma, \mathfrak{R})$.*

The proof of this theorem is given in Section 4.5.

Theorem 83 entails the following corollary:

Corollary 129. *Let $(\Sigma, \mathfrak{R})$ be a reduction signature. If Σ is algebraic (in the sense of Definition 82), then $(\Sigma, \mathfrak{R})$ is effective.*

All the examples of reduction signatures considered here satisfy the algebraicity condition of Corollary 129.

Example 130 (Reduction signature for Example 104)**.** The reduction monad of Example 104 is generated by the reduction signature $\mathcal{S}_{\mathsf{LC}}$ that is given by the signature Σ_{LC} together with the following reduction rules (see Section 4.3.5):

- the reduction rule for β-reduction;
- the reduction rule for η-expansion;
- the congruence rule for abstraction;

- two unary congruence rules for application:

$$\frac{T \rightsquigarrow T'}{\mathsf{app}(T,U) \rightsquigarrow \mathsf{app}(T',U)} \qquad \frac{U \rightsquigarrow U'}{\mathsf{app}(T,U') \rightsquigarrow \mathsf{app}(T,U')}$$

Example 131 (Reduction signature of lambda calculus with a fixpoint operator)**.** The signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ specifying the reduction monad $\mathsf{LC}_{\mathsf{fix}}$ of the lambda calculus with a fixpoint operator extends the signature $\mathcal{S}_{\mathsf{LC}}$ of Example 130 with:

- a new operation $\mathsf{fix} : \Theta' \to \Theta$ (thus extending the signature for monads Σ_{LC});
- the reduction rule for the fixpoint reduction (cf. Section 4.3.5);
- a congruence rule for fix:

$$\frac{T \rightsquigarrow T'}{\mathsf{fix}(T) \rightsquigarrow \mathsf{fix}(T')}$$

4.5 Proof of Theorem 128

This section details the proof of Theorem 128.

Let $\mathcal{S} = (\Sigma, (\mathcal{A}_i)_{i\in I}))$ be a reduction signature. We denote by $\mathcal{U}^{\Sigma}$ the forgetful functor from the category of reduction Σ-monads to the category of Σ-monads.

In Section 4.5.1, we first reduce to the case of reduction rules $(\mathcal{V}, (n_j, h_j)_{j\in J}, (n, c))$ for which $n = 0$, that we call *normalized*. Then, in Section 4.5.2, we give an alternative definition of the category of models that we make use of in the proof of effectivity, in Section 4.5.3.

4.5.1 Normalizing reduction rules

Definition 132. A reduction rule $(\mathcal{V}, (n_j, h_j)_{j\in J}, (n, c))$ is said **normalized** if $n = 0$.

Lemma 133. *Let $\mathcal{A} = (\mathcal{V}, (n_j, h_j)_{j\in J}, (n, c))$ be a reduction rule over Σ. Then there exists a normalized reduction rule $\mathcal{A}'$ over Σ such that the induced notion of action is equivalent, in the sense that:*

- *given a reduction Σ-monad R, there is a bijection between actions of $\mathcal{A}$ in R and actions of $\mathcal{A}'$ in R;*

- *a morphism between reduction Σ-monads equipped with an action of $\mathcal{A}$ preserves the action (in the sense of Definition 124) if and only if it preserves the corresponding action of $\mathcal{A}'$ through the bijection.*

Before tackling the proof, we give an alternative definition of action and model morphism:

Lemma 134. *Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i\in I}, (n, c))$ be a reduction rule over Σ. By universal property of the pullback $\mathsf{Con}_{\mathcal{A}}(R) = c^*\mathsf{Red}(R)^{(n)}$, an action can be alternatively be defined as an R-module morphism $\sigma : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathsf{Red}(R)^{(n)}$ making the following diagram commute*

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\sigma} & \mathsf{Red}(R)^{(n)} \\ \downarrow & & \downarrow{\scriptstyle \mathsf{red}_R{}^{(n)}} \\ \mathcal{V}(R) & \xrightarrow[c]{} & R^{(n)} \times R^{(n)} \end{array} \tag{4.3}$$

Lemma 135. *Using this alternative definition of action, a morphism between models R and T of a reduction signature $\mathcal{S} = (\Sigma, \mathfrak{R})$ is a morphism f of reduction Σ-monads making the following diagram commutes, for any reduction rule $\mathcal{A} = (\mathcal{V}, (n_i, l_i, r_i)_{i\in I}, (n, l, r))$ of $\mathfrak{R}$:*

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \longrightarrow & \mathsf{Red}(R)^{(n)} \\ {\scriptstyle \mathsf{Hyp}_{\mathcal{A}}(f)}\downarrow & & \downarrow{\scriptstyle \mathsf{Red}(f)^{(n)}} \\ \mathsf{Hyp}_{\mathcal{A}}(T) & \longrightarrow & \mathsf{Red}(T)^{(n)} \end{array}$$

We now prove Lemma 133 using these alternative definitions:

Proof of Lemma 133. The reduction rule $\mathcal{A}' = (\mathcal{V}', (n'_j, h'_j)_{j\in J}, (0, c'))$ is defined as follows:

- Metavariables: $\mathcal{V}' = \mathcal{V} \times \Theta^n$
- Hypotheses: For each $j \in J$, $h'_j : \mathcal{V}' \to \Theta^{(n_j)} \times \Theta^{(n_j)}$ is defined as the composition of $\pi_1 : \mathcal{V} \times \Theta^n \to \mathcal{V}$ with $h_j : \mathcal{V} \to \Theta^{(n_j)} \times \Theta^{(n_j)}$.
- Conclusion: The morphism $c' : \mathcal{V} \times \Theta^m \to \Theta \times \Theta$ is the m^{th} uncurrying (see Definition 61) of $c : \mathcal{V} \to \Theta^{(m)} \times \Theta^{(m)}$.

Now, consider an action for the reduction rule $\mathcal{A}$ in a reduction Σ-monad R: it is an R-module morphism $\tau : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathsf{Red}(R)^{(n)}$ such that the following square commutes:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\tau} & \mathsf{Red}(R)^{(n)} \\ \downarrow & & \downarrow{\scriptstyle \mathsf{red}_R^{(n)}} \\ \mathcal{V}(R) & \xrightarrow[c]{} & R^{(n)} \times R^{(n)} \end{array}$$

Equivalently, through the adjunction mentioned above, it is given by an R-module morphism $\tau^* : \mathsf{Hyp}_R \times R^m \to M$ such that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) \times R^n & \xrightarrow{\tau^*} & \mathsf{Red}(R) \\ \downarrow & & \downarrow{\scriptstyle \mathsf{red}_R} \\ \mathcal{V}(R) \times R^n & \xrightarrow[c^*]{} & R \times R \end{array}$$

This is exactly the definition of an action of $\mathcal{A}'$. It is then straightforward to check that one action is preserved by a reduction monad morphism if and only if the other one is. □

Corollary 136. *For each reduction signature, there exists a reduction signature yielding an isomorphic category of models and whose underlying reduction rules are all normalized.*

Proof. Just replace each reduction rule with the one given by Lemma 133. □

Thanks to this lemma, we assume in the following that all the reduction rules of the given signature $\mathcal{S}$ are normalized.

4.5.2 Models as vertical algebras

In this section, we give an alternative definition for the category of models of $\mathcal{S}$ that is convenient in the proof of effectivity.

First we rephrase the notion of action of a reduction rule as an algebra structure for a suitably chosen endofunctor. Indeed, an action of a normalized reduction rule $\mathcal{A} = (\mathcal{V}, (n_j, h_j)_{j \in J}, (0, c))$ in a reduction Σ-monad R is given by a R-module morphism

$\tau : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathsf{Red}(R)$ such that the following square commutes:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\tau} & \mathsf{Red}(R) \\ \downarrow & & \downarrow{\scriptstyle \mathsf{red}_R} \\ \mathcal{V}(R) & \xrightarrow[p]{} & R \times R \end{array}$$

We can rephrase this commutation by stating that this morphism τ is a morphism in the slice category $\mathsf{Mod}(\underline{R})/\underline{R}^2$ from an object that we denote by $F_{\mathcal{A}|\underline{R}}(\mathsf{Red}(R), \mathsf{red}_R)$, to $(\mathsf{Red}(R), \mathsf{red}_R)$. Actually, the domain is functorial in its argument, and thus the action τ can be thought of as an algebra structure on $(\mathsf{Red}(R), \mathsf{red}_R)$:

Lemma 137. *Given any Σ-monad R, the assignment $(M, p : M \to R \times R) \mapsto F_{\mathcal{A}|R}(M, c)$ yields an endofunctor $F_{\mathcal{A}|R}$ on $\mathsf{Mod}(R)/R^2$. An action of $\mathcal{A}$ in a reduction Σ-monad R is exactly the same as an algebra structure for this endofunctor on $(\mathsf{Red}(R), \mathsf{red}_R) \in \mathsf{Mod}(R)/R^2$.*

Furthermore, the assignment $R \mapsto F_{\mathcal{A}|\underline{R}}(\mathsf{Red}(R), \mathsf{red}_R)$ yields an endofunctor $F_{\mathcal{A}}$ on the category of reduction Σ-monads. This functor preserves the underlying Σ-monad, in the sense that $\mathcal{U}^{\Sigma} \cdot F_{\mathcal{A}} = \mathcal{U}^{\Sigma}$.

Proof. This is a consequence of the functoriality of $\mathsf{Hyp}_{\mathcal{A}}$, as noticed in Section 4.4.2. □

Now, we give our alternative definition of the category of models:

Proposition 138. *Let $F_{\mathcal{S}} : \mathsf{RedMon}^{\Sigma} \to \mathsf{RedMon}^{\Sigma}$ be the coproduct $\coprod_i F_{\mathcal{A}_i}$. Then, the category of models of $\mathcal{S}$ is isomorphic to the **category of vertical algebras** of $F_{\mathcal{S}}$ defined as follows:*

- *an object is an algebra $r : F_{\mathcal{S}}(R) \to R$ such that r is mapped to the identity by $\mathcal{U}^{\Sigma}$*
- *morphisms are the usual $F_{\mathcal{S}}$-algebra morphisms.*

We adopt this definition in the following. We show now a property of the category of models that will prove useful in the proof of effectivity:

Lemma 139. *The forgetful functor from the category of models of $\mathcal{S}$ to the category of Σ-monads is a fibration.*

The proof relies on some additional lemmas, in particular the following one, that we will specialize by taking $p = \mathcal{U}^\Sigma$ (requiring to show that $\mathcal{U}^\Sigma : \mathsf{RedMon}^\Sigma \to \mathsf{Mon}^\Sigma$ is a fibration) and $F = F_\mathcal{S}$:

Lemma 140. *Let $p : E \to B$ be a fibration and F and endofunctor on E satisfying $p \cdot F = p$. Then the category of vertical algebras of F is fibered over the category of B.*

Proof. Let $r : F(R) \to R$ be an algebra over $X \in B$. Let $a : Y \to X$ be a morphism in B. Let $\overline{a} : a^*R \to R$ the associated cartesian morphism in E. We define the pullback of r along $\overline{a}$ as follows: the base object is a^*R, and the algebra structure $\rho : F(a^*R) \to R$ is given by the unique morphism which factors $F(a^*R) \xrightarrow{F(\overline{a})} F(R) \xrightarrow{r} R$ through the cartesian morphism $\overline{a} : a^*R \to R$. Thus, the square

$$\begin{array}{ccc} F(a^*R) & \xrightarrow{F(\overline{a})} & F(R) \\ {\scriptstyle\rho}\downarrow & & \downarrow{\scriptstyle r} \\ a^*R & \xrightarrow[\overline{a}]{} & R \end{array}$$

commutes, so $\overline{a}$ is a morphism of algebras between ρ and r. Next, we prove that it is a cartesian morphism: let $s : F(S) \to S$ be a vertical algebra over an object Z of B, and $v : s \to r$ be a morphism of algebras such that there exists $b : Z \to Y$ such that $p(v) = Z \xrightarrow{b} Y \xrightarrow{a} X$. We need to show that there exists a unique algebra morphism $w : s \to \rho$ such that $v = \overline{a} \circ w$ and $p(w) = b$. Uniqueness follows from the fact that $\overline{a}$ is cartesian for the fibration $p : E \to B$. Moreover, as $\overline{a}$ is cartesian, we get a morphism $w : S \to a^*R$. We turn it into an algebra morphism by showing that the following square commutes:

$$\begin{array}{ccc} F(S) & \xrightarrow{F(w)} & F(a^*R) \\ {\scriptstyle s}\downarrow & & \downarrow{\scriptstyle\rho} \\ S & \xrightarrow[w]{} & a^*R \end{array}$$

As $\overline{a}$ is cartesian and both w and $F(w)$ are sent to b by p, it is enough to show equalities of both morphisms after postcomposing with $\overline{a}$. The fact that v is an algebra morphism allows to conclude. □

We want to apply this lemma for proving Lemma 139. We thus need to show that $\mathcal{U}^\Sigma : \mathsf{RedMon}^\Sigma \to \mathsf{Mon}^\Sigma$ is a fibration:

Lemma 141. *The forgetful functors* $\mathsf{RedMon} \to \mathsf{Mon}$ *and* $\mathcal{U}^\Sigma : \mathsf{RedMon}^\Sigma \to \mathsf{Mon}^\Sigma$ *are fibrations.*

Proof. We have the two following pullbacks:

$$\begin{array}{ccccc} \mathsf{RedMon}^\Sigma & \longrightarrow & \mathsf{RedMon} & \longrightarrow & V(\int \mathsf{Mod}) \\ \downarrow{\scriptstyle \mathcal{U}^\Sigma} & & \downarrow & & \downarrow{\scriptstyle \mathrm{codom}} \\ \mathsf{Mon}^\Sigma & \longrightarrow & \mathsf{Mon} & \xrightarrow[\Theta\times\Theta]{} & \int \mathsf{Mod} \end{array}$$

where $V(\int \mathsf{Mod})$ is the full subcategory of arrows of $\int \mathsf{Mod}$ which are vertical (that is, they are mapped to the identity monad morphism by the functor from $\int \mathsf{Mod}$ to Mon), and codom maps such an arrow to its codomain. By Propositions 4 and 8, the category $\int \mathsf{Mod}$ has fibred finite limits TODO maybe formulate this in Chapter 2 ?, so that codom is a fibration ([See00, Exercice 9.4.2 (i)]).

Now, Proposition 8.1.15 of [Bor94] states that a pullback of a fibration is a fibration. Thus, the middle functor $\mathsf{RedMon} \to \mathsf{Mon}$ is a fibration, and then, $\mathcal{U}^\Sigma : \mathsf{RedMon}^\Sigma \to \mathsf{Mon}^\Sigma$ also is. □

Finally, gathering all these lemmas yields a proof that the category of models of $\mathcal{S}$ is indeed fibered over the category of Σ-monads:

Proof of Lemma 139. Apply Lemma 140 with the fibration $p = \mathcal{U}^\Sigma$ (Lemma 141) and $F = F_{\mathcal{S}}$. □

4.5.3 Effectivity

In this section, we prove that $\mathcal{S}$ has an initial model, provided that there exists an initial Σ-monad. The category of models of $\mathcal{S}$ is fibered over the category of Σ-monads. A promising candidate for the initial model is the initial object, if it exists, in the fiber category over the initial Σ-monad:

Lemma 142. *Let* $p : E \to B$ *be a fibration,* b_0 *be an initial object in* B *and* e_0 *be an object over* b_0 *that is initial in the fiber category over* b_0*. Then* e_0 *is initial in* E*.*

In the following, we thus construct the initial object in a fiber category over a given Σ-monad R. This fiber category can be characterized as a category of algebras:

Lemma 143. *The fiber category over a given Σ-monad R through the fibration from models of $\mathcal{S}$ (Lemma 139) is the category of algebras of the endofunctor $F_{\mathcal{S}|R} = \coprod_i F_{\mathcal{A}_i|R}$ on the slice category* $\mathrm{Mod}(R)/R^2$.

Thus, our task is to construct the initial algebra of some specific endofunctor. Adámek's theorem [Adá74] provides a sufficient condition for the existence of an initial algebra:

Lemma 144 (Adámek)**.** *Let F be a finitary endofunctor on a cocomplete category C. Then the category of algebras of F has an initial object.*

This initial object can be computed as a colimit of a chain, but we do not rely here on the exact underlying construction.

The first requirement to apply this lemma is that the base category is cocomplete, and this is indeed the case:

Lemma 145. *The category* $\mathrm{Mod}(R)/R^2$ *is cocomplete for any monad R.*

Proof. The category of modules $\mathrm{Mod}(R)$ over a given monad R is cocomplete by Proposition 4, so any of its slice categories is, by the dual of [ML98, Exercise V.1.1], in particular $\mathrm{Mod}(R)/R^2$. □

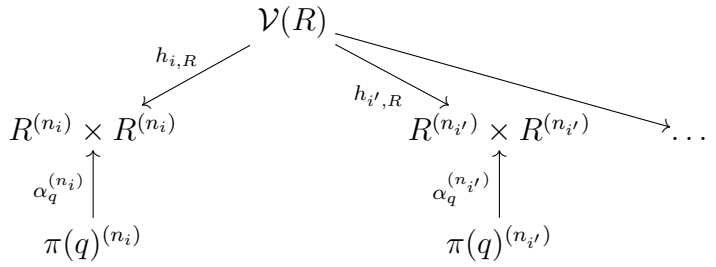
Let us show that the finitary requirement of Lemma 144 is also satisfied for the case of a signature with a single reduction rule:

Lemma 146. *Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i\in I}, (0, c))$ be a normalized reduction rule over Σ, and R be a Σ-monad. Then, $F_{\mathcal{A}|R}$ is finitary.*

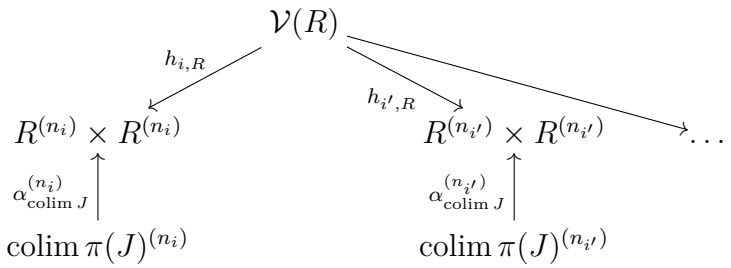
Proof. In this proof, we denote by F the endofunctor $F_{\mathcal{A}|R}$ on $\mathrm{Mod}(R)/R^2$, by $\pi : D/d \to D$ the projection for a general slice category, and by $\alpha : \pi \to d$ the natural transformation induced by the underlying morphism of a slice object: $\alpha_p : \pi(p) \to d$. Note that π creates colimits, by the dual of [ML98, Exercise V.1.1].

Given a filtered diagram we want to show that the image by F of the colimiting cocone is colimiting. As π creates colimits, this is enough to show that the image by $\pi \cdot F$ of the colimiting cocone is colimiting. Thus, it is enough to prove that $\pi \cdot F : \mathrm{Mod}(R)/R^2 \to \mathrm{Mod}(R)$ is finitary.

Given any $q \in \mathsf{Mod}(R)/R^2$ the module $\pi(F(q))$ is $\mathsf{Hyp}_{\mathcal{A}}(R)$, which can be computed as the limit of the following finite diagram:



Let $J : \mathsf{C} \to \mathsf{Mod}(R)/R^2$ be a filtered diagram. As π preserves colimits (since it creates them), $\pi(F(\operatorname{colim} J))$ is the limit of the following diagram:



Now, as limits and colimits are computed pointwise in the category of modules, and that finite limits commute with filtered colimits in Set ([ML98, Section IX.2, Theorem 1]), we have that $\pi(F(\operatorname{colim} J))$, as the limit of such a diagram, is canonically isomorphic to the colimit of $\pi \cdot F \cdot J$.

□

Now, consider a signature $\mathcal{S}$ with a family of reduction rules $(\mathcal{A}_i)_i$. The functor that we are concerned with is $F_{\mathcal{S}|R} = \coprod_i F_{\mathcal{A}_i|R}$, for a given Σ-monad R:

Lemma 147. *For any Σ-monad R, the functor $F_{\mathcal{S}|R} = \coprod_i F_{\mathcal{A}_i|R}$ is finitary.*

Proof. This is a coproduct of finitary functors (by Lemma 146), and so is finitary as colimits commute with colimits, by [ML98, Equation V.2.2]. □

Now we are ready to tackle the proof of our main result:

Proof of Theorem 128. We assume that Σ is effective; let R be the initial Σ-monad. We want to show that $\mathcal{S}$ has an initial model. We apply Lemma 142 with p the fibration from

models to Σ-monads (Lemma 139): we are left with providing an initial object in the fiber category over R. By Lemma 143, this boils down to constructing an initial algebra for the endofunctor $F_{\mathcal{S}|R}$ on the category $\mathsf{Mod}(R)/R^2$. We apply Lemma 144: $\mathsf{Mod}(R)/R^2$ is indeed cocomplete by Lemma 145, and $F_{\mathcal{S}|R}$ is finitary by Lemma 147). □

4.6 Example: Lambda calculus with explicit substitutions

In this section, we give a signature specifying the reduction monad of the lambda calculus with explicit substitutions as described in [Kes09]. This example is particular because it involves operations subject to some equations, and on top of this syntax with equations, a "graph of reductions".

In Section 4.6.1, we present the underlying signature for monads, and in Section 4.6.2, we list the reduction rules of the signature.

4.6.1 Signature for the monad of the lambda calculus with explicit substitutions

We give here the signature for the monad of the lambda calculus with explicit substitutions: first the syntactic operations, and then the equation that the explicit substitution must satisfy.

Operations

The lambda calculus with explicit substitutions extends the lambda calculus with an explicit unary substitution operator $t[x/u]$. Here, the variable x is assumed not to occur freely in u. In our setting, it is specified as an operation $\mathsf{esubst}_X : \mathsf{LC}'(X) \times \mathsf{LC}(X) \to \mathsf{LC}(X)$. It is thus specified by the signature $\Theta' \times \Theta$. An action of this signature in a monad R yields a map $\mathsf{esubst}_X : R(X + \{*\}) \times R(X) \to R(X)$ for each set X, where $\mathsf{esubst}_X(t, u)$ is meant to model the explicit substitution $t[*/u]$.

Definition 148. The signature $\Upsilon_{\mathsf{LC_{ex}}}$ for the monad of the lambda calculus with explicit substitutions without equations is the coproduct of $\Theta' \times \Theta$ and Σ_{LC}.

Equation

The syntax of lambda calculus with explicit substitutions of [Kes09] is subject to the equation (see [Kes09, Figure 1, “Equations”])

$$t[x/u][y/v] = t[y/v][x/u] \quad \text{if } y \notin \mathsf{fv}(u) \ . \tag{4.4}$$

We rephrase it as an equality between two parallel $\Upsilon_{\mathsf{LC_{ex}}}$-module morphisms from $\Theta'' \times \Theta \times \Theta$, modelling the metavariables t, u, and v, to Θ:

$$\begin{array}{l} \Theta'' \times \Theta \times \Theta \xrightarrow{\Theta'' \times \iota \times \Theta} \Theta'' \times \Theta' \times \Theta \xrightarrow{\mathsf{esubst} \times \Theta} \Theta' \times \Theta \xrightarrow{\mathsf{esubst}} \Theta \\ \Theta'' \times \Theta \times \Theta \xrightarrow[\Theta'' \times \Theta \times \iota]{} \Theta'' \times \Theta \times \Theta' \xrightarrow[\mathsf{esubst}' \circ \pi_{1,3}, \pi_2]{} \Theta' \times \Theta \xrightarrow[\mathsf{esubst}]{} \Theta \end{array} \tag{4.5}$$

Here, ι denotes the inclusion $\Theta \to \Theta'$.

Now we are ready to define the signature of the lambda calculus monad with explicit substitutions:

Definition 149. The signature $\Sigma_{\mathsf{LC_{ex}}}$ of the lambda calculus monad with explicit substitutions consists of $\Upsilon_{\mathsf{LC_{ex}}}$ and the single $\Sigma_{\mathsf{LC_{ex}}}$-equation stating the equality between the two morphisms of Equation 4.5.

Lemma 150. *The signature $\Sigma_{\mathsf{LC_{ex}}}$ for monads is effective*

Proof. This is a direct consequence of Corollary 129. □

4.6.2 Reduction rules for lambda calculus with explicit substitutions

The reduction signature for the lambda calculus with explicit substitutions consists of two components: the first one is the signature for monads $\Sigma_{\mathsf{LC_{ex}}}$ of Definition 149; the second one is the list of reduction rules that we enumerate here, taken from [Kes09, Figure 1, “Rules”]. Except for congruence, none of them involve hypotheses.

First, let us state the congruence rules (that are implicit in [Kes09]):

$$\frac{T \rightsquigarrow T'}{\mathsf{app}(T,U) \rightsquigarrow \mathsf{app}(T',U)} \quad \frac{U \rightsquigarrow U'}{\mathsf{app}(T,U) \rightsquigarrow \mathsf{app}(T,U')} \quad \frac{T \rightsquigarrow T'}{\mathsf{abs}(T) \rightsquigarrow \mathsf{abs}(T')}$$

$$\frac{T \rightsquigarrow T'}{\mathsf{esubst}(T,U) \rightsquigarrow \mathsf{esubst}(T',U)} \quad \frac{U \rightsquigarrow U'}{\mathsf{esubst}(T,U) \rightsquigarrow \mathsf{esubst}(T,U')}$$

$$\frac{}{(\lambda x.t)u \rightsquigarrow t[x/u]}\ \beta\text{-red} \qquad \frac{x \notin \mathsf{fv}(t)}{t[x/u] \rightsquigarrow t}\ \mathsf{Gc} \qquad \frac{}{x[x/u] \rightsquigarrow u}\ \mathsf{var}[]$$

$$\frac{}{(t\,u)[x/v] \rightsquigarrow t[x/v]\,u[x/v]}\ \mathsf{app}[] \qquad \frac{}{(\lambda y.t)[x/v] \rightsquigarrow \lambda y.t[x/v]}\ \mathsf{abs}[]$$

$$\frac{y \in \mathsf{fv}(u)}{t[x/u][y/v] \rightsquigarrow t[y/v][x/u[y/v]]}\ [][]$$

Figure 4.3: Reduction rules of lambda calculus with explicit substitutions.

$$\frac{}{\mathsf{app}(\mathsf{abs}(T), U) \rightsquigarrow \mathsf{esubst}(T, U)}\ \beta\text{-red} \qquad \frac{}{\mathsf{esubst}(\iota(T), U) \rightsquigarrow T}\ \mathsf{Gc}$$

$$\frac{}{\mathsf{esubst}(\mathsf{app}(T, U), V) \rightsquigarrow \mathsf{app}(\mathsf{esubst}(T, V), \mathsf{esubst}(U, V))}\ \mathsf{app\text{-}esubst}$$

$$\frac{}{\mathsf{esubst}(*, T) \rightsquigarrow T}\ \mathsf{var\text{-}esubst} \qquad \frac{}{\mathsf{esubst}(\mathsf{abs}(T), V) \rightsquigarrow \mathsf{abs}(\mathsf{esubst}'(T, \iota(V)))}\ \mathsf{abs\text{-}esubst}$$

$$\frac{}{\mathsf{esubst}(\mathsf{esubst}(T, \kappa(U)), V) \rightsquigarrow \mathsf{esubst}(\mathsf{esubst}'(T, \iota(V)), \mathsf{esubst}(\kappa(U), V))}\ \mathsf{esubst\text{-}esubst}$$

$$\kappa : \Theta_* \to \Theta'$$

where Θ_* is the 1-hole context $\Sigma_{\mathsf{LC_{ex}}}$-submodule of Θ' (Definition 151)

Figure 4.4: Reduction rules of Figure 4.3 reformulated in our setting.

They are translated into reduction rules through the protocol described in Section 4.3.5.

Figure 4.3 gives Kesner's rules. Five out of six of Kesner's rules translate straightforwardly, see Figure 4.4. Note how the explicit weakening $\iota : \Theta \to \Theta'$ takes into account the side condition $x \notin \mathsf{fv}(t)$ of the Gc-rule in Figure 4.4.

The side condition $y \in \mathsf{fv}(u)$ of [][]-rule is more involved and necessitates the definition of the $\Sigma_{\mathsf{LC_{ex}}}$-module Θ_* such that $\mathsf{LC_{ex}}_*$ is the submodule of $\mathsf{LC_{ex}}'$ of terms that really depend on the fresh variable.

We propose an approach based on the informal intuitive idea of defining inductively the submodule R_* of R' depending on the fresh variable $*$ as follows, for a given $\Sigma_{\mathsf{LC_{ex}}}$-monad R:

- $\eta(*) \in R_*(X)$, for any set X;
- (application)
  - if $t \in R(X)$ and $u \in R_*(X)$, then $\mathsf{app}(\iota(t), u) \in R_*(X)$
  - if $t \in R_*(X)$ and $u \in R(X)$, then $\mathsf{app}(t, \iota(u)) \in R_*(X)$
  - if $t \in R_*(X)$ and $u \in R_*(X)$, then $\mathsf{app}(t, u) \in R_*(X)$
- if $t \in R_*(X + \{x\})$, then $\lambda x.t \in R_*(X)$;
- (explicit substitution)
  - if $t \in R(X + \{x\})$ and $u \in R_*(X)$, then $\iota(t)[x/u] \in R_*(X)$;
  - if $t \in R_*(X + \{x\})$ and $u \in R(X)$, then $t[x/\iota(u)] \in R_*(X)$;
  - if $t \in R_*(X + \{x\})$ and $u \in R_*(X)$, then $t[x/u] \in R_*(X)$.

Guided by this intuition, we now formally define a a $\Sigma_{\mathsf{LC_{ex}}}$-module Θ_* equipped with a morphism $\kappa : \Theta_* \to \Theta'$.

The previous informal inductive definition is translated as an initial algebra for an endofunctor on the category of $\Sigma_{\mathsf{LC_{ex}}}$-modules, which is cocomplete (colimits are computed pointwise). This endofunctor maps a $\Sigma_{\mathsf{LC_{ex}}}$-module M to the coproduct of the following $\Sigma_{\mathsf{LC_{ex}}}$-modules:

- the terminal $\Sigma_{\mathsf{LC_{ex}}}$-module 1, playing the rôle of the fresh variable;
- the coproduct $M \times \Theta + \Theta \times M + M \times M$, one summand for each case of the application;
- the derived module Θ' for abstraction;
- the coproduct $M' \times \Theta + \Theta' \times M + M' \times M$, one summand for each case of the explicit substitution.

This functor is finitary, so the initial algebra exists thanks to Adámek's theorem (already cited, as Theorem 144). Unfortunately, the resulting $\Sigma_{\mathsf{LC_{ex}}}$-module does not yield the module that we are expecting in the case of the monad $\mathsf{LC_{ex}}$: it does not satisfy Equation 4.5, and thus contains more terms than necessary. To obtain the desired $\Sigma_{\mathsf{LC_{ex}}}$-module, we equip Θ' with its canonical algebra structure, inducing a morphism from the initial algebra, and we define Θ_* as the image of this morphism, thus equipped with an inclusion $\kappa : \Theta_* \to \Theta'$.

Definition 151. We define the $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$**-module of "one-hole contexts"** to be Θ_*, equipped with an inclusion $\kappa : \Theta_* \to \Theta$.

Remark 152. Such a definition can be worked out for any algebraic signature for monads.

Now we define the signature of the reduction monad of lambda calculus with explicit substitutions:

Definition 153. The reduction signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{ex}}}$ of the lambda calculus reduction monad with explicit substitutions consists of the signature $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$ of Definition 149 and all the reduction rules specified in this section.

Lemma 154. *The reduction signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{ex}}}$ is effective.*

Proof. We apply Theorem 128—the underlying signature for monads is effective by Lemma 150. □

4.7 Recursion

In this section, we derive, for any effective reduction signature $\mathcal{S}$, a recursion principle from initiality. In Section 4.7.1, we state this recursion principle, then we give an example of application in Section 4.7.2, by translating lambda calculus with a fixpoint operator to lambda calculus. In Section 4.7.2, we apply this principle to translate lambda calculus with explicit substitutions into lambda calculus. The non-trivial part in this translation consists in exhibiting the congruence of the monadic substitution. This is done by induction, and motivates the last Section 4.7.4.

4.7.1 Recursion principle for effective signatures

The recursion principle associated to an effective signature provides a way to construct a morphism from the reduction monad underlying the initial model of that signature to a given reduction monad.

Proposition 155 (Recursion principle)**.** *Let $\mathcal{S}$ be an effective reduction signature, and R be the reduction monad underlying the initial model. Let T be a reduction monad. Any action τ of $\mathcal{S}$ in T induces a reduction monad morphism $\hat{\tau} : R \to T$.*

Proof. The action τ defines a model M of $\mathcal{S}$. By initiality, there is a unique model morphism from the initial model to M, and $\hat{\tau}$ is the reduction monad morphism underlying it. □

In the next sections, we illustrate this principle.

4.7.2 Translation of lambda calculus with fixpoint to lambda calculus

In this section, we consider the signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ of Example 131 for the lambda calculus with an explicit fixpoint operator.

We build, by recursion, a reduction monad morphism from the initial model $\mathsf{LC}_{\mathsf{fix}}$ of this signature to $\mathsf{LC}^*_{\beta/\eta}$, the "closure under identity and composition of reductions" (Definition 105) of the initial model $\mathsf{LC}_{\beta/\eta}$ of the signature $\mathcal{S}_{\mathsf{LC}}$ (Example 130).

As explained in Section 4.7.1, we need to define an action of $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ in $\mathsf{LC}^*_{\beta/\eta}$. Note that $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ is an extension of $\mathcal{S}_{\mathsf{LC}}$ (Example 131). First, we focus on the core $\mathcal{S}_{\mathsf{LC}}$ part: we show that the reduction monad $\mathsf{LC}^*_{\beta/\eta}$ inherits the canonical action of $\mathcal{S}_{\mathsf{LC}_{\beta/\eta}}$ in $\mathsf{LC}_{\beta/\eta}$.

Lemma 156. *There is an action of $\mathcal{S}_{\mathsf{LC}_{\beta/\eta}}$ in $\mathsf{LC}^*_{\beta/\eta}$.*

Proof. The challenge is to give an action of reduction rules with hypotheses: now the input reductions of the rule may be actually sequences of reductions. This concerns congruence for application and abstraction. We take the example of abstraction: suppose we have a sequence of reductions $r_1 \dots r_n$ going from t_0 to t_n. We want to provide a reduction between $\mathsf{abs}(t_0)$ and $\mathsf{abs}(t_n)$. For each i, we have a reduction between $\mathsf{abs}(t_{i-1})$ and $\mathsf{abs}(t_i)$. By composing the corresponding sequence, we obtain the desired reduction. □

The action for the extra parts of $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ require the following:

- an operation $\mathsf{fix} : \mathsf{LC}_{\mathsf{fix}}{}' \to \mathsf{LC}_{\mathsf{fix}}$: for this, we choose a fixpoint combinator Y (for example, the one of Curry), and set $\mathsf{fix}_X(t) = \mathsf{app}(Y, \mathsf{abs}(t))$, in accordance with Section 2.8.4
- an action of the reduction rule

$$\frac{}{\mathsf{fix}(T) \rightsquigarrow T\{* := \mathsf{fix}(T)\}}$$

A fixpoint combinator Y is a closed term with the property that for any other term t, the term $\mathsf{app}(Y,t)$ β-reduces in some steps to $\mathsf{app}(t,\mathsf{app}(Y,t))$. We denote by $r \in \mathsf{Red}(\mathsf{LC}^*_{\beta/\eta})(\{*\})$ a reduction between $\mathsf{app}(Y,*)$ and $\mathsf{app}(*,\mathsf{app}(Y,*))$. Then, r induces an $\underline{\mathsf{LC}_{\beta/\eta}}$-module morphism $\hat{r} : \underline{\mathsf{LC}_{\beta/\eta}} \to \mathsf{Red}(\mathsf{LC}^*_{\beta/\eta})$ by mapping an element $t \in \mathsf{LC}_{\beta/\eta}(X)$ to $r\{* := t\}$. We define the action of this reduction rule as the composition of the following reductions:

$$\mathsf{app}(Y,\mathsf{abs}(t)) \rightsquigarrow_{\hat{r}(\mathsf{abs}(t))} \mathsf{app}(\mathsf{abs}(t),\mathsf{app}(Y,\mathsf{abs}(t))) \rightsquigarrow_{\beta} t\{* := \mathsf{app}(Y,\mathsf{abs}(t))\}$$

- an action of the congruence rule

$$\frac{T \rightsquigarrow T'}{\mathsf{fix}(T) \rightsquigarrow \mathsf{fix}(T')}$$

that can be defined in the obvious way using the congruences of application and abstraction.

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

4.7.3 Translation of lambda calculus with explicit substitutions into lambda calculus with β-reduction

In this section, we consider the reduction signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{ex}}} = (\Sigma_{\mathsf{LC}_{\mathsf{ex}}}, \mathfrak{R}_{\mathsf{LC}_{\mathsf{ex}}})$ introduced in Definition 153. The underlying monad of the initial model $\mathsf{LC}_{\mathsf{ex}}$ is the monad of lambda calculus with an application and abstraction operation, and an explicit substitution operator $\mathsf{LC}_{\mathsf{ex}}{}' \times \mathsf{LC}_{\mathsf{ex}} \to \mathsf{LC}_{\mathsf{ex}}$ satisfying Equation 4.5, for $R = \mathsf{LC}_{\mathsf{ex}}$. The associated reduction monad has all the rules specified in Section 4.6.

We build, by recursion, a reduction monad morphism from the initial model $\mathsf{LC}_{\mathsf{fix}}$ of this signature to $\mathsf{LC}^*_{\beta/\eta}$, the same reduction monad as in the previous section.

As explained in Section 4.7.1, we need to define an action of $\mathcal{S}_{\mathsf{LC}_{\mathsf{ex}}}$ in $\mathsf{LC}^*_{\beta/\eta}$. For the action of the underlying 1-signature, we give the canonical application and abstraction operations of $\mathsf{LC}^*_{\beta/\eta}$, as the initial Σ_{LC}-monad. The explicit substitution operation $\mathsf{LC}^*_{\beta/\eta}{}' \times \mathsf{LC}^*_{\beta/\eta} \to \mathsf{LC}^*_{\beta/\eta}$ is defined using the monadic substitution, mapping a pair

$(t, u) \in \mathsf{LC}^*_{\beta/\eta}(X + \{*\}) \times \mathsf{LC}^*_{\beta/\eta}(X)$ to the monadic substitution $t\{* := u\}$. One checks that this operation satisfies Equation 4.5, thanks to the usual monadic equations.

The action of the underlying signature for the congruence rules of application and abstraction come from Lemma 156. The other actions are all given by reflexivity, except the β-rule which already has an action in $\mathsf{LC}^*_{\beta/\eta}$.

The non-obvious actions are the ones of the congruence rules for explicit substitution:

$$\frac{T \rightsquigarrow T'}{T[x/U] \rightsquigarrow T[x/U']} \qquad \frac{U \rightsquigarrow U'}{T[x/U] \rightsquigarrow T[x/U']}$$

The left one is obtained from the substitution of the module of reductions (see Remark 95). The action of the right one is constructed by induction from congruences of $\mathsf{LC}^*_{\beta/\eta}$: it is a consequence of Lemma 159, presented in the next section as an example of induction, for the specific case of a unary substitution.

Thus, we get an action for the congruence rule for explicit substitution in $\mathsf{LC}^*_{\beta/\eta}$. Finally, by the recursion principle, we get a reduction monad morphism from $\mathsf{LC}_{\mathsf{ex}}$ to $\mathsf{LC}^*_{\beta/\eta}$. This translation replaces the explicit substitution operator $t[x/u]$ with the corresponding monadic substitution $t\{x := u\}$, and all the reductions are translated to the reflexivity except for the ones for the β-reduction and congruence for the substitution.

4.7.4 Induction principle

TODO déplacer dans chapitre FSCD ? Let Σ be an algebraic signature for monads. In this section, we state an induction principle based on initiality in the category of Σ-monads. This is a variant of the recursion principle of Section 3.5. Although a similar one can be stated for reduction monads, here we focus on the mere monadic case as we have a case use: congruence for substitution (Lemma 159), that is actually used in translating the reduction monad of lambda calculus with explicit substitutions into the reduction monad of the lambda calculus with β-reduction (Section 4.7.3).

Let $\widehat{\Sigma}$ be the initial model of Σ and $P_X(t)$ be predicates on $t \in \widehat{\Sigma}(X)$ for any X. The induction principle allows to prove that for any set X, the elements of $\widehat{\Sigma}(X)$ all satisfy P_X.

The basic idea is to construct the submonad of $\widehat{\Sigma}$ satisfying this property, show that it induces a Σ-monad, and deduce by initiality the universal satisfaction. The first

requirement is that the subsets of elements $t \in \widehat{\Sigma}(X)$ satisfying $P_X(t)$ induces a monad:

Lemma 157. *Let R be a monad. For a given set X and element $t \in \mathsf{LC}^*_{\beta/\eta}(X)$, let $P_X(t)$ be a predicate on t. Then, the mapping $R_{|P} : X \mapsto \{t \in \widehat{\Sigma}(X) | P_X(t)\}$ induces a submonad $R_{|P}$ of R on the following conditions:*

Variables *the predicate holds for any variable: for any set X and element $x \in X$, the statement $P_X(\eta(x))$ is satisfied;*

Monadic substitution *the predicate is stable under substitution: for any substitution $f : X \to R(Y)$, any term $t \in R(X)$, if $P_X(t)$ and $P_Y(f(x))$ are satisfied for all $x \in X$, then so is $P_Y(t\{f\})$.*

Proof. We show that $R_{|P}$ inherits the monadic structure of R.

- Variables: Let X be a set and $x \in X$. We check that $\eta(x) \in R(X)$ is actually in $R_{|P}(X)$, and indeed, the predicate holds for any variable by assumption.
- Monadic substitution: Let $u : X \to R_{|P}(Y)$ be a function and $t \in R_{|P}(X)$. The monadic substitution of R provides an element $t\{u\}$ in $R(Y)$. We check that it is actually in $R_{|P}(Y)$ and indeed, $P_Y(t\{u\})$ holds because substitution preserves the predicate by assumption.

The monadic equations automatically hold since R is a monad. It is clear that the inclusion $R_{|P} \to R$ is a monad morphism. □

Giving the monad $\widehat{\Sigma}_{|P}$ an action of Σ allows to prove the following induction principle:

Lemma 158 (Induction principle)**.** *For a given set X and element $t \in \widehat{\Sigma}(X)$, let $P_X(t)$ be a predicate on t. Suppose that P satisfies the conditions of Lemma 157, so that $\widehat{\Sigma}_{|P}$ denotes the induced submonad of $\widehat{\Sigma}$. Then, for any set X, the predicate P_X is satisfied for any element $t \in \widehat{\Sigma}(X)$ provided that the predicate is stable under the action $\sigma : \Sigma(\widehat{\Sigma}) \to \widehat{\Sigma}$ in the following sense: for any set X and any term $t \in \Sigma(\widehat{\Sigma}_{|P})(X)$, the term $\sigma_X(\Sigma(p)(t))$ satisfies P_X, where:*

- *$p : \widehat{\Sigma}_{|P} \to \widehat{\Sigma}$ is the monadic inclusion;*
- *$\Sigma(\_) : \mathsf{Mon} \to \int \mathsf{Mod}$ denotes the underlying 1-signature functor of the signature Σ.*

Proof. Our goal is to show that the inclusion $p : \widehat{\Sigma}_{|P} \to \widehat{\Sigma}$ is (pointwise) surjective. The plan is the following:

1. show that $\widehat{\Sigma}_{|P}$ inherits the action of Σ in $\widehat{\Sigma}$;
2. show that the monadic inclusion $p : \widehat{\Sigma}_{|P} \to \widehat{\Sigma}$ is a Σ-monad morphism;
3. by initiality, we get a section $i : \widehat{\Sigma} \to \widehat{\Sigma}_{|P}$ of p, implying that p is indeed pointwise surjective.

Step 1 First, we show that $\widehat{\Sigma}_{|P}$ inherits the action of Σ in $\widehat{\Sigma}$. Let $t \in \Sigma(\widehat{\Sigma}_{|P})(X)$. We check that $\sigma(\Sigma(p)(t)) \in \widehat{\Sigma}(X)$ is actually in $\widehat{\Sigma}_{|P}$: and indeed $P_X(\sigma(\Sigma(p)(t)))$ holds as P is stable by the action, by assumption.

The equations specified by the signature Σ automatically hold for $\widehat{\Sigma}_{|P}$, because they do for $\widehat{\Sigma}$. This exploits the fact that Σ is algebraic, in particular, that the codomain of any equation is of the shape $\Theta^{(n)}$ and thus preserves pointwise monomorphisms.

Step 2 The fact that the inclusion p commutes with the action is true by definition.

Step 3 As $\widehat{\Sigma}$ is the initial Σ-monad, we get a morphism $i : \widehat{\Sigma} \to \widehat{\Sigma}_{|P}$ of Σ-monad. Then $p \circ i$ is an endomorphism of the Σ-monad $\widehat{\Sigma}$, and by uniqueness of the initial morphism, it is the identity morphism, so i is a section of p. □

We give now an example of use: giving $\mathsf{LC}^*_{\beta/\eta}$ an "action" of the following informal reduction rule (which is made precise in the upcoming Remark 161),

$$\frac{\forall x \in X,\ f(x) \rightsquigarrow f'(x)}{T\{f\} \rightsquigarrow T\{f'\}}$$

Lemma 159. *Consider* $\mathsf{LC}^*_{\beta/\eta}$*, the closure under identity and composition of reductions of* $\mathsf{LC}^*_{\beta/\eta}$*, the reduction monad* $\mathsf{LC}^*_{\beta/\eta}$ *of lambda calculus with* β*-reduction and congruences. Then, monadic substitution lifts congruence, in the following sense: for any sets* X *and* Y*, any element* t *in* $\mathsf{LC}^*_{\beta/\eta}(X)$*, any families* $(u_x)_{x\in X}$ *and* $(v_x)_{x\in X}$ *of elements of* $\mathsf{LC}^*_{\beta/\eta}(Y)$*, and any family* $(m_x)_{x\in X}$ *of reductions such that* $m_x : u_x \blacktriangleright v_x$ *for all* $x \in X$*, there is a reduction between* $t\{x \mapsto u_x\}$ *and* $t\{x \mapsto v_x\}$*.*

Proof. It is a straightforward application of Lemma 158. □

We end this section with some remarks about this congruence result:

Remark 160. There is another informal reduction rule that we can think of regarding congruence of the monadic substitution:

$$\frac{T \rightsquigarrow T'}{T\{f\} \rightsquigarrow T'\{f\}}$$

Actually, this rule is automatically validated in any reduction monad thanks to the module structure on the reductions (see Remark 95)

Remark 161. If, in the definition of reduction rules, we allow that the Σ-module of metavariables depends on the module of reductions, then this statement is equivalent to saying that $\mathsf{LC}^*_{\beta/\eta}$ is equipped with an action of the following reduction rule:

- Metavariables: $\mathcal{V} = \Theta \cdot \mathsf{Red}$
- Hypotheses: No hypothesis.
- Conclusion: The term-pair of the conclusion is defined as follows:

$$\Theta \cdot \mathsf{Red} \xrightarrow{\Theta \cdot \mathsf{red}} \Theta \cdot (\Theta \times \Theta) \xrightarrow{\Theta \cdot \pi_1, \Theta \cdot \pi_2} (\Theta \cdot \Theta)^2 \xrightarrow{\mu \times \mu} \Theta^2$$

Remark 162. Given any reduction signature $\mathcal{S}$ such that the underlying 1-signature is algebraic, or more generally, comes from a strengthened signature in the sense of [HM12], it is possible to define a congruence reduction rule for the specified operations of the monads. If this reduction rule is in the signature $\mathcal{S}$, then a similar lemma about the congruence of monadic substitution can be proven.

CHAPTER 5

OPERATIONAL MONADS AND THEIR SIGNATURES

In this chapter, we extend the notion of *reduction monads* of Chapter 4. Our goal is to deal with programming languages whose terms do not gather into a monad, but rather into a module over a monad. Consider the example of the lambda calculus in the call-by-value variant: there, variables can be replaced with values only, rather than any term. Yet, we are interested in reductions between terms rather than values. In this situation, there is a monad of values, and terms gather into a module over this monad.

To be more precise, we limit ourselves to languages whose terms form a free module over a monad R, that is, a module of the shape $T \cdot R$ for some endofunctor $T : \mathsf{Set} \to \mathsf{Set}$. This is indeed the case of the call-by-value lambda calculus: each term can be decomposed uniquely as a binary tree whose leaves are values. For this specific example, R is the monad of values of the lambda calculus, and T is the endofunctor generating binary trees out of leaves taken in its domain.

This chapter is a product of a collaboration with André Hirshowitz and Tom Hirschowitz. We adopt the same terminology and notations as in Chapter 4. We are quickier here, as all the definitions and proofs are straightforward generalizations of those found in this previous chapter.

Plan of the chapter

TODO

5.1 Operational monads

TODO écrire un chapeau

5.1.1 Operational monads

Definition 163. Let $T : \mathsf{Set} \to \mathsf{Set}$ be a functor. A **T-reduction monad** R is given by:

1. a monad on sets (the monad of *terms*), that we still denote by R, or by $\underline{R}$ when we want to be explicit;
2. an R-module $\mathsf{Red}(R)$ (the module of *reductions*);
3. a morphism of R-modules $\mathsf{red}_R : \mathsf{Red}(R) \to (T \cdot R) \times (T \cdot R)$ (*source* and *target* of rules).

We set $\mathsf{source}_R := \pi_1 \circ \mathsf{red}_R : \mathsf{Red}(R) \to T \cdot R$, and $\mathsf{target}_R := \pi_2 \circ \mathsf{red}_R : \mathsf{Red}(R) \to T \cdot R$

For a T-reduction monad R, a set X, and elements $s, t \in T(R(X))$, we think of the fiber $\mathsf{red}_R(X)^{-1}(s,t)$ as the set of "reductions from s to t". We sometimes write $m : s \blacktriangleright t : T(R(X))$, or even $m : s \blacktriangleright t$ when there is no ambiguity, instead of $m \in \mathsf{red}_R(X)^{-1}(s,t)$.

Remark 164. We recover chapter 4 by taking $T = \mathsf{Id}_{\mathsf{Set}}$.

Definition 165. Let $T : \mathsf{Set} \to \mathsf{Set}$ be a functor. A **morphism of T-reduction monads** from R to S is given by a pair (f, α) of

1. a monad morphism $f : R \to S$, and
2. a natural transformation $\alpha : \mathsf{Red}(R) \to \mathsf{Red}(S)$

satisfying the following two conditions:

3. α is an R-module morphism between $\mathsf{Red}(R)$ and $f^*\mathsf{Red}(S)$, and
4. the square

$$\begin{array}{ccc} \mathsf{Red}(R) & \xrightarrow{\quad\alpha\quad} & \mathsf{Red}(S) \\ \downarrow{\scriptstyle \mathsf{red}_R} & & \downarrow{\scriptstyle \mathsf{red}_S} \\ (T \cdot R) \times (T \cdot R) & \xrightarrow[Tf \times Tf]{} & (T \cdot S) \times (T \cdot S) \end{array}$$

commutes in the category of natural transformations.

Proposition 166 (Category of T-reduction monads)**.** *Let $T : \mathsf{Set} \to \mathsf{Set}$ be a functor. Then, T-reduction monads and their morphisms, with the obvious composition and identity, form a category T-RedMon, equipped with a forgetful functor to the category of monads.*

We get an analogous of Theorem 98:

Theorem 167. *Let* $T : \mathsf{Set} \to \mathsf{Set}$ *be a functor. The category of* T*-reduction monads is isomorphic to the category of monads relative to the functor mapping a set to its discrete* T*-graph, where the category of* T*-graphs is the comma category* $\mathsf{Set}/(T \times T)$*.*

Definition 168. An **operational monad** is a pair (T, R) of a functor $T : \mathsf{Set} \to \mathsf{Set}$ and a T-reduction monad R.

Any example of reduction monad R of Chapter 4 yields an operational monad (Id, R).

Example 169 (Call-by-value lambda calculus)**.** In call-by-value, variables can only be substituted with values. A value is either a variable or a lambda abstraction of an arbitrary term. Note that values are stable under substitution: they induce a monad $\mathsf{LC}_{cbv} = \mathsf{Id} + \mathsf{LC}'$ equipped with a monad morphism $i : \mathsf{LC}_{cbv} \to \mathsf{LC}'$.

Now, reductions are between terms rather than between values, so we need to devise a functor $T : \mathsf{Set} \to \mathsf{Set}$ such that $T \cdot \mathsf{LC}_{cbv}$ is isomorphic to LC. Note that a lambda term can always be decomposed as a binary tree whose leaves are values, that is variables or lambda abstractions: each node of this tree is an application. Hence, we choose T to be the monad of binary trees specified by the 1-signature $\Theta \times \Theta$: then, there is an isomorphism $\alpha : \mathsf{LC} \to T \cdot \mathsf{LC}_{cbv}$ in the category of LC_{cbv}-modules (LC is indeed equipped with a structure of LC_{cbv}-module thanks to the inclusion i of LC_{cbv} in LC as a monad morphism). The operational monad of call-by-value lambda calculus is the pair (T, LC_{cbv}) where the T-reduction monad LC_{cbv} is defined as follows:

1. the underlying monad is $\mathsf{LC}_{cbv} = \mathsf{Id} + \mathsf{LC}'$;
2. $\mathsf{red}_{\mathsf{LC}_{cbv}} : \mathsf{Red}(\mathsf{LC}_{cbv}) \to T \cdot \mathsf{LC}_{cbv} \simeq \mathsf{LC}$ is generated by the following constructions:
   (a) for $t \in \mathsf{LC}'(X)$ and $u \in \mathsf{LC}_{cbv}(X)$, we have $\beta(t, u) : \alpha(\mathsf{app}(\mathsf{abs}(t), i(u))) \rightsquigarrow \alpha(t)\{* := u\}$;
   (b) for $m : \alpha(u) \rightsquigarrow \alpha(v) : T(\mathsf{LC}_{cbv}(X))$ in $\mathsf{Red}(\mathsf{LC}_{cbv})$ and $t \in \mathsf{LC}(X)$, we have $\mathsf{app\text{-}cong}_1(m, t) : \alpha(\mathsf{app}(u, t)) \rightsquigarrow \alpha(\mathsf{app}(v, t))$
   (c) for $m : \alpha(u) \rightsquigarrow \alpha(v) : T(\mathsf{LC}_{cbv}(X))$ in $\mathsf{Red}(\mathsf{LC}_{cbv})$ and $t \in \mathsf{LC}(X)$, we have $\mathsf{app\text{-}cong}_2(t, m) : \alpha(\mathsf{app}(t, u)) \rightsquigarrow \alpha(\mathsf{app}(t, v))$
   (d) for $m : \alpha(u) \rightsquigarrow \alpha(v) : T(\mathsf{LC}'_{cbv}(X))$ in $\mathsf{Red}(\mathsf{LC}_{cbv})$ we have $\mathsf{abs\text{-}cong}(m) : \alpha(\mathsf{abs}(u)) \rightsquigarrow \alpha(\mathsf{abs}(v))$

We call this operational monad *the operational monad of the call-by-value lambda calculus*. A signature for it is given in Section 5.4.1.

5.2 T-Reduction rules

TODO écrire un chapeau

5.2.1 Definition of T-reduction rules

In this subsection, Σ is a signature for monads, and T is an endofunctor on Set. We present our notion of *T-reduction rule over Σ*, from which we build *T-reduction signatures* in Section 5.3.

We begin with the definition of *T-term-pair*:

Definition 170. Given a Σ-module $\mathcal{V}$, a **T-term-pair from $\mathcal{V}$** is a pair (n, p) of a natural number n and a morphism of Σ-modules $p : \mathcal{V} \to (T \cdot \Theta^{(n)}) \times (T \cdot \Theta^{(n)})$.

We now give our definition of *T-reduction rule*.

Definition 171. A **T-reduction rule** $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$ **over** Σ is given by:

- Metavariables: a Σ-module $\mathcal{V}$ of metavariables, that we sometimes denote by $\mathsf{MVar}_{\mathcal{A}}$;
- Hypotheses: a finite family of T-term-pairs $(n_i, h_i)_{i \in I}$ from $\mathcal{V}$;
- Conclusion: a T-term-pair (n, c) from $\mathcal{V}$.

5.2.2 T-Reduction Σ-monads

Similarly to Chapter 4, the destiny of a T-reduction rule is to be validated in a T-reduction monad R. Here again, as the hypotheses or the conclusion of the T-reduction rule may refer to some operations specified by a signature Σ for monads, this T-reduction monad R must be equipped with an action of Σ, hence the following definition:

Definition 172. Let Σ be a signature for monads, and T be an endofunctor on Set. The **category T-RedMon^{Σ} of T-reduction Σ-monads** is defined as the following pullback:

$$\begin{array}{ccc} T\text{-}\mathsf{RedMon}^{\Sigma} & \longrightarrow & T\text{-}\mathsf{RedMon} \\ \downarrow & & \downarrow \\ \mathsf{Mon}^{\Sigma} & \longrightarrow & \mathsf{Mon} \end{array}$$

More concretely,

- a **T-reduction Σ-monad** is a T-reduction monad R equipped with an **action** ρ of Σ in R, thus inducing a Σ-monad that we denote also by R, or by $\underline{R}$ when we want to be explicit;
- a **morphism of T-reduction Σ-monads** $R \to S$ is a morphism $f : R \to S$ of T-reduction monads which commutes with the action of Σ, i.e, whose underlying monad morphism is a Σ-monad morphism.

5.2.3 Action of a T-reduction rule

Let Σ be a signature for monads and T be an endofunctor on Set. In this section, we introduce the notion of *action of a T-reduction rule over Σ in a T-reduction Σ-monad*. Intuitively, such an action is a "map from the hypotheses to the conclusion" of the T-reduction rule. To make this precise, we need to first take the product of the hypotheses; this product is, more correctly, a *fibred* product.

Definition 173. Let (n, p) be a T-term-pair from a Σ-module $\mathcal{V}$, and R be a T-reduction Σ-monad. We denote by $p^*(\mathsf{Red}(R)^{(n)})$ the pullback of $\mathsf{red}_R^{(n)} : \mathsf{Red}(R)^{(n)} \to (T \cdot R^{(n)}) \times (T \cdot R^{(n)})$ along $p_R : \mathcal{V}(R) \to (T \cdot R^{(n)}) \times (T \cdot R^{(n)})$:

$$\begin{array}{ccc} p^*(\mathsf{Red}(R)^{(n)}) & \longrightarrow & \mathsf{Red}(R)^{(n)} \\ \downarrow & & \downarrow{\scriptstyle \mathsf{red}_R^{(n)}} \\ \mathcal{V}(R) & \xrightarrow[p_R]{} & (T \cdot R^{(n)}) \times (T \cdot R^{(n)}) \end{array}$$

We denote by $p^*(\mathsf{red}_R^{(n)}) : p^*(\mathsf{Red}(R)^{(n)}) \to \mathcal{V}(R)$ the projection morphism on the left.

Definition 174. Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$ be a T-reduction rule.

The R**-module** $\mathsf{Hyp}_{\mathcal{A}}(R)$ **of hypotheses of** $\mathcal{A}$ is $\prod_{i \in I/\mathcal{V}(R)} h_i^* \mathsf{Red}(R)^{(n_i)}$ the fiber product of all the R-modules $h_i^* \mathsf{Red}(R)^{(n_i)}$ along their projection to $\mathcal{V}(R)$. It thus comes with a projection $\mathsf{hyp}_{\mathcal{A}}(R) : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathcal{V}(R)$

The R**-module** $\mathsf{Con}_{\mathcal{A}}(R)$ **of conclusion of** $\mathcal{A}$ is $c^* \mathsf{Red}(R)^{(n)}$, and comes with a projection $\mathsf{con}_{\mathcal{A}}(R) : \mathsf{Con}_{\mathcal{A}}(R) \to \mathcal{V}(R)$.

Definition 175. Let $\mathcal{A}$ be a T-reduction rule over Σ. An **action of** $\mathcal{A}$ **in a** T**-reduction** Σ**-monad** R is a morphism between $\mathsf{hyp}_{\mathcal{A}}(R)$ and $\mathsf{con}_{\mathcal{A}}(R)$ in the slice category $\mathsf{Mod}(R)/\mathsf{MVar}_{\mathcal{A}}$, that is, a morphism of R-modules

$$\tau : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathsf{Con}_{\mathcal{A}}(R)$$

making the following diagram commute:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\quad\tau\quad} & \mathsf{Con}_{\mathcal{A}}(R) \\ & \searrow \quad \swarrow & \\ & \mathsf{MVar}_{\mathcal{A}}(R) & \end{array} \tag{5.1}$$

5.3 Signatures for operational monads and Initiality

In this section, we define the notion of T*-reduction signature*, consisting of a signature for monads Σ and a family of T-reduction rules over Σ (see Section 5.3.1). As usual, we assign to each such signature a *category of models*. We call a T-reduction signature *effective* if the associated category of models has an initial object. Our main result, Theorem 184, states that a T-reduction signature is effective as soon as its underlying signature for monads is effective.

5.3.1 Signatures and their models

We define here T*-reduction signatures* and their *models*.

Definition 176. A T**-reduction signature** is a pair $(\Sigma, \mathfrak{R})$ of a signature Σ for monads and a family $\mathfrak{R}$ of T-reduction rules over Σ.

Definition 177. Given an operational monad (T, R) and an operational signature $\mathcal{S} = (\Sigma, T', \mathfrak{R})$, an **action of** $\mathcal{S}$ **in** R consists of:

- an action of Σ in its underlying monad $\underline{R}$;
- an assertion that $T = T'$;
- an action of each T-reduction rule of $\mathfrak{R}$ in R.

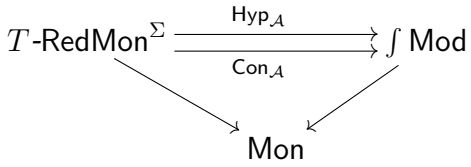
Definition 178. Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a T-reduction signature. A **model of** $\mathcal{S}$ is a T-reduction monad equipped with an action of $\mathcal{S}$, or equivalently, a T-reduction Σ-monad equipped with an action of each T-reduction rule of $\mathfrak{R}$.

5.3.2 The functors $\mathsf{Hyp}_{\mathcal{A}}$ and $\mathsf{Con}_{\mathcal{A}}$

The definition of morphism between models of a T-reduction signature relies on the functoriality of the assignments $R \mapsto \mathsf{Hyp}_{\mathcal{A}}(R)$ and $R \mapsto \mathsf{Con}_{\mathcal{A}}(R)$, for a given T-reduction rule $\mathcal{A}$ on a signature Σ for monads.

Definition 179. Let Σ be a signature for monads, and $\mathcal{A}$ be a T-reduction rule over Σ. Definition 174 assigns to each Σ-monad R the R-modules $\mathsf{Hyp}_{\mathcal{A}}(R)$ and $\mathsf{Con}_{\mathcal{A}}(R)$. These assignments extend to functors $\mathsf{Hyp}_{\mathcal{A}}, \mathsf{Con}_{\mathcal{A}} : T\text{-}\mathsf{RedMon}^{\Sigma} \to \mathsf{Mon}^{\Sigma}$.

Proposition 180. *Given the same data, the functors* $\mathsf{Hyp}_{\mathcal{A}}$ *and* $\mathsf{Con}_{\mathcal{A}}$ *commute with the forgetful functors to* Mon*:*



5.3.3 The main result

For a T-reduction signature $\mathcal{S}$, we define here the notion of $\mathcal{S}$-model morphism, inducing a **category of models of** $\mathcal{S}$. We then state our main result, Theorem 184, which give a sufficient condition for $\mathcal{S}$ to admit an initial model.

Definition 181. Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a T-reduction signature. A morphism between models R and S of $\mathcal{S}$ is a morphism f of T-reduction Σ-monads commuting with the action of any T-reduction rule, in the sense that for any T-reduction rule $\mathcal{A} \in \mathfrak{R}$, the following

diagram of natural transformations commutes:

$$\begin{array}{ccc} \mathsf{Hyp}_{\mathcal{A}}(R) & \longrightarrow & \mathsf{Con}_{\mathcal{A}}(R) \\ {\scriptstyle \mathsf{Hyp}_{\mathcal{A}}(f)}\downarrow & & \downarrow{\scriptstyle \mathsf{Con}_{\mathcal{A}}(f)} \\ \mathsf{Hyp}_{\mathcal{A}}(S) & \longrightarrow & \mathsf{Con}_{\mathcal{A}}(S) \end{array}$$

Proposition 182. *Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a T-reduction signature. Models of $\mathcal{S}$ and their morphisms, with the obvious composition and identity, define a category that we denote by T-RedMon$^{\mathcal{S}}$, equipped with a forgetful functor to T-RedMon$^{\Sigma}$.*

Definition 183. A T-reduction signature $\mathcal{S}$ is said to be **effective** if its category of models T-RedMon$^{\mathcal{S}}$ has an initial object, denoted $\widehat{\mathcal{S}}$. In this case, we say that $\widehat{\mathcal{S}}$ is **generated (or specified) by** $\mathcal{S}$.

We now have all the ingredients required to state our main result:

Theorem 184. *Let $(\Sigma, \mathfrak{R})$ be a T-reduction signature. If Σ is effective, then so is $(\Sigma, \mathfrak{R})$.*

Proof. The proof of this theorem is a straightforward adaptation of the proof described in Section 4.5. □

Theorem 83 entails the following corollary:

Corollary 185. *Let $(\Sigma, \mathfrak{R})$ be a T-reduction signature. If Σ is algebraic (in the sense of Definition 82), then $(\Sigma, \mathfrak{R})$ is effective.*

All the examples of T-reduction signatures considered here satisfy the algebraicity condition of Corollary 185.

5.4 Examples of signatures

We give examples of signatures, giving a schematic presentation of reduction rules as in Section 4.3.5.

5.4.1 Call-by-value lambda calculus

We give a signature for the operational monad of the call-by-value lambda calculus (T, LC_{cbv}) (Example 169). Recall that T is the monad of binary trees underlying the

initial model of the 1-signature $\Theta \times \Theta$: it comes equipped with an application operation $T \times T \to T$.

Now, we give a signature for the monad LC_{cbv} of values of the lambda calculus. A value is either a variable or a an abstracted lambda term, and we argued in Example 169 that a lambda term can be specified as a binary tree whose leaves are values. Thus, we choose the 1-signature $T \cdot \Theta'$ to specify the monad of values. Note that this is algebraic as T is polynomial:

$$T \cdot \Theta' = \coprod_{n \in \mathbb{N}} B_n \times (\Theta')^n,$$

where B_n is the set of binary trees with n leaves.

Recall from Example 169 that there is an isomorphism $\alpha : \mathsf{LC} \to T \cdot \mathsf{LC}_{cbv}$ (rephrasing the fact that lambda terms are binary trees with values as leaves), and we denote $i : \mathsf{LC}_{cbv} \to \mathsf{LC}$ the canonical inclusion of values into terms.

We give now the reduction rules:

$$\alpha(\mathsf{app}(\mathsf{abs}(t), i(u))) \rightsquigarrow \alpha(t)\{* := u\} \qquad \frac{\alpha(T) \rightsquigarrow \alpha(T')}{\alpha(\mathsf{abs}(T)) \rightsquigarrow \alpha(\mathsf{abs}(T'))}$$

$$\frac{\alpha(T) \rightsquigarrow \alpha(T')}{\alpha(\mathsf{app}(T, U)) \rightsquigarrow \alpha(\mathsf{app}(T', U))} \qquad \frac{\alpha(U) \rightsquigarrow \alpha(U')}{\alpha(\mathsf{app}(T, U')) \rightsquigarrow \alpha(\mathsf{app}(T, U'))}$$

5.4.2 Differential lambda calculus

Let us now explain how operational signatures cover differential lambda calculus. This gives an example of a reduction rule with a hypothesis which is not a congruence rule.

Briefly, the syntax is standardly defined by

$$\begin{array}{rcll} t, u & ::= & x \mid \lambda x.t \mid t\ U \mid Ds \cdot t & \text{(terms)} \\ U, V & ::= & \langle t_1, \ldots, t_n \rangle & \text{(multi-terms)}, \end{array}$$

where $\langle t_1, \ldots, t_n \rangle$ denotes a multiset, i.e., the ordering is irrelevant.

This may be specified by the signature $\Sigma_{\Lambda_\partial} = \Theta' + \Theta \times (! \cdot \Theta) + \Theta \times \Theta$ with three operations:

$$\mathsf{abs} : \Theta' \to \Theta \qquad \mathsf{app} : \Theta \times ! \cdot \Theta \to \Theta \qquad \mathsf{Diff} : \Theta \times \Theta \to \Theta,$$

where ! is the free commutative monoid monad, specifiable through an algebraic 2-signature. Then, $\Sigma_{\Lambda_\partial}$ is a presentable signature (see Section 2.8.1), that we can encode alternatively as an algebraic 2-signature, thanks to Corollary 90.

Our now standard first step is to define the state functor T, which is here $T(X) = !X$, as reduction relates multi-terms: indeed, even starting from a simple term, the β-reduction may relate such a term to a multi-term.

The two main reductions of the differential lambda calculus are not that straightforward to handle as they both rely on an intermediate recursive definition:

β-reduction the reduction $(\lambda x.s)\ U \rightsquigarrow s\{x := U\}$ is based on the *multi-term substitution* $s\{x := U\}$ of a variable with a multi-term, defined by induction on s;

∂-reduction the reduction $D(\lambda x.s) \cdot t \rightsquigarrow \lambda x.\left(\frac{\partial s}{\partial x} \cdot t\right)$ is based on the *partial derivative* $\frac{\partial s}{\partial x} \cdot t$ of a term s w.r.t. a variable x along a term t, defined by induction on s.

The difficulty is that we are only able to define the multi-term substitution and the partial derivative for the initial $\Sigma_{\Lambda_\partial}$-monad, taking advantage of the recursion principle. We would not know how to define them for an arbitrary $\Sigma_{\Lambda_\partial}$-monad, which is necessary if we were to define the relevant reduction rules.

Our solution is to use the following logical equivalences:

$$\langle u_1, \dots, u_n \rangle = \frac{\partial s}{\partial x} \cdot t \qquad \Longleftrightarrow \qquad D(\lambda x.s) \cdot t \rightsquigarrow \langle \lambda x.u_1, \dots, \lambda x.u_n \rangle \tag{5.2}$$

$$u = s\{x := U\} \qquad \Longleftrightarrow \qquad (\lambda x.s)\ U \rightsquigarrow u \tag{5.3}$$

Variable For variables, the recursive definitions yield:

$$\frac{\partial y}{\partial x} \cdot t = 0 \qquad y\{x := U\} = y \qquad (x \neq y)$$

$$\frac{\partial x}{\partial x} \cdot t = \lambda x.t \qquad x\{x := U\} = U$$

By the equivalences 5.2 and 5.3, we rephrase these equalities as the following reduction rules:

$$D(\lambda x.y) \cdot t \rightsquigarrow 0 \qquad (\lambda x.y)\ U \rightsquigarrow y \qquad (x \neq y)$$

$$D(\lambda x.x) \cdot t \rightsquigarrow \lambda x.t \qquad (\lambda x.x)\ U \rightsquigarrow U$$

In our framework, we will model this using the following two rules

$$D(\lambda x.u) \cdot T \rightsquigarrow 0 \ (x \notin \mathsf{fv}(u)) \qquad\qquad D(\lambda x.x) \cdot T \rightsquigarrow \lambda x.T.$$

Abstraction The inductive case for abstraction is

$$\frac{\partial \lambda y.s}{\partial x} \cdot U = \lambda y. \left(\frac{\partial s}{\partial x} \cdot U \right) \qquad (\text{for } y \notin \{x\} \cup \mathsf{fv}(U)).$$

We rephrase it as:

$$\frac{z = \dfrac{\partial s}{\partial x} \cdot U \qquad y \notin \{x\} \cup \mathsf{fv}(U)}{\dfrac{\partial \lambda y.s}{\partial x} \cdot U = \lambda y.z}.$$

Using the equivalence 5.2, this yields the rule

$$\frac{D(\lambda x.s) \cdot U \rightsquigarrow \lambda x.V}{D(\lambda x.\lambda y.s) \cdot U \rightsquigarrow \lambda x.\lambda y.V}$$

Application The application case is analogous to the chain rule:

$$\frac{\partial (t\ U)}{\partial x} \cdot V = \left(\frac{\partial t}{\partial x} \cdot V \right) U + \left(Dt \cdot \left(\frac{\partial U}{\partial x} \cdot V \right) \right) U.$$

We rephrase it as:

$$\frac{W = \dfrac{\partial t}{\partial x} \cdot V \qquad W' = \dfrac{\partial U}{\partial x} \cdot V}{\dfrac{\partial (t\ U)}{\partial x} \cdot V = (W\ U) + ((Dt \cdot W')\ U)},$$

Using the equivalence 5.2, this yields the rule

$$\frac{D(\lambda x.t) \cdot V \rightsquigarrow \lambda x.W \qquad D(\lambda x.U) \cdot V \rightsquigarrow \lambda x.W'}{D(\lambda x.t\ U) \cdot V \rightsquigarrow (\lambda x.W\ U) + \lambda x.((Dt \cdot W')\ U)},$$

Differential Finally, the rule for differentiation is

$$\frac{\partial Dt \cdot u}{\partial x} \cdot U = D \left(\frac{\partial t}{\partial x} \cdot U \right) \cdot u + Dt \cdot \left(\frac{\partial u}{\partial x} \cdot U \right).$$

We rephrase it as:

$$\frac{V = \dfrac{\partial t}{\partial x} \cdot U \qquad V' = \dfrac{\partial u}{\partial x} \cdot U}{\dfrac{\partial Dt \cdot u}{\partial x} \cdot U = DV \cdot u + Dt \cdot V'},$$

Using the equivalence 5.2, this yields the rule

$$\frac{D(\lambda x.t) \cdot U \rightsquigarrow \lambda x.V \qquad D(\lambda x.u) \cdot U \rightsquigarrow \lambda x.V'}{D(\lambda x.(Dt \cdot u)) \cdot U \rightsquigarrow \lambda x.(DV \cdot u) + \lambda x.(Dt \cdot V')},$$

We recap now all the reduction rules using the operations as specified by the signature $\Sigma_{\Lambda_\partial}$:

$$\mathsf{Diff}(\mathsf{abs}(\iota(u)), T) \rightsquigarrow 0 \qquad\qquad \mathsf{Diff}(\mathsf{abs}(*), T) \rightsquigarrow \lambda x.\iota(T).$$

where $\iota : \Theta \to \Theta'$ is the canonical inclusion.

CONCLUSION

In this PhD, we have studied signatures specifying monads that we think of as modelling mathematically the well behaved substitution of the informal notion of syntax.

In chapters 2 and 3, we have presented notions of signatures for monads and their models. More precisely, in Chapter 2, we have defined the class of *presentable* signatures, which are quotients of traditional algebraic signatures. Presentable signatures are closed under various operations, including colimits. One of the main results of this chapter says that any presentable signature is effective. Despite the fact that the constructions in Section 2.7 make heavy use of quotients, there is no need to appeal to the axiom of choice. While a previous version of the formalisation did use the axiom of choice to show that certain functors preserve epimorphisms, we managed subsequently to 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).

In Chapter 3, we extend the notion of signature for monads to take into account more general equations. This yields the definition of a 2-signature, as a pair of a 1-signature Σ (that is, a signature in the sense of Chapter 2) and a set of Σ-*equations* that must be satisfied.

Finally, in Chapter 4, we have introduced the notions of reduction monad and reduction signature: they are meant to model syntaxes with a notion of reduction. For each such signature, we define a category of models, equipped with a forgetful functor to the category of reduction monads. We say that a reduction signature is effective if its associated category of models has an initial object; in this case, we say that the reduction monad underlying the initial object is generated by the signature. We identify a simple sufficient condition for a reduction signature to be effective. This chapter is the first step towards a theory for the algebraic specification of programming languages and their semantics. In future work, we aim to generalize our notion of signature to

encompass richer languages and to present a notion of signature that allows for the specification of equalities between reductions (cf. Remark 121).

We anticipate that our work extends to simply-typed languages, by changing the base category Set to a presheaf category Set^T, where T is the set of simple types that we are interested in.

BIBLIOGRAPHY

[Aba+90] M. Abadi et al., “Explicit Substitutions”, *in*: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL ’90, San Francisco, California, USA: ACM, 1990, pp. 31–46, ISBN: 0-89791-343-4, DOI: 10.1145/96709.96712, URL: http://doi.acm.org/10.1145/96709.96712.

[ACU15] Thorsten Altenkirch, James Chapman, and Tarmo Uustalu, “Monads need not be endofunctors”, *in*: *Logical Methods in Computer Science* 11.*1* (2015), DOI: 10.2168/LMCS-11(1:3)2015.

[Adá74] Jirí Adámek, “Free algebras and automata realizations in the language of categories”, eng, *in*: *Commentationes Mathematicae Universitatis Carolinae* 015.*4* (1974), pp. 589–602, URL: http://eudml.org/doc/16649.

[Ahr+19a] Benedikt Ahrens et al., “High-level signatures and initial semantics”, *in*: *CoRR* (2019), Extended version of publication at CSL 2018 (doi), arXiv: 1805.03740v2.

[Ahr+19b] Benedikt Ahrens et al., *Modular specification of monads through higher-order presentations*, 2019, arXiv: 1903.00922 [cs.LO].

[Ahr16] Benedikt Ahrens, “Modules over relative monads for syntax and semantics”, *in*: *Mathematical Structures in Computer Science* 26 (1 2016), pp. 3–37, ISSN: 1469-8072, DOI: 10.1017/S0960129514000103.

[AL17] Benedikt Ahrens and Peter LeFanu Lumsdaine, “Displayed Categories”, *in*: *2nd International Conference on Formal Structures for Computation and Deduction*, ed. by Dale Miller, vol. 84, Leibniz International Proceedings in Informatics, Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 5:1–5:16, ISBN: 978-3-95977-047-7, DOI: 10.4230/LIPIcs.FSCD.2017.5.

[AMM18] Benedikt Ahrens, Ralph Matthes, and Anders Mörtberg, “From Signatures to Monads in UniMath”, *in*: *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”, *in*: *Theor. Comput. Sci.* 311.*1-3* (Jan. 2004), pp. 257–283, ISSN: 0304-3975, DOI: 10.1016/S0304-3975(03)00378-5, URL: http://dx.doi.org/10.1016/S0304-3975(03)00378-5.

[AR99] Thorsten Altenkirch and Bernhard Reus, “Monadic Presentations of Lambda Terms Using Generalized Inductive Types”, *in*: *Computer Science Logic, 13th International Workshop, CSL '99, 8th Annual Conference of the EACSL, Madrid, Spain, September 20-25, 1999, Proceedings*, ed. by Jörg Flum and Mario Rodríguez-Artalejo, vol. 1683, Lecture Notes in Computer Science, Springer, 1999, pp. 453–468, ISBN: 3-540-66536-6, DOI: 10.1007/3-540-48168-0_32.

[Bar70] Michael Barr, “Coequalizers and free triples”, *in*: *Mathematische Zeitschrift* 116.*4* (1970), pp. 307–322, ISSN: 1432-1823, DOI: 10.1007/BF01111838.

[BM97] Richard S. Bird and Oege de Moor, *Algebra of programming*, Prentice Hall International series in computer science, Prentice Hall, 1997, pp. I–XIV, 1–295, ISBN: 978-0-13-507245-5.

[Bor94] Francis Borceux, *Handbook of Categorical Algebra*, vol. 2, Encyclopedia of Mathematics and its Applications, Cambridge University Press, 1994, DOI: 10.1017/CBO9780511525865.

[BP99] Richard S. Bird and Ross Paterson, “Generalised folds for nested datatypes”, *in*: *Formal Asp. Comput.* 11.*2* (1999), pp. 200–222, DOI: 10.1007/s001650050047.

[Bra14] Martin Brandenburg, “Tensor categorical foundations of algebraic geometry”, PhD thesis, Universität Münster, 2014, arXiv: 1410.1716.

[Clo10] Ranald Clouston, “Binding in Nominal Equational Logic”, *in*: *Electr. Notes Theor. Comput. Sci.* 265 (2010), pp. 259–276, DOI: 10.1016/j.entcs.2010.08.016.

[CoqDev19] The Coq development team, *The Coq Proof Assistant, version 8.9*, Version 8.9, 2019, URL: http://coq.inria.fr.

[ER03] Thomas Ehrhard and Laurent Regnier, “The differential lambda-calculus”, *in*: *Theor. Comput. Sci.* 309.*1-3* (2003), pp. 1–41, DOI: 10.1016/S0304-3975(03)00392-X, URL: https://doi.org/10.1016/S0304-3975(03)00392-X.

[FG10] Maribel Fernández and Murdoch J. Gabbay, “Closed nominal rewriting and efficiently computable nominal algebra equality”, *in*: *Proceedings 5th International Workshop on Logical Frameworks and Meta-languages: Theory and Practice, LFMTP 2010, Edinburgh, UK, 14th July 2010.* Ed. by Karl Crary and Marino Miculan, vol. 34, EPTCS, 2010, pp. 37–51, DOI: 10.4204/EPTCS.34.5.

[FH09] Marcelo P. Fiore and Chung-Kil Hur, “On the construction of free algebras for equational systems”, *in*: *Theor. Comput. Sci.* 410.*18* (2009), pp. 1704–1729, DOI: 10.1016/j.tcs.2008.12.052.

[FH10] Marcelo P. Fiore and Chung-Kil Hur, “Second-Order Equational Logic (Extended Abstract)”, *in*: *CSL*, ed. by Anuj Dawar and Helmut Veith, vol. 6247, Lecture Notes in Computer Science, Springer, 2010, pp. 320–335, ISBN: 978-3-642-15204-7, DOI: 10.1007/978-3-642-15205-4_26.

[FH13] Marcelo P. Fiore and Makoto 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*, IEEE Computer Society, 2013, pp. 520–529, ISBN: 978-1-4799-0413-6, DOI: 10.1109/LICS.2013.59.

[Fio08] Marcelo P. Fiore, “Second-Order and Dependently-Sorted Abstract Syntax”, *in*: *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science*, IEEE Computer Society, 2008, pp. 57–68, ISBN: 978-0-7695-3183-0, DOI: 10.1109/LICS.2008.38.

[FM10] Marcelo P. Fiore and Ola Mahmoud, “Second-Order Algebraic Theories (Extended Abstract)”, *in*: *MFCS*, ed. by Petr Hlinený and Antonín Kucera, vol. 6281, Lecture Notes in Computer Science, Springer, 2010, pp. 368–380, ISBN: 978-3-642-15154-5, DOI: 10.1007/978-3-642-15155-2_33.

[FPT99] Marcelo P. Fiore, Gordon D. Plotkin, and Daniele Turi, “Abstract Syntax and Variable Binding”, *in*: *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, 1999, pp. 193–202, DOI: 10.1109/LICS.1999.782615.

[FS17] Marcelo P. Fiore and Philip Saville, “List Objects with Algebraic Structure”, *in*: *2nd International Conference on Formal Structures for Computation and Deduction (FSCD 2017)*, ed. by Dale Miller, vol. 84, Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 16:1–16:18, ISBN: 978-3-95977-047-7, DOI: 10.4230/LIPIcs.FSCD.2017.16.

[Gir87] Jean-Yves Girard, “Linear Logic”, *in*: *Theor. Comput. Sci.* 50.*1* (Jan. 1987), pp. 1–102, ISSN: 0304-3975, DOI: 10.1016/0304-3975(87)90045-4.

[GP99] Murdoch J. Gabbay and Andrew M. Pitts, “A New Approach to Abstract Syntax Involving Binders”, *in*: *14th Annual Symposium on Logic in Computer Science*, Washington, DC, USA: IEEE Computer Society Press, 1999, pp. 214–224, ISBN: 0-7695-0158-3, DOI: 10.1109/LICS.1999.782617.

[Gra12] Bernhard Gramlich, “Modularity in term rewriting revisited”, *in*: *Theoretical Computer Science* 464 (2012), New Directions in Rewriting (Honoring the 60th Birthday of Yoshihito Toyama), pp. 3 –19, ISSN: 0304-3975, DOI: 10.1016/j.tcs.2012.09.008.

[GU03] Neil Ghani and Tarmo Uustalu, “Explicit substitutions and higher-order syntax”, *in*: *MERLIN ’03: Proceedings of the 2003 ACM SIGPLAN workshop on Mechanized reasoning about languages with variable binding*, Uppsala, Sweden: ACM Press, 2003, pp. 1–7, ISBN: 1-58113-800-8.

[GUH06] Neil Ghani, Tarmo Uustalu, and Makoto Hamana, “Explicit substitutions and higher-order syntax”, *in*: *Higher-Order and Symbolic Computation* 19.*2-3* (2006), pp. 263–282, DOI: 10.1007/s10990-006-8748-4.

[Ham03] Makoto 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, Uppsala, Sweden: ACM, 2003, pp. 148–159, ISBN: 1-58113-705-2, DOI: 10.1145/888251.888266.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin, “A Framework for Defining Logics”, *in*: *J. ACM* 40.*1* (Jan. 1993), pp. 143–184, ISSN: 0004-5411, DOI: 10.1145/138027.138060.

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

[HM07] André Hirschowitz and Marco Maggesi, “Modules over Monads and Linearity”, *in*: *WoLLIC*, ed. by D. Leivant and R. J. G. B. de Queiroz, vol. 4576, Lecture Notes in Computer Science, Springer, 2007, pp. 218–237, ISBN: 978-3-540-73443-7, DOI: 10.1007/978-3-540-73445-1_16.

[HM10] André Hirschowitz and Marco Maggesi, “Modules over monads and initial semantics”, *in*: *Information and Computation* 208.*5* (2010), Special Issue: 14th Workshop on Logic, Language, Information and Computation (WoLLIC 2007), pp. 545–564, DOI: 10.1016/j.ic.2009.07.003.

[HM12] André Hirschowitz and Marco Maggesi, “Initial Semantics for Strengthened Signatures”, *in*: *Proceedings 8th Workshop on Fixed Points in Computer Science, FICS 2012, Tallinn, Estonia, 24th March 2012.* Ed. by Dale Miller and Zoltán Ésik, vol. 77, EPTCS, 2012, pp. 31–38, DOI: 10.4204/EPTCS.77.5, URL: https://doi.org/10.4204/EPTCS.77.5.

[Hof99] Martin Hofmann, “Semantical Analysis of Higher-Order Abstract Syntax”, *in*: *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, IEEE Computer Society, 1999, pp. 204–213, ISBN: 0-7695-0158-3, DOI: 10.1109/LICS.1999.782616.

[HP07] Martin Hyland and John Power, “The category theoretic understanding of universal algebra: Lawvere theories and monads”, *in*: *Electronic Notes in Theoretical Computer Science*, Electron. Notes Theor. Comput. Sci. 172 (2007), pp. 437–458, DOI: 10.1016/j.entcs.2007.02.019.

[JG07] Patricia Johann and Neil Ghani, “Initial Algebra Semantics Is Enough!”, *in*: *Typed Lambda Calculi and Applications, 8th International Conference, TLCA 2007, Paris, France, June 26-28, 2007, Proceedings*, 2007, pp. 207–222, DOI: 10.1007/978-3-540-73228-0_16.

[JGW78] J.W. Thatcher J.A. Goguen and E.G. Wagner, “An initial algebra approach to the specification, correctness and implementation of abstract data types”, *in*: *Current Trends in Programming Methodology, IV: Data Structuring*, ed. by R. Yeh, Prentice-Hall, 1978, pp. 80–144.

[Kes09] Delia Kesner, “A Theory of Explicit Substitutions with Safe and Full Composition”, *in*: *Logical Methods in Computer Science* 5.*3* (2009), DOI: 10.2168/LMCS-5(3:1)2009.

[KP10] Alexander Kurz and Daniela Petrisan, “On universal algebra over nominal sets”, *in*: *Mathematical Structures in Computer Science* 20.*2* (2010), pp. 285–318, DOI: 10.1017/S0960129509990399.

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

[LG97] Christoph Lüth and Neil Ghani, “Monads and Modular Term Rewriting”, *in*: *Category Theory and Computer Science, 7th International Conference, CTCS '97*, ed. by Eugenio Moggi and Giuseppe Rosolini, vol. 1290, Lecture Notes in Computer Science, Springer, 1997, pp. 69–86, ISBN: 3-540-63455-X, DOI: 10.1007/BFb0026982.

[Man76] Ernest Manes, *Algebraic Theories*, vol. 26, Graduate Texts in Mathematics, Springer, 1976.

[ML98] Saunders Mac Lane, *Categories for the working mathematician*, Second, vol. 5, Graduate Texts in Mathematics, New York: Springer-Verlag, 1998, pp. xii+314, ISBN: 0-387-98403-8.

[MU04] Ralph Matthes and Tarmo Uustalu, “Substitution in non-wellfounded syntax with variable binding”, *in*: *Theor. Comput. Sci.* 327.*1-2* (2004), pp. 155–174, ISSN: 0304-3975, DOI: 10.1016/j.tcs.2004.07.025.

[Pow07] A. John Power, “Abstract Syntax: Substitution and Binders: Invited Address”, *in*: *Electr. Notes Theor. Comput. Sci.* 173 (2007), pp. 3–16, DOI: 10.1016/j.entcs.2007.02.024.

[See00] R. A. G. Seely, “Bart Jacobs. Categorical logic and type theory. Studies in logic and the foundations of mathematics, vol. 141. Elsevier, Amsterdam etc. 1999, xvii 760 pp.”, *in*: *Bulletin of Symbolic Logic* 6.*2* (2000), 225–229, DOI: 10.2307/421214.

[Sel08] Peter Selinger, “Lecture notes on the lambda calculus”, *in*: *CoRR* abs/0804.3434 (2008), arXiv: 0804.3434, URL: http://arxiv.org/abs/0804.3434.

[Sta13] Sam Staton, “An Algebraic Presentation of Predicate Logic (Extended Abstract)”, *in*: *Foundations of Software Science and Computation Structures - 16th International Conference, FOSSACS 2013*, ed. by Frank Pfenning, vol. 7794, Lecture Notes in Computer Science, Springer, 2013, pp. 401–417, ISBN: 978-3-642-37074-8, DOI: 10.1007/978-3-642-37075-5_26.

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

[VK11] Jiří Velebil and Alexander Kurz, “Equational presentations of functors and monads”, *in*: *Mathematical Structures in Computer Science* 21.*2* (2011), pp. 363–381, DOI: 10.1017/S0960129510000575.

UNIVERSITE BRETAGNE LOIRE MATHSTIC

UNIVERSITÉ SAVOIE MONT BLANC

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Titre : Vers une approche impartiale pour spécifier, implémenter, et prouver des propriétés des langages de programmation

Mot clés : monades, semantique initiale, syntaxe

Résumé : Cette thèse traite de la spécification et la construction de syntaxes avec des équations et des réductions. Nous travaillons avec une notion générale de “signature” pour spécifier une syntaxe, définie comme l’objet initial dans une catégorie appropriée de modèles. Cette caractérisation, dans l’esprit de la sémantique initiale, donne une justification du principe de récursion.

Les languages avec liaisons, telles que le lambda calcul pur, sont spécifiable par les signatures algébriques classiques. Les premières extensions de syntaxes avec des équations que nous considérons sont des “quotients” de ces signatures algébriques. Ils permettent, par exemple, de spécifier une opération commutative binaire. Cependant, certaines équations, comme l’associativité, semblent hors d’atteinte. Ceci motive la notion de 2-signature qui complète la définition précédente avec la donnée d’un ensemble d’équations. Nous identifions la classe des “2-signatures algébriques” pour lesquelles l’existence de la syntaxe associée est garantie.

Quotienter la syntaxe par les règle de réduction peut être trop brutal. C’est pourquoi nous introduisons la notion de signatures de réduction, qui spécifient des monades de réduction, telles que le lambda calcul avec β-réduction.

Title: Towards an unbiased approach to specify, implement, and prove properties on programming languages

Keywords: monads, initial semantics, syntax

Abstract: This thesis deals with the specification and construction of syntaxes with equations and reductions. 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. But some equations, such as associativity, seem to remain out of reach. We thus introduce the notion of 2-signature, consisting in two parts: a specification 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.

Sometimes, quotienting the syntax by the reduction rules is too harsh. This is why we introduce the notion of reduction signatures, that specify reduction monads, such as the lambda calculus with β-reduction.