

# THÈSE DE DOCTORAT DE

L'ÉCOLE NATIONALE SUPÉRIEURE 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**

**Signatures, syntax, semantics**

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)
Examineurs :	Benedikt Ahrens	Birmingham Fellow, University of Birmingham
	Delia Kesner	Professeur, Université Paris Diderot
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

---

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Résumé long (en français) . . . . .	9
1.1.1	Présentation naïve de la syntaxe du lambda calcul . . . . .	10
1.1.2	La monade du lambda calcul . . . . .	12
1.1.3	Les constructions sont des morphismes de modules . . . . .	13
1.1.4	Récursion et initialité . . . . .	15
1.1.5	Signatures et modèles (Chapitre 2) . . . . .	16
1.1.6	Syntaxes avec équations (chapitres 3 et 4) . . . . .	19
1.1.7	Sémantique (chapitres 5 and 6) . . . . .	20
1.1.8	Formalisation . . . . .	23
1.2	Long summary . . . . .	25
1.2.1	Naive presentation of the lambda calculus syntax . . . . .	27
1.2.2	The lambda calculus monad . . . . .	28
1.2.3	Constructions are morphisms of modules . . . . .	29
1.2.4	Recursion and initiality . . . . .	31
1.2.5	Signatures and modèles (Chapter 2) . . . . .	33
1.2.6	Syntaxes with equations (Chapters 3 and 4) . . . . .	35
1.2.7	Semantics (Chapters 5 and 6) . . . . .	36
1.2.8	Formalisation . . . . .	39
1.3	Synopsis . . . . .	41
1.4	Computer-checked formalization . . . . .	42
1.5	Related work . . . . .	42
1.5.1	Syntax and monads . . . . .	42
1.5.2	Syntax with equations . . . . .	43
1.5.3	Syntax with reductions . . . . .	45
<b>2</b>	<b>General signatures</b>	<b>47</b>
2.1	Arities over a category . . . . .	47
2.1.1	General arities . . . . .	48

## TABLE OF CONTENTS

---

2.1.2	Equational arities . . . . .	49
2.1.3	Family of arities and their models . . . . .	50
2.2	Signatures for a category . . . . .	50
2.3	Pulling back arities and signatures . . . . .	51
2.3.1	Pullback of arities . . . . .	51
2.3.2	Pullback of signatures . . . . .	52
<b>I</b>	<b>Syntax</b>	<b>54</b>
<b>3</b>	<b>Presentable signatures for monads</b>	<b>55</b>
3.1	Categories of modules over monads . . . . .	56
3.1.1	Modules over monads . . . . .	56
3.1.2	The total category of modules . . . . .	58
3.1.3	Derivation . . . . .	58
3.2	The category of signatures . . . . .	60
3.3	Categories of models . . . . .	62
3.4	Syntax . . . . .	63
3.4.1	Representations of a signature . . . . .	64
3.4.2	Modularity . . . . .	66
3.5	Recursion . . . . .	67
3.5.1	Example: Translation of intuitionistic logic into linear logic . . . . .	67
3.5.2	Example: Computing the set of free variables . . . . .	68
3.5.3	Example: Computing the size of a term . . . . .	69
3.5.4	Example: Counting the number of redexes . . . . .	70
3.6	Presentations of signatures and syntaxes . . . . .	71
3.7	Proof of Theorem 54 . . . . .	73
3.8	Constructions of presentable signatures . . . . .	76
3.8.1	Post-composition with a presentable functor . . . . .	76
3.8.2	Example: Adding a syntactic closure operator . . . . .	78
3.8.3	Example: Adding an explicit substitution . . . . .	79
3.8.4	Example: Adding a coherent fixed-point operator . . . . .	82
<b>4</b>	<b>Algebraic 2-signatures for monads</b>	<b>87</b>
4.1	Introduction . . . . .	88

4.2	2-Signatures and their models . . . . .	90
4.2.1	Equations . . . . .	90
4.2.2	2-signatures and their models . . . . .	92
4.2.3	Modularity for 2-signatures . . . . .	93
4.2.4	Initial Semantics for 2-Signatures . . . . .	96
4.3	Proof of Theorem 104 . . . . .	97
4.4	Examples of algebraic 2-signatures . . . . .	100
4.4.1	Monoids . . . . .	100
4.4.2	Colimits of algebraic 2-signatures . . . . .	101
4.4.3	Algebraic theories . . . . .	103
4.4.4	Fixpoint operator . . . . .	104
4.5	Recursion . . . . .	105
4.5.1	Principle of recursion . . . . .	105
4.5.2	Translation of lambda calculus with fixpoint to lambda calculus . . . . .	106
<b>II</b>	<b>Semantics</b>	<b>107</b>
<b>5</b>	<b>Reduction monads and their signatures</b>	<b>108</b>
5.1	Introduction . . . . .	109
5.1.1	Terminology and notations . . . . .	109
5.1.2	Plan of the chapter . . . . .	110
5.2	Reduction monads . . . . .	110
5.2.1	The category of reduction monads . . . . .	110
5.2.2	Examples of reduction monads . . . . .	112
5.3	Reduction rules . . . . .	114
5.3.1	Example: congruence rule for application . . . . .	115
5.3.2	Definition of reduction rules . . . . .	116
5.3.3	Reduction $\Sigma$ -monads . . . . .	117
5.3.4	Action of a reduction rule . . . . .	118
5.3.5	Protocol for specifying reduction rules . . . . .	120
5.3.6	Examples of reduction rules . . . . .	120
5.4	Signatures for reduction monads and Initiality . . . . .	121
5.4.1	Signatures and their models . . . . .	122
5.4.2	The functors $\text{Hyp}_{\mathcal{A}}$ and $\text{Con}_{\mathcal{A}}$ . . . . .	122

## TABLE OF CONTENTS

---

5.4.3	The main result . . . . .	123
5.5	Proof of Theorem 155 . . . . .	125
5.5.1	Normalizing reduction rules . . . . .	125
5.5.2	Models as vertical algebras . . . . .	127
5.5.3	Effectivity . . . . .	130
5.6	Example: Lambda calculus with explicit substitutions . . . . .	132
5.6.1	Signature for the monad of the lambda calculus with explicit substitutions . . . . .	133
5.6.2	Reduction rules for lambda calculus with explicit substitutions . . . . .	134
5.7	Recursion . . . . .	137
5.7.1	Recursion principle for effective signatures . . . . .	137
5.7.2	Translation of lambda calculus with fixpoint to lambda calculus . . . . .	138
5.7.3	Translation of lambda calculus with explicit substitutions into lambda calculus with $\beta$ -reduction . . . . .	139
5.7.4	Induction principle . . . . .	140
<b>6</b>	<b>Operational monads and their signatures</b>	<b>145</b>
6.1	Operational monads . . . . .	146
6.1.1	Operational monads . . . . .	146
6.1.2	Examples of operational monads . . . . .	148
6.2	$\mathbb{S}$ -reduction rules . . . . .	150
6.2.1	Definition of $\mathbb{S}$ -reduction rules . . . . .	150
6.2.2	$\mathbb{S}$ -Reduction $\Sigma$ -monads . . . . .	151
6.2.3	Action of a $\mathbb{S}$ -reduction rule . . . . .	152
6.3	Signatures for operational monads and Initiality . . . . .	154
6.3.1	Operational signatures and their models . . . . .	154
6.3.2	The functors $\text{Hyp}_{\mathcal{A}}$ and $\text{Con}_{\mathcal{A}}$ . . . . .	155
6.3.3	The main result . . . . .	155
6.3.4	Specifying the state functors . . . . .	157
6.4	Examples of operational signatures . . . . .	161
6.4.1	Call-by-value lambda calculus . . . . .	161
6.4.2	Call-by-value lambda calculus with big-step operational semantics . . . . .	162
6.4.3	$\pi$ -calculus . . . . .	162
	<b>Conclusion</b>	<b>165</b>

**Bibliography**

**167**





# 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 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.

En ce qui concerne la mathématisation de la syntaxe, la spécification de structures algébriques avec variables liées est un enjeu majeur. Deux lignes principales de recherche sont en concurrence : les *ensembles nominaux* [GP99] et les *algèbres de substitution* [FPT99]. Dans cette thèse, nous explorons une variante des algèbres de substitution, proposée par [HM07 ; HM10], qui s'appuie sur la notion de *module sur une monade*.

Dans le chapitre 5, nous définissons les *monades de réduction* : elles constituent notre contrepartie mathématique de la notion de langage de programmation. Nous proposons alors un protocole fondé sur une signature à trois niveaux pour les spécifier :

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  $0 + a \rightsquigarrow a$ , par des règles schématiques.

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 5.6 une spécification du lambda calcul avec substitution explicite décrit dans [Kes09]. Nous définissons dans le chapitre 6 la notion de *monade opérationnelle*, généralisant celle de monade de réduction, afin de traiter de nouveaux exemples, comme le lambda calcul en appel par valeurs avec sa sémantique à grands pas.

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, modélisée dans le cadre des monades de réduction par un ensemble de réductions entre chaque paire de 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 occurrences de  $x$  dans  $t$  sont alors qualifiées de liées. Les occurrences de variables qui ne sont pas liées sont dites *libres*<sup>1</sup>.

Ici, les termes  $\lambda x.t$  et  $\lambda y.t'$  sont dits  *$\alpha$ -équivalents*. Plus généralement, deux termes sont  *$\alpha$ -équivalents* si l'on peut renommer les variables liées de l'un pour obtenir l'autre terme. La définition précise de la relation d' *$\alpha$ -équivalence* requiert quelques précautions. Par exemple, dans le cas précédent, il est sous-entendu que la variable  $y$  n'apparaît 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  $t\{x := u\}$ . 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 la  *$\beta$ -équation*

$$(\lambda x.t) u = t\{x := u\}. \quad (1.1)$$

Cette substitution dite *unaire* est un cas particulier de l'opération de *substitution parallèle*  $t\{x \mapsto u_x\}$ , qui remplace simultanément toutes les variables d'un terme  $t$  par un terme correspondant.

La substitution ne remplace que les occurrences libres d'une variable, afin de préserver la propriété suivante : étant donné deux termes  $\lambda x.t$  et  $\lambda y.t'$  supposés  *$\alpha$ -*

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

équivalents, substituer le même terme à la même variable dans chacun d'eux fournit deux termes  $\alpha$ -équivalents. Si la variable concernée est identique à la variable abstraite, le terme  $(\lambda x.t)\{x := u\}$  est donc tout simplement égal à  $\lambda x.t$  : par exemple  $(\lambda x.x)\{x := u\} = \lambda x.x$  est bien  $\alpha$ -équivalent à  $(\lambda y.y)\{x := u\} = \lambda y.y$ .

### 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  $\alpha$ -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 l'ensemble  $X$ . Remarquons qu'un terme  $t \in L(X)$  se retrouve également dans  $L(Y)$  pour toute 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 éléments de n'importe quel ensemble  $Y$  qui contient  $X$ .

Toute variable est en particulier un terme valide ; il y a donc une inclusion  $\text{var}_X : X \rightarrow L(X)$  pour tout ensemble  $X$ . D'autre part, si l'on se donne pour toute variable  $x \in X$  un terme  $u_x$  dont les variables libres sont choisies dans  $Y$ , nous obtenons, à partir de n'importe quel terme  $t \in L(X)$ , un terme  $t\{x \mapsto u_x\} \in L(Y)$ . Cette opération de substitution parallèle vérifie les propriétés suivantes :

- chaque variable est remplacée par le terme adéquat :

$$x'\{x \mapsto u_x\} = u_{x'}$$

- la substitution identité est neutre :

$$t\{x \mapsto x\} = t$$

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

$$t\{x \mapsto u_x\}\{y \mapsto v_y\} = t\{x \mapsto u_x\{y \mapsto v_y\}\}$$

L'inclusion des variables dans les termes et l'opération de substitution parallèle 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.

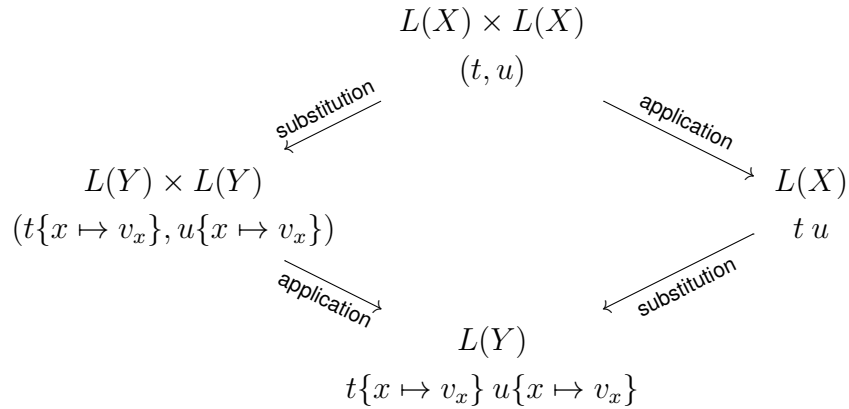
### 1.1.3 Les constructions sont des morphismes de modules

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

La compatibilité de l'application avec la substitution se traduit par la commutation

$$(t u)\{x \mapsto v_x\} = t\{x \mapsto v_x\} u\{x \mapsto v_x\}$$

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  $(L(X) \times L(X))_X$  :

$$(t, u)\{x \mapsto v_x\} = (t\{x \mapsto v_x\}, u\{x \mapsto v_x\})$$

Cette substitution vérifie les propriétés suivantes :

- la substitution identité est neutre :

$$(t, u)\{x \mapsto x\} = (t, u)$$

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

$$(t, u)\{x \mapsto u_x\}\{y \mapsto v_y\} = (t, u)\{x \mapsto u_x\{y \mapsto v_y\}\}$$

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

$$L(X) \times L(X) \rightarrow L(X)$$

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

$$L(X \amalg \{\star\}) \rightarrow L(X)$$

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 :

$$(\lambda \star .t)\{x \mapsto u_x\} = \lambda \star . \left( t \left\{ x \mapsto \begin{cases} \star & \text{si } x = \star \\ u_x & \text{sinon.} \end{cases} \right\} \right)$$

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 sont compatibles 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  $uv$ , 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 3.5.2).

Considérons la monade  $\mathcal{P}$  qui associe à  $X$  l'ensemble  $\mathcal{P}(X)$  de ses parties : une variable  $x \in X$  induit un “terme”  $\{x\} \in \mathcal{P}(X)$  ; la substitution  $t\{x \mapsto u_x\}$  est calculée par la réunion  $\bigcup_{z \in t} u_z$ . 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  $\mathcal{P}(X \amalg \{\star\}) \rightarrow \mathcal{P}(X)$  est donnée par  $t \mapsto t \cap X$ , ou, de manière équivalente, par  $t \mapsto t \setminus \{\star\}$ . Comme nous

l'expliquerons dans la section suivante, ces constructions font de la monade  $P$  un *modèle de la signature du lambda calcul*.

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

- $\text{free}$  préserve les variables :

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

(rappelons que pour la monade  $\mathcal{P}$ , la variable  $x$  est vue comme le sous-ensemble  $\{x\}$ )

- $\text{free}$  préserve la substitution :

$$\text{free}_Y(t\{x \mapsto u_x\}) = \bigcup_{z \in \text{free}_X(t)} \text{free}_Y(u_z)$$

- $\text{free}$  préserve l'application :

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

- $\text{free}$  préserve l'abstraction :

$$\text{free}(\lambda x.t) = \text{free}(t) \setminus \{x\}$$

Les deux premiers points caractérisent  $\text{free}$  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  $\text{free}$  est alors un *morphisme de modèles* de cette signature, grâce aux deux derniers points.

### 1.1.5 Signatures et modèles (Chapitre 2)

Caractériser un objet d'un certain type (i.e., d'une certaine catégorie  $C$ ) par une propriété d'initialité est l'objectif de ce que l'on appelle *sémantique initiale* ou *spécification*



*algébrique* [JGW78], popularisés par [BM97]. Dans cette thèse, nous nous intéressons à la spécification des objets de la catégorie des monades (chapitres 3 et 4), de la catégorie des *monades de réduction* (chapitre 5), et de la catégorie des *monades opérationnelles* (chapitre 6), notre objectif étant de définir un *langage formel*<sup>2</sup>, modélisé par un objet d'une catégorie adéquate.

La méthodologie générale de la sémantique initiale peut être décrite selon les étapes suivantes :

1. Introduire une notion de signature (pour la catégorie  $C$ ).
2. Construire une notion de modèle associée, s'organisant en une catégorie munie d'un foncteur vers la catégorie  $C$ .
3. Définir l'objet spécifié par la signature comme le modèle initial, s'il existe (la signature est alors dite *effective*).
4. Trouver une condition suffisante pour qu'une signature soit effective<sup>3</sup>.

Les modèles d'une signature constituent le domaine atteint par le principe de récurrence, lequel est induit par l'initialité de l'objet spécifié par la signature.

Dans le chapitre 2, nous définissons une notion générale de signature permettant de caractériser un objet d'une catégorie arbitraire  $C$ . Une telle signature est donnée par une liste de familles d'*arités* spécifiant des opérations ou des équations. Les chapitres suivants en fournissent divers cas particuliers, de la spécification de la syntaxe à la spécification de la sémantique.

Concentrons-nous sur l'aspect syntaxique, mathématiquement représenté par le concept de monade. Nous identifions une première classe de signatures spécifiant des monades : la catégorie de modèles d'une signature est munie d'un foncteur vers la catégorie des monades. Considérons l'exemple de la syntaxe du lambda calcul. Dans la section précédente, nous l'avons caractérisée 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

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.

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

d'une opération binaire, c'est-à-dire d'un morphisme de modules  $\text{app}^R : R \times R \rightarrow R$ , et d'un morphisme de modules  $\text{abs}^R : R' \rightarrow R$ . De manière équivalente, c'est une monade  $R$  avec un morphisme de modules de  $(R \times R) \amalg R'$  vers  $R$ .

La signature  $\Sigma_{\text{LC}}$  du lambda calcul associe à toute monade  $R$  le module  $(R \times R) \amalg R'$  sur  $R$ . Plus généralement, une signature  $\Sigma$  associe à toute monade  $R$  un module  $\Sigma(R)$ . Un *modèle*  $(R, \rho)$  (parfois noté simplement  $R$ ) de  $\Sigma$  est alors une monade  $R$  munie d'un morphisme de modules  $\rho : \Sigma(R) \rightarrow R$ . Ainsi, la monade des parties  $\mathcal{P}$  et la monade du lambda calcul LC induisent des modèles de  $\Sigma_{\text{LC}}$ . En fait, le lambda calcul est le modèle initial : si  $R$  est un modèle, alors il existe un unique morphisme de monades  $f : L \rightarrow R$  qui préserve l'opération binaire et l'abstraction.

Plus généralement, la syntaxe spécifiée par une signature  $\Sigma$  est le modèle initial  $(S, \sigma)$  au sens suivant : étant donné un modèle  $(R, \rho)$ , il existe un unique morphisme de monades  $f : S \rightarrow R$  qui préserve la structure de modèle, i.e., vérifiant

$$f(\sigma(t)) = \rho(f(t)) \quad (1.2)$$

Un tel morphisme de monades définit ce que nous appelons un *morphisme de modèles* entre  $(S, \sigma)$  et  $(R, \rho)$ . Notons que le membre de droite de l'équation 1.2 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,  $\Sigma(S)_X = S(X) \times S(X)$ . Donc  $t$  est 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  $\Sigma$  vienne avec une action "fonctorielle" : tout morphisme de monades  $f : R \rightarrow T$  induit un morphisme de modules<sup>4</sup>  $\Sigma(R) \rightarrow \Sigma(T)$  sur la monade  $R$  que nous notons abusivement  $f$ , 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<sup>5</sup>. C'est néanmoins le cas de toute signature que nous appelons *algébrique*, qui spécifie une syntaxe disposant d'un ensemble d'opérations n-aires, dont certaines lient des variables dans leurs arguments. La signature du lambda calcul est algébrique : l'application est une opération binaire classique, tandis que l'abstraction est une opération unaire liant une seule variable dans son unique argument.

---

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

5. La signature du contre-exemple 46 associe à toute monade  $R$  le module  $(\mathcal{P}(R(X)))_X$ .

### 1.1.6 Syntaxes avec équations (chapitres 3 et 4)

Dans cette thèse, nous abordons la spécification de langages de programmation vérifiant des équations syntaxiques, comme par exemple le lambda calcul modulo  $\beta$ -équation (1.1).

Dans le chapitre 3, nous étudions les signatures que nous appelons *présentables* : ce sont, en quelque sorte, des quotients de signatures algébriques. Nous montrons qu'elles sont effectives (Théorème 54). 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 3.8.1). Pour cela, il suffit de remarquer que la donnée d'une telle opération est équivalente à la donnée d'une opération prenant en argument une paire non ordonnée de termes. Décrivons maintenant la signature présentable  $\Sigma_{\text{comm-bin}}$  associée : il s'agit d'un quotient de la signature algébrique  $\Sigma_{\text{bin}}$  d'une opération binaire qui associe à toute monade  $R$  le module  $R \times R$ . Plus précisément, à toute monade  $R$ , la signature  $\Sigma_{\text{comm-bin}}$  associe le module  $(S^2 R(X))_X$ , où  $S^2 R(X)$  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  $b : R \times R \rightarrow R$  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 4, nous donnons une définition d'équation généralisant cet exemple pour les modèles d'une signature  $\Sigma$  quelconque : il s'agit de la donnée,

- pour chaque modèle  $R$  de la signature  $\Sigma$ , de deux morphismes de modules  $e_R, e'_R$  de même domaine  $A_R$  et codomaine  $B_R$  (dans l'exemple ci-dessus,  $e_R, e'_R : R \times R \times R \rightarrow R$  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  $f : R \rightarrow S$ , de deux morphismes de modules  $A_f : A_R \rightarrow A_S$  et  $B_f : B_R \rightarrow B_S$  tels que les diagrammes suivants commutent :

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

Ces données sont soumises comme d'habitude à une condition additionnelle de fonctorialité. On dit qu'un modèle  $R$  de la signature  $\Sigma$  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 introduisons les 2-signatures algébriques : il s'agit de 2-signatures composées d'une signature algébrique au sens précédent, et d'un ensemble d'équations dites *élémentaires* (Définition 101). Techniquement, il s'agit d'équations dont l'action fonctorielle du domaine envoie des morphismes surjectifs de monades sur des morphismes surjectifs de foncteurs, et dont le codomaine est de la forme  $R \mapsto R' \cdots'$ . Tout exemple de signature présentable que nous considérons dans le chapitre 3 peut être reformulé en tant que 2-signature algébrique, induisant la même catégorie de modèles (à isomorphisme près). Enfin, nous montrons l'existence d'un 2-modèle initial pour toute 2-signature algébrique (Théorème 104).

### 1.1.7 Sémantique (chapitres 5 and 6)

Il est possible de spécifier par une 2-signature la syntaxe du lambda calcul quotientée par la  $\beta$ -équation (1.1) : les termes  $(\lambda x.t)u$  et  $t\{x := u\}$  sont ainsi égalisés. Cependant, cette équation est habituellement orientée, et considérée comme une étape d'exécution lorsque l'on considère le lambda calcul comme un langage de programmation fonctionnel. A ce titre, il est plus adéquat d'intégrer la  $\beta$ -réduction à la sémantique du langage, plutôt que d'imposer la  $\beta$ -équation dans la syntaxe. Ceci motive la notion de *monade de réduction*, que nous introduisons dans le chapitre 5, 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  $u$ , cette substitution associe une réduction  $m\{x \mapsto v_x\}$  entre  $t\{x \mapsto v_x\}$  et  $u\{x \mapsto v_x\}$ . 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  $\text{app}$  :

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

Cette règle se décompose en trois paires de termes : les hypothèses  $(T, T')$  et  $(U, U')$ , et la conclusion  $(\text{app}(T, U), \text{app}(T', U'))$ , construits à partir des métavariabes  $T, T', U$  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  $f : R \rightarrow S$  est un morphisme de modèles, alors les diagrammes suivants sont commutatifs :

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

Ainsi formulée, cette règle de congruence entre dans notre définition de *règle de réduction* détaillée en Section 5.3. Une *action* de cette règle dans une monade de réduction  $R$  munie d'une opération binaire  $\text{app}$  est la donnée d'une réduction  $\text{app-cong}(m_T, m_U)$  entre  $\text{app}(T, T')$  et  $\text{app}(U, U')$  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  $\text{app-cong}$  commute avec la substitution, c'est-à-dire que l'équation suivante soit vérifiée :

$$\text{app-cong}(m_T, m_U)\{x \mapsto v_x\} = \text{app-cong}(m_T\{x \mapsto v_x\}, m_U\{x \mapsto v_x\})$$

Le théorème principal (Théorème 155) du chapitre 5 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. Précisons la notion de modèle mise en jeu ici : il s'agit d'une monade de réduction munie

- d'une structure de modèle de la 2-signature pour la monade sous-jacente,
- d'une action de chaque règle de réduction de la signature.

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 signatures pour des variantes du lambda calcul avec  $\beta$ -réduction, nous proposons, dans la section 5.6, 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'ajoutent des règles de réduction entre termes. Cette spécification se fait en trois étapes : 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.

Dans le chapitre 6, nous généralisons ces développements, et traitons de nouveaux exemples, comme le lambda calcul en appel par valeurs avec réduction à grands pas : la syntaxe est celle du lambda calcul classique, mais dans cette variante, un terme se réduit en une valeur, c'est-à-dire en une variable ou bien en une lambda abstraction. La notion de valeur n'est pas stable par substitution arbitraire : par exemple,  $x\{x := y\} = y$  n'est pas une valeur. En revanche, elle est stable par substitution de valeurs : si  $v$  et  $(w_x)_{x \in X}$  sont des valeurs, alors  $v\{x \mapsto w_x\}$  est une valeur. Nous pouvons ainsi définir la monade  $LC_v$  des valeurs du lambda calcul : elle est munie d'une inclusion dans la monade  $L$  du lambda calcul qui nous permet de considérer cette dernière comme un module sur  $LC_v$ .

Dans cette variante du lambda calcul, la  $\beta$ -réduction est formulée ainsi :

$$\frac{t \rightsquigarrow \lambda x.t' \quad u \rightsquigarrow u' \quad t'\{x := u'\} \rightsquigarrow v}{t u \rightsquigarrow v}$$

Une autre règle de réduction assure que toute valeur, en tant que terme, se réduit vers elle-même.

Les réductions sont stables par substitution : si  $t \in L(X)$  se réduit en une valeur  $v \in LC_v(X)$  et  $(u_x)_{x \in X}$  est une famille de valeurs indexée par l'ensemble des variables

libres  $X$ , alors  $t\{x \mapsto u_x\}$  se réduit en  $v\{x \mapsto u_x\}$ .

La collection des ensembles de réductions entre un terme et une valeur est définie par induction, en appliquant successivement les règles de réduction envisagées. Elle est munie d'une opération de substitution adéquate, comme expliqué précédemment.

Avant de passer au cas général, remarquons qu'un lambda terme peut être représenté par un arbre binaire dont les feuilles sont des valeurs et les nœuds correspondent aux applications. Cette représentation induit une bijection : un tel arbre binaire détermine un lambda terme de manière unique.

$$(x (\lambda y.t)) z \quad \Leftrightarrow \quad \begin{array}{c} \diagup \quad \diagdown \\ x \quad \lambda y.t \end{array} z$$

Ainsi, nous pouvons identifier le module  $L$  des lambda termes avec la composition  $B \cdot \text{LC}_v$ , où  $B$  est la monade des arbres binaires :  $B(X)$  est l'ensemble des arbres binaires dont les feuilles sont choisies dans  $X$ .

S'inspirant de cet exemple, nous définissons la notion de *monade opérationnelle*. Il s'agit :

- d'une monade  $R$  (dans notre exemple,  $R = \text{LC}_v$ ) ;
- d'une paire d'*endofoncteurs*  $(T_1, T_2)$  sur  $\text{Set}$  (dans notre exemple,  $T_1 = B$  et  $T_2 = \text{Id}$ ) ;
- d'un ensemble de réductions entre  $t$  et  $u$ , pour chaque paire  $(t, u) \in T_1(R(X)) \times T_2(R(X))$  ;
- d'une opération de substitution appropriée sur ces ensembles de réductions.

Nous définissons ensuite les *signatures opérationnelles* permettant de spécifier ce type d'objet mathématique, et démontrons un résultat d'initialité adéquat (Théorème 221).

### 1.1.8 Formalisation

Les preuves des résultats principaux des chapitres 3 et 4 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 théoriquement incontestables. Cette garantie 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 exempt d'erreurs d'implémentation 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.

En principe, la formalisation des démonstrations mathématiques limite la charge de travail du lecteur soupçonneux, qui n'a plus qu'à se poser la question, soulevée par le premier point, 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) adaptée à l'outil informatique. Il n'en est pas moins vrai que si les fondements théoriques de Coq s'avèrent incohérents, toute démonstration effectuée dans ce cadre devient potentiellement inexploitable. Ce risque est présent pour toute proposition de fondements des mathématiques.

Jusqu'à présent, aucun problème 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 corrigés).

Le troisième point est problématique : en effet, il est possible de se tromper dans la formalisation d'une définition, et 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. Un exemple extrême consisterait à définir par des moyens élaborés un certain ensemble qui se révélerait être 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 certain nombre de définitions et de résultats que nous avons pu exploiter ;
- ii elle utilise un nombre limité de fonctionnalités du langage ;
- iii elle intègre l'*axiome d'univalence* et un certain 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 cruciale. 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 estimons que nos preuves n'exploitent pas cette faille logique.

## 1.2 Long summary

This section is an English translation of the previous one.

This thesis deals with the mathematization of the notion of *language of programming*, paying particular attention to the notion of *substitution*.

Research in the field of programming languages traditionally relies on a definition of *syntax* modulo renaming of bound variables, with its associated *operational semantics*. We are interested in mathematical tools allowing to automatically generate syntax and semantics from basic data.

As regards the mathematization of syntax, the specification of algebraic structures with related variables is a major issue. Two main lines of research are in competition: *nominal sets* [GP99] and *substitution algebras* [FPT99]. In this thesis, we explore a variant of substitution algebras, proposed by [HM07; HM10], which is based on the notion of *module over a monad*.

In chapter 5, we define the *monads of reduction*: they constitute our mathematical counterpart of the notion of programming language. We then propose a protocol based on a three-level signature to specify them:

1. specification of constructions, for example a binary operation  $+$ ;
2. specification of equations, for example  $a + b = b + a$  (commutativity the binary operation);
3. specification of reductions between terms, for example  $0 + a \rightsquigarrow a$ , by schematic rules.

The first two points define what we call the *syntax* of the programming language, while the last point concerns the *semantics* of language: a reduction between two terms is a program execution path evolving from the first term to the second. As an example of language integrating the three levels, we propose in the section 5.6 a specification of lambda calculus with explicit substitution described in [Kes09]. We define in Chapter 6 the notion of *operational monad*, generalizing that of monad of reduction, to deal with new examples, as the lambda calculus in call by value with its big-step semantics.

In order to motivate the mathematical notions involved, we examine in this summary the simplest functional programming language that can be considered: the pure lambda calculus. In section 1.2.1, we give a first presentation of its syntax, and endow it with an operation of substitution. We then explain, in section 1.2.2, how the mathematical notion of monad can account for it, then, in the section 1.2.3, how the notion of morphism of modules provides a way to express an essential property of the constructions of the syntax: *compatibility with substitution*. In the section 1.2.4, we characterize the syntax by its principle of recurrence, which we formulate by a property of *initiality*.

We explain in section 1.2.5 that specifying this property of initiality requires an adequate notion of model, which is determined by the *signature*: this is the opportunity to present the general methodology of *initial semantics*. We examine then the case of syntaxes satisfying equations (section 1.2.6), before approaching, in section 1.2.7, the specification of semantics, modeled as part of the reduction monads by a set of reductions between each pair of terms.

### 1.2.1 Naive presentation of the lambda calculus syntax

We give here a presentation of the syntax of lambda calculus, as well as a overview of some of the difficulties usually associated with such a presentation. We fix an infinite set  $V$  of variables, and we characterize recursively all of the valid *terms* or expressions of the lambda calculus:

- each variable  $x \in V$  is a term of the lambda calculus,
- if  $t$  and  $u$  are terms, then  $t u$  is a term, called *application* of  $t$  to  $u$  ;
- if  $t$  is a term, then  $\lambda x.t$  is a term, called lambda abstraction of  $t$ , where  $x$  is a variable that can appear in  $t$ .

The expression  $\lambda x.t$  corresponds to the mathematical notation  $x \mapsto t$ . It is about defining a function depending on the variable  $x$ , the body of this function being given by the term  $t$ . The expression  $f t$  corresponds to the mathematical notation  $f(t)$ : it is the application of the function  $f$  to the argument  $t$ .

In mathematics, the name of the variable chosen to define a function is purely conventional: the functions  $x \mapsto f(x)$  and  $y \mapsto f(y)$  are identical. Let us transpose this identification into the language of lambda calculus: we want to equalize the term  $\lambda x.t$  with the term  $\lambda y.t'$ , where  $t'$  is obtained from the term  $t$  by replacing all occurrences of the variable  $x$  with the variable  $y$ . In this situation, we say that  $x$  is a *variable bound* in  $\lambda x.t$ , and the occurrences of  $x$  in  $t$  are then qualified as bound. Occurrences of variables that are not bound are said *free*<sup>6</sup>.

Here, the terms  $\lambda x.t$  and  $\lambda y.t'$  are said  *$\alpha$ -equivalent*. More generally, two terms are  *$\alpha$ -equivalents* if we can rename the bound variables of one to get the other term.

---

6. A variable can have a bound instance and a free occurrence in the same term: for example,  $x$  in  $(\lambda x.x) x$  is bound in  $\lambda x.x$ , but appears freely on the right.

The precise definition of the relationship  $\alpha$ -equivalence requires some precautions. For example, in the previous case, it is implied that the variable  $y$  does not appear in  $t$ ; otherwise, we would identify (against our will) the terms  $\lambda x.y$  and  $\lambda y.y$ .

*Substitution* is another essential aspect of the syntax of lambda calculus: given a term  $t$ , if we replace all (free) occurrences of a variable  $x$  by the same term  $u$ , we get a new valid expression, which we note  $t\{x := u\}$ . The substitution operation allows to express the following mathematical intuition: the result of a function  $x \mapsto t$  applied to a  $u$  argument is obtained by replacing the variable  $x$  in  $t$  by  $u$ . This statement is transposed, for the lambda calculus, in the  $\beta$ -equation

$$(\lambda x.t) u = t\{x := u\}. \quad (1.3)$$

This so-called *unary* substitution is a special case of the operation of *parallel substitution*  $t\{x \mapsto u_x\}$ , which replaces simultaneously each variable of a term  $t$  by a corresponding term.

Substitution only replaces the free occurrences of a variable, so as to preserve the following property: given two terms  $\lambda x.t$  and  $\lambda y.t'$  assumed  $\alpha$ -equivalent, replacing the same variable with the same term yield two  $\alpha$ -equivalent terms. If the involved variable is identical to the abstracted variable, the term  $(\lambda x.t)\{x := u\}$  is simply equal to  $\lambda x.t$ : for example  $(\lambda x.x)\{x := u\} = \lambda x.x$  is indeed  $\alpha$ -equivalent to  $(\lambda y.y)\{x := u\} = \lambda y.y$ .

### 1.2.2 The lambda calculus monad

The concept of monad provides a mathematical counterpart to the intuitive notion of syntax with a substitution operation. We motivate this definition by the example of the lambda calculus. In the point of view that we adopt here, the terms  $\alpha$ -equivalents are considered identical: thus,  $\lambda x.x = \lambda y.y$ .

Instead of considering a single set of terms with a set of variables  $V$  fixed in advance, we define classes of terms which use the same free variables. Let  $L(X)$  be the set of terms whose free variables are selected from the set  $X$ . Note that a term  $t \in L(X)$  is also found in  $L(Y)$  for any inclusion  $X \subset Y$ : indeed, if the free variables are chosen from the elements of a set  $X$ , they are also chosen from the elements of any superset  $Y$  of  $X$ .

Any variable is in particular a valid term: there is an inclusion  $\text{var}_X : X \rightarrow L(X)$  for any set  $X$ . On the other hand, if one gives for all variable  $x \in X$  a term  $u_x$  whose free

variables are chosen in  $Y$ , we get, from any term  $t \in L(X)$ , a term  $t\{x \mapsto u_x\} \in L(Y)$ . This parallel substitution operation satisfies the following properties:

- each variable is replaced by the appropriate term:

$$x'\{x \mapsto u_x\} = u_{x'}$$

- identity substitution is neutral:

$$t\{x \mapsto x\} = t$$

- any succession of substitutions is equivalent to a composed substitution:

$$t\{x \mapsto u_x\}\{y \mapsto v_y\} = t\{x \mapsto u_x\{y \mapsto v_y\}\}$$

The inclusion of variables in terms and the parallel substitution operation satisfying the equations above define a *monad on the category of sets*. This mathematical object is at the heart of developments that we expose in this thesis.

### 1.2.3 Constructions are morphisms of modules

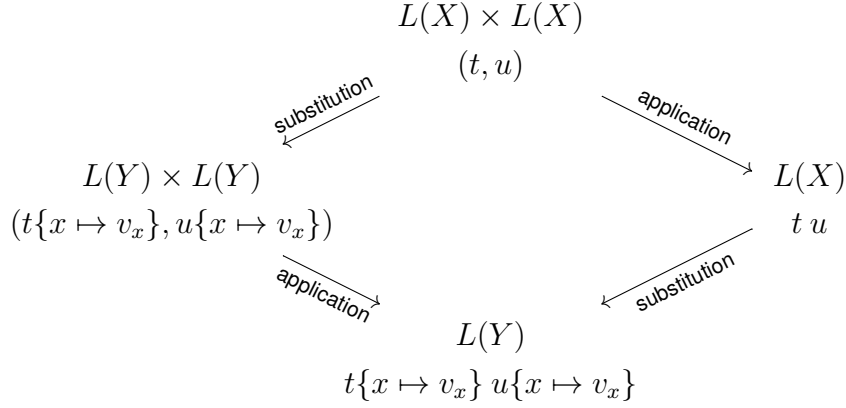
The mathematical concepts of modules and morphisms of modules provide a framework for expressing the compatibility of a construction in the syntax with the substitution operation. We illustrate this with the  $t\ u$  application of the lambda calculus.

The compatibility of the application with the substitution results in the commutation

$$(t\ u)\{x \mapsto v_x\} = t\{x \mapsto v_x\}\ u\{x \mapsto v_x\}$$

Informally, this equation means that there is no difference between performing the substitution before the application and performing the substitution after the application, as

shown in the following commutative diagram:



This observation implicitly relies on the following substitution operation of which the collection of sets  $(L(X) \times L(X))_X$ :

$$(t, u)\{x \mapsto v_x\} = (t\{x \mapsto v_x\}, u\{x \mapsto v_x\})$$

This substitution verifies the following properties:

- identity substitution is neutral:

$$(t, u)\{x \mapsto x\} = (t, u)$$

- any succession of substitutions is equivalent to a single compound substitution:

$$(t, u)\{x \mapsto u_x\}\{y \mapsto v_y\} = (t, u)\{x \mapsto u_x\{y \mapsto v_y\}\}$$

As such, the collection of sets of terms pairs defines a *module over the monad*  $L$ , which we note  $L \times L$ .

The definitions of monad and module are similar. Every monad defines a module on itself. In fact,  $L \times L$  also defines a monad, but the associated substitution does not derive from its module structure. Given a pair of terms  $(t, u)$  whose free variables are chosen in  $X$ , and for any variable  $x \in X$ , a pair of terms  $(v_x, w_x)$  whose free variables are chosen in  $Y$ , this monadic substitution provides a pair of terms whose free variables are in  $Y$ . The given substitution by the module structure is not appropriate, since it does not apply to the case where each variable is associated with a term in the monad  $L$ ,

rather than a pair of terms.

The application of lambda calculus induces a collection of functions

$$L(X) \times L(X) \rightarrow L(X)$$

that associate with any pair  $(t, u)$  the term  $t u$ . The commutativity property with the aforementioned substitution makes it a *morphism of modules* from  $L \times L$  to  $L$ , where  $L$  is seen as a module on the homonymous monad.

Similarly, the abstraction of lambda calculus induces a collection of functions

$$L(X \amalg \{\star\}) \rightarrow L(X)$$

which to any term  $t$  whose free variables are chosen in the set  $X$  extended with a new  $\star$  element, associates the term  $\lambda \star . t$ . The family  $L(X \amalg \{\star\})_X$  is canonically equipped with an operation of substitution and therefore defines a module. The collection of functions of abstraction then induces a morphism of modules, due to the commutation with substitution:

$$(\lambda \star . t)\{x \mapsto u_x\} = \lambda \star . \left( t \left\{ x \mapsto \begin{cases} \star & \text{if } x = \star \\ u_x & \text{otherwise.} \end{cases} \right\} \right)$$

The module construction that we encounter here generalizes to any module  $M$  on a monad  $R$ : the *derived module*  $M'$  is defined as the collection of sets  $(M(X \amalg \{\star\}))_X$  equipped with a canonical substitution operation.

In this thesis, we are interested exclusively in programming languages whose constructions and reductions are compatible with substitution, hence our interest in the notions of module and morphism of modules.

## 1.2.4 Recursion and initiality

The naive presentation of lambda calculus naturally induces a principle of recurrence on the syntax. Suppose, for example, that we want calculate the set of free variables of a term  $t$  of the lambda calculus. To do this, we reason by recurrence on the structure of the term. If  $t$  is a  $x$  variable, then the set of its free variables is the singleton  $\{x\}$ . If  $t$  is an application  $u v$ , then the set of its free variables is the union of the free variables

of  $u$  and  $v$ . If  $t$  is a lambda abstraction  $\lambda x.u$ , then any free variable of  $u$  different from  $x$  is a free variable of  $t$ .

In our framework, we adopt the point of view of Initial Semantics: the principle of recurrence is then a consequence of a property of *initiality*. The lambda calculus is thus characterized as the "minimal" monad provided with an application and a lambda abstraction, in a sense that we will illustrate with the example of computation of free variables (this example is studied more formally in Section 3.5.2).

Consider the monad  $\mathcal{P}$  that associates to  $X$  its powerset  $\mathcal{P}(X)$ : a variable  $x \in X$  induces a "term"  $\{x\} \in \mathcal{P}(X)$ ; the substitution  $t\{x \mapsto u_x\}$  is calculated by the meeting  $\bigcup_{z \in t} u_z$ . The union of two subsets provides a binary operation for  $\mathcal{P}$ , which we call an "application operation" by analogy with the binary application operation of the lambda calculus. This operation associates the subset  $t \cup u$  to the pair  $(t, u)$ . Similarly, an adequate abstraction operation  $\mathcal{P}(X \amalg \{\star\}) \rightarrow \mathcal{P}(X)$  is given by  $t \mapsto t \cap X$ , or equivalently, by  $t \mapsto t \setminus \{\star\}$ . As we will explain in the following section, these constructions make the monad  $\mathcal{P}$  a *model of the signature of lambda calculus*.

The initiality property of the mentioned lambda calculus then instantiates by the existence of a single family of functions  $(\text{free}_X : L(X) \rightarrow \mathcal{P}(X))_X$  verifying the following properties:

- free preserves the variables:

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

(remember that for the monad  $\mathcal{P}$ , the variable  $x$  is seen as the subset  $\{x\}$ )

- free preserves the substitution:

$$\text{free}_Y(t\{x \mapsto u_x\}) = \bigcup_{z \in \text{free}_X(t)} \text{free}_Y(u_z)$$

- free preserves the application:

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

- free preserves abstraction:

$$\text{free}(\lambda x.t) = \text{free}(t) \setminus \{x\}$$



The first two points characterize *free* as a *morphism of monads* between  $L$  and  $\mathcal{P}$ . The following section defines the notion of model so that  $L$  and  $\mathcal{P}$ , with their respective operations of application and abstraction, are models of the signature of the lambda calculus. The morphism of monads *free* is then a *morphism of models* of this signature, thanks to the last two points.

### 1.2.5 Signatures and modèles (Chapter 2)

The concept of characterising an object of a given type (i.e., of a certain category  $C$ ) 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]. In this thesis, we look at the specification of objects in the category of monads (Chapters 3 and 4), the category of *monads of reduction* (chapitre 5), and the category of *operational monads* (Chapter 6), our objective being to define a *formal language*<sup>7</sup>, modeled by an object of a relevant category.

The general methodology of the initial semantics can be described according to the following steps:

1. Introduce a notion of signature (for the category  $C$ ).
2. Construct a notion of associated model, organizing into a category with a functor to the category  $C$ .
3. Define the object specified by the signature as the initial model, if it exists (the signature is then called *effective*).
4. Find a sufficient condition for a signature to be effective<sup>8</sup>.

The models of a signature form the domain reached by the principle of recurrence, which is induced by the initiality of the object specified by the signature.

In Chapter 2, we define a general notion of signature to characterize an object of an arbitrary category  $C$ . Such a signature is given by a list of families of *arities* specifying

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

8. In the literature, the word signature is often reserved for the case where such a sufficient condition is automatically satisfied.

operations or equations. The remaining chapters provide various special cases, from the specification of the syntax to the specification of the semantics.

Let us focus on the syntactical aspect, mathematically represented by the concept of monad. We identify a first class of signatures specifying monads: the category of models of a signature is equipped with a functor to the category of monads. Consider the example of the syntax of lambda calculus. In the previous section, we characterized it as the initial monad equipped with an application and an abstraction. More specifically, generalizing the example of the powerset monad  $\mathcal{P}$  studied in the previous section we say that a monad  $R$  is equipped with application and abstraction if it has both a binary operation, that is to say a morphism of modules  $\text{app}^R : R \times R \rightarrow R$ , and a module morphism  $\text{abs}^R : R' \rightarrow R$ . Equivalently, it is a monad  $R$  with a morphism of modules from  $(R \times R) \amalg R'$  to  $R$ .

The signature  $\Sigma_{\text{LC}}$  of lambda calculus associates to any monad  $R$  the module  $(R \times R) \amalg R'$  over  $R$ . More generally, a signature  $\Sigma$  associates to any monad  $R$  a module  $\Sigma(R)$  over  $R$ . A *model*  $(R, \rho)$  (sometimes simply noted  $R$ ) of  $\Sigma$  is then a monad  $R$  with a module morphism  $\rho : \Sigma(R) \rightarrow R$ . Thus, the powerset monad  $\mathcal{P}$  and the lambda calculus monad  $\text{LC}$  induce models of  $\Sigma_{\text{LC}}$ . In fact, the lambda calculus is the initial model: if  $R$  is a model, then there is a unique morphism of monads  $f : L \rightarrow R$  which preserves binary operation and abstraction.

More generally, the syntax specified by a signature  $\Sigma$  is the initial model  $(S, \sigma)$  in the following sense: given a model  $(R, \rho)$ , there exists a unique morphism of monads  $f : S \rightarrow R$  which preserves the model structure, i.e., verifying

$$f(\sigma(t)) = \rho(f(t)) \quad (1.4)$$

Such a morphism of monads defines what we call a *morphism of models* between  $(S, \sigma)$  and  $(R, \rho)$ . Note that the right-hand side of Equation 1.4 requires making sense of the expression  $f(t)$  when  $t$  is an element of  $\Sigma(S)_X$ . In the case of the binary operation,  $\Sigma(S)_X = S(X) \times S(X)$ . So  $t$  is a pair  $(u, v)$  and we define  $f(u, v)$  by  $(f(u), f(v))$ . In the general case, we ask that any signature  $\Sigma$  comes with a "functorial" action: any morphism of monads  $f : R \rightarrow T$  induces a morphism of modules<sup>9</sup>  $\Sigma(R) \rightarrow \Sigma(T)$  on the monad  $R$  we note  $\Sigma(f)$ , or just  $f$ . The additional property of functoriality that we impose means that this action preserves the identity morphism and the composition of

---

9. In fact,  $\Sigma(T)$  is not a module on  $R$ , but on  $T$ . However, any morphism of monads  $R \rightarrow T$  induces on any module on  $T$  a module structure on  $R$ .

morphisms.

The existence of the syntax associated with any signature is not systematic<sup>10</sup>. This is nevertheless the case with any signature that we call *algebraic*, which specifies a syntax that has a set of  $n$ -ary operations, some of which bind variables in their arguments. The signature of lambda calculus is algebraic: the application is a classic binary operation, while abstraction is a unary operation binding a single variable in its unique argument.

### 1.2.6 Syntaxes with equations (Chapters 3 and 4)

In this thesis, we deal with the specification of programming languages satisfying syntactic equations, such as the lambda calculation modulo  $\beta$ -equation.

In Chapter 3, we study the signatures that we call *presentable*: these are, in a way, quotients of algebraic signatures. We show that they are effective (Theorem 54). The associated syntaxes can be seen as syntaxes derived from algebraic signatures, satisfying some equations. For example, it is possible to specify a commutative binary operation (Section 3.8.1). For that, it suffices to notice that the data of such an operation is equivalent to the data of an operation taking as argument an unordered pair of terms. Now let us describe the associated presentable signature  $\Sigma_{\text{comm-bin}}$ : this is a quotient of the algebraic signature  $\Sigma_{\text{bin}}$  of a binary operation that associates to any monad  $R$  the module  $R \times R$ . More specifically, to any monad  $R$ , the signature  $\Sigma_{\text{comm-bin}}$  associates the module  $(\mathcal{S}^2 R(X))_X$ , where  $\mathcal{S}^2 R(X)$  is the set of pairs of elements of  $R(X) \times R(X)$  quotiented by the relation  $(t, u) \sim (u, t)$ , that is, the set of unordered pairs. The syntax then comes with an operation that takes as argument a non ordered of pair of terms, as desired.

Nevertheless, the class of presentable signatures seems limited. Consider the example of an associative binary operation: we do not know how to give a signature, presentable or not. Note that we can nevertheless give an intuitive definition of model in this particular case: it is a monad  $R$  equipped with a binary operation  $b : R \times R \rightarrow R$  such that for all  $x, y, z$  in  $R(X)$  the expressions  $b(b(x, y), z)$  and  $b(x, b(y, z))$  are equal. In other words, it is a model  $(R, b)$  of the signature of a binary operation, such that the two morphisms of  $R$ -modules from  $R \times R \times R$  to  $R$ , associating to any triple  $(x, y, z)$  the two considered expressions are equal.

10. The counterexample 46 is the signature associating to any monad  $R$  the module  $(\mathcal{P}(R(X)))_X$ .

In Chapter 4, we give a definition of equations generalizing this example for models of any signature  $\Sigma$ : this is the data,

- for each  $R$  model of the signature  $\Sigma$ , of two morphisms of modules  $e_R, e'_R$  with the same domain  $A_R$  and codomain  $B_R$  (in the example above,  $e_R, e'_R : R \times R \times R \rightarrow R$  associate respectively  $b(b(t, u), v)$  and  $b(t, b(u, v))$  to the same triple  $(t, u, v) \in R^3(X)$ );
- for any morphism of  $f : R \rightarrow S$  models, of two morphisms of modules  $A_f : A_R \rightarrow A_S$  and  $B_f : B_R \rightarrow B_S$  such that following diagrams commute:

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

These data are submitted as usual to an additional condition of functoriality. It is said that a model  $R$  of the signature  $\Sigma$  satisfies the equation when  $e_R = e'_R$ .

We then consider an extension of the notion of signature: a *2-signature* is a pair composed of a signature (in the previous meaning) and a set of associated equations. A model (or 2-model) of a 2-signature is a  $R$  model of the underlying signature that satisfies all the equations. We introduce the algebraic 2-signatures: they are 2-signatures composed of an algebraic signature in the previous sense, and a set of so-called *elementary equations* (Definition 101). Technically, they are equations whose functorial action of the domain sends surjective morphisms of monads on surjective morphisms of functors, and whose codomain is of the form  $R \mapsto R' \cdots'$ . All the examples of presentable signature that we consider in Chapter 3 can be reformulated as an algebraic 2-signature, inducing the same category of models (up to isomorphism). Finally, we show the existence of an initial 2-model for any algebraic 2-signature (Theorem 104).

### 1.2.7 Semantics (Chapters 5 and 6)

It is possible to specify by a 2-signature the syntax of the lambda calculus quotiented by the  $\beta$ -equation: the terms  $(\lambda x.t)u$  and  $t\{x := u\}$  are thus equalized. However, this equation is usually oriented, and considered as an execution step when we consider lambda calculus as a functional programming language. As such, it is more appropriate

to integrate the  $\beta$ -reduction into the semantics of language, rather than impose the  $\beta$ -equation in the syntax. This motivates the concept of *monad of reduction*, which we introduce in the chapter 5, extending that of monad, to account for the additional structure of reduction. Intuitively, a monad of reduction is a monad  $R$  equipped, for each pair of terms  $(t, u) \in R(X)$ , with a set of reductions between  $t$  and  $u$  that we note  $t \blacktriangleright u$ , and an associated substitution operation: for any family of terms  $(v_x)_{x \in X}$  with  $v_x \in R(Y)$ , for any reduction  $m$  between  $t$  and  $u$ , this substitution associates a reduction  $m\{x \mapsto v_x\}$  between  $t\{x \mapsto v_x\}$  and  $u\{x \mapsto v_x\}$ . Equations analogous to those involved in the definition of module are required.

In this context, a *reduction signature* consists of a signature for monads, that is to say a 2-signature, and a family of *reduction rules*. For example, the reduction rule for the congruence of the application of the lambda calculus is informally expressed as follows, by explicitly designating the application of the lambda calculated by the app construction:

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

This rule breaks down into three pairs of terms: the hypotheses  $(T, T')$  and  $(U, U')$ , and the conclusion  $(\text{app}(T, U), \text{app}(T', U'))$ , constructed from the metavariables  $T$ ,  $T'$ ,  $U$ , and  $U'$ . These parameterized pairs induce pairs of morphisms  $(h_{1,1}(R), h_{1,2}(R))$ ,  $(h_{2,1}(R), h_{2,2}(R))$  and  $(c_1(R), c_2(R))$  of  $R$ -modules between  $R^4$  and  $R$  for any model  $R$  of the signature of lambda calculus (as a monad). In addition, this construction is functorial. More precisely, if  $f : R \rightarrow S$  is a morphism of models, then the following diagrams are commutative:

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

Thus formulated, this rule of congruence fits into our definition of *reduction rule* detailed in Section 5.3. An *action* of this rule in a reduction monad  $R$  equipped with a binary operation  $\text{app}$  is given by a reduction  $\text{app-cong}(m_T, m_U)$  between  $\text{app}(T, T')$  and  $\text{app}(U, U')$ , for all  $(T, T', U, U') \in R^4(X)$ , any reduction  $m_T$  between  $T$  and  $T'$ , and any reduction  $m_U$  between  $U$  and  $U'$ . It is moreover necessary that  $\text{app-cong}$  commutes with

substitution, that is, the following equation is verified:

$$\text{app-cong}(m_T, m_U)\{x \mapsto v_x\} = \text{app-cong}(m_T\{x \mapsto v_x\}, m_U\{x \mapsto v_x\})$$

The main theorem (Theorem 155) of Chapter 5 asserts the existence of an initial model of a reduction signature composed of an effective 2-signature and any family of reduction rules.

Let us specify the notion of model put into play here: it is a reduction monad equipped with

- a model structure of the 2-signature for the underlying monad,
- an action of each reduction rule of the signature.

The initial model is built from the initial model of the 2-signature, and the additional structure of reductions is built inductively from the reduction rules. Besides some signatures for variants of lambda calculus with  $\beta$ -reduction, we propose in the section 5.6 a signature for the lambda calculus with explicit substitution, as described in [Kes09], as a set of operations subject to a syntactic equation, with the addition of reduction rules between terms. This specification is done in three steps: a 1-signature for language operations, a 2-signature taking into account the syntactic equation, and a reduction signature specifying the relevant reductions.

In Chapter 6, we generalize these developments, and deal with new examples, such as lambda calculus in call by value with big-step reductions: the syntax is that of lambda calculus, but in this variant, a term reduces to a value, that is to say in a variable or a lambda abstraction. The notion of value is not stable by arbitrary substitutions: for example,  $x\{x := y\} = y$  is not a value. On the other hand, it is stable by substitution of values:  $v$  and  $(w_x)_{x \in X}$  are values, then  $v\{x \mapsto w_x\}$  is a value. We can thus define the monad  $\text{LC}_v$  of the values of lambda calculus: it is equipped with an inclusion in the monad  $L$  of lambda calculus that allows us to consider the latter as a module over  $\text{LC}_v$ .

In this variant of lambda calculus, the  $\beta$ -reduction is formulated as:

$$\frac{t \rightsquigarrow \lambda x. t' \quad u \rightsquigarrow u' \quad t'\{x := u'\} \rightsquigarrow v}{t u \rightsquigarrow v}$$

Another reduction rule ensures that any value, as a term, reduces to itself.

The reductions are stable by substitution: if  $t \in L(X)$  reduces to a value  $v \in \text{LC}_v(X)$

The collection of sets of reductions between a term and a value is defined by induction, by successively applying the reduction rules. It is equipped with an adequate substitution operation, as explained previously.

$$(x \ (\lambda y.t)) \ z \quad \Leftrightarrow \quad \begin{array}{c} \diagup \quad \diagdown \\ x \quad \lambda y.t \end{array}$$

1. the dependent type theory on which the software is based is complex, as well as its proof of correctness;
2. like any computer project, the software suffers from bugs that may lead to accepting erroneous demonstrations;
3. even if the proof of a theorem is validated, the mathematician must still check that the mathematical statement that he has formalized is actually what he wrote on the paper.

In principle, the formalization of mathematical proofs limits the workload of the suspicious reader, who just has to ask himself the question, raised by the first point, of the validity of the evidence of correction of type theory, and is no longer forced to check each demonstration, which is a mechanical task (if the proof is sufficient detailed) adapted to the computer tool. It remains true that if the theoretical foundations of Coq prove to be inconsistent, any proof performed in this setting becomes potentially unusable. This risk is present in any proposition of foundations of mathematics.

So far, no serious problem questioned the type theory on which Coq is based. On the other hand, as the second point suggests, contradictions are regularly exhibited by exploiting defects in the implementation of the software (which are then corrected).

The third point is problematic: indeed, it is possible to make a mistake in the formalization of a definition, and the software Coq does not help in this step. As a result, a statement evoking this definition does not have the meaning attributed to it a priori. An extreme example would be to define by elaborate means a certain set that turns out to be empty, because of a mistake in an intermediate step in the definition. Note that the empty set has this particular feature that its elements satisfy any property. The reader of a formalization can certainly dispense to verify the evidence, but he must pay close attention to the definitions.

For our developments, we chose to rely on the Coq library `UniMath`. This has some advantages, despite a "size" issue (which we will briefly mention):

- i it contains useful definitions and results for our purpose;
- ii it uses a limited amount of language features;
- iii it incorporates the *axiom of univalence* and many of its consequences.



Regarding point i, `UniMath` proposes, among other things, an implementation of the theory of categories, and quotients of sets.

Point ii limits the complexity of the type theory in which we must trust (as mentioned above in point 1), although this argument is highly questionable in view of the size issue. It also reduces the risk to be confronted with implementation bugs (point 2 above), by restricting itself to tried and tested features of Coq.

The axiom of univalence mentioned in point iii as such is not used crucially. It has the advantage, however, of gathering in this single axiom a number of useful consequences, for example functional extensionality (two functions are equal if they send each element on the same image), or propositional extensionality (logically equivalent propositions are equal) which makes it possible to handle the notion of subset.

Finally, the `UniMath` library has a major flaw due to a "size" issue. Indeed, it requires a compilation option that makes the system contradictory, allowing the implementation of a variant of Russel's paradox. This option is in practice used in the (impredicative) definition of quotients. We have ignored the existence of this particular inconsistency in the system in the proofs that we have formalized. We thus believe that our formalization does not exploit this logical flaw.

## 1.3 Synopsis

We give a synopsis of this thesis before presenting related work.

Chapter 2 presents a general notion of signature to specify objects of an arbitrary category. All the other chapters give particular instances of these general signatures.

Chapter 3 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 4, we extend the notion of signature, so that equations can also be specified, to yield the notion of 2-signature. The main result of this chapter is that "algebraic 2-signatures" have an initial model. This is based on the following article: [Ahr+19b] *Modular specification of monads through higher-order presentations*.

Finally, in Chapter 5, 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 3 and 4.

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*<sup>11</sup>, that we avoid

---

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

here. Any signature with strength gives rise to a signature in our sense, cf. Proposition 41. 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].

## 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 4, we consider an analogous notion of 2-signature based on monads on sets. In some sense, Chapter 4 tackles the technical issue of quotienting the initial (relative) monad constructed in [Ahr16] by the preorder.

In a classical paper, Barr [Bar70] explained the construction of the “free monad” generated by an endofunctor<sup>12</sup>. In another classical paper, Kelly and Power [KP93] explained how any finitary monad can be presented as a coequalizer of free monads<sup>13</sup>. There, free monads correspond to our initial models of an algebraic 1-signature without

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

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

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 Section 5.7 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<sup>14</sup>. We highlight some contributions:

---

14. 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.

- 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  $\beta$ - and  $\eta$ -equations.

### 1.5.3 Syntax with reductions

Reduction signatures of Chapter 5 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.

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- $\beta$ -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]).



# GENERAL SIGNATURES

---

In this chapter, we give an abstract notion of signature for specifying objects of a category  $\mathcal{C}$ . All the notions of signatures presented in this manuscript are instances of these. In Chapters 3 and 4, we study the case when  $\mathcal{C} = \mathbf{Mon}$  the category of monads on sets; in Chapter 5, we focus on the case when  $\mathcal{C}$  is the category of reduction monads, and in Chapter 6, we consider the more general category of operational monads.

To each signature  $S$ , there is an associated category of models  $\mathcal{C}^S$  equipped with a forgetful functor  $U^S : \mathcal{C}^S \rightarrow \mathcal{C}$ . For example, the category of models of the *empty signature* is just  $\mathcal{C}$ . The object of  $\mathcal{C}$  specified by a signature is the image by  $U^S$  of the initial model, if any.

For each signature  $S$ , we define a notion of  $S$ -arity, specifying operations or equations. A family  $E$  of such arities induces an extended signature  $S, E$ , equipped with a functor  $U^E$  from  $\mathcal{C}^{S,E}$  to  $\mathcal{C}^S$ , factorizing  $U^{S,E} : \mathcal{C}^{S,E} \rightarrow \mathcal{C}$ .

Any signature is constructed inductively in this manner from the empty signature. Intuitively, a signature is thus a dependent context, where dependent types correspond to families of arities.

## 2.1 Arities over a category

We first define arities over a category  $\mathcal{C}$ , fixed in this section. Then,  $S$ -arities will be defined as arities over the category of models of the signature  $S$ .

We define what is an *action* of an arity in an object  $c \in \mathcal{C}$ , and the associated notion of compatible morphism of  $\mathcal{C}$  between objects equipped with such an action.

We begin by introducing the general notion of arity over a category in Section 2.1.1, before considering the particular case of equational arities in Section 2.1.2, allowing to specify equations.

### 2.1.1 General arities

**Definition 1.** An **arity over**  $C$  is a quadruple  $(D, a, u, v)$  consisting of:

- a category  $D$ ;
- a functor  $a : D \rightarrow C$ ;
- two sections  $u, v : C \rightarrow D$ .

**Remark 2.** We will mainly consider the case where  $a$  is a Grothendieck fibration. In this case, thanks to the Grothendieck construction, an arity over  $C$  can be defined as a pseudo functor  $a : C^o \rightarrow \text{Cat}$ , and two natural transformations  $u, v : 1 \rightarrow a$ , where  $1$  is the terminal functor from  $C$  to  $\text{Cat}$ .

**Example 3.** Any endofunctor  $F : C \rightarrow C$  induces an arity  $(D, a, u, v)$  specifying algebra structures for this endofunctor, as we will explain in Example 6:

- $a : C \times C \rightarrow C$  is the first projection;
- $u = \text{Id}_C, F$  maps an object  $c$  to the pair  $(c, F(c))$ ;
- $v$  is the diagonal functor, mapping an object  $c$  to  $(c, c)$ .

Next, we define what the notion of action of an arity in an object of  $C$ , and the associated notion of compatible morphism of  $C$ , which always applies to identity morphisms.

**Definition 4.** An **action of an arity**  $A = (D, a, u, v)$  **in an object**  $c \in \text{ob } C$  is a morphism  $h : u(c) \rightarrow v(c)$  such that  $a(h) = \text{id}_c$ .

**Definition 5.** Let  $A = (D, a, u, v)$  be an arity over  $C$ . Let  $c_1$  and  $c_2$  be two objects of  $C$  equipped with actions  $h_1 : u(c_1) \rightarrow v(c_1)$  and  $h_2 : u(c_2) \rightarrow v(c_2)$ . A morphism  $f : c_1 \rightarrow c_2$  is **compatible with the actions**  $h_1$  and  $h_2$  if the following diagram commutes:

$$\begin{array}{ccc} u(c_1) & \xrightarrow{h_1} & v(c_1) \\ u(f) \downarrow & & \downarrow v(f) \\ u(c_2) & \xrightarrow{h_2} & v(c_2) \end{array}$$

**Example 6.** Let  $F$  be an endofunctor on  $C$ . In Example 3, we constructed from it an arity . An action of it in an object  $c \in \text{ob } C$  is a morphism  $h : F(c) \rightarrow c$ . A compatible morphism between  $c$  and  $c'$  equipped with actions  $h$  and  $h'$  is a morphism between the induced algebras of  $F$ .



## 2.1.2 Equational arities

**Definition 7.** An arity  $(D, a, u, v)$  over  $C$  is said **equational** if  $a : D \rightarrow C$  is faithful.

**Remark 8.** An action of an equational arity  $(D, a, u, v)$  in an object  $c \in C$  is always an identity morphism, as  $a$  is faithful. Thus, an object  $c$  is equipped with an action if and only if  $u_c = v_c$ . Any morphism is then compatible with actions of an equational arity.

**Example 9.** An *equational system* [FH09] induces an equational arity over a category of algebras. More precisely, an equational system  $\mathbb{S} = (C : \Sigma \triangleright \Gamma \vdash L = R)$  consists of endofunctors  $\Sigma, \Gamma : C \rightarrow C$ , and functors  $L, R : \Sigma\text{-alg} \rightarrow \Gamma\text{-alg}$  between categories of algebras preserving the underlying object, that is, such that the following diagram commutes:

$$\begin{array}{ccc} \Sigma\text{-alg} & \begin{array}{c} \xrightarrow{L} \\ \xrightarrow{R} \end{array} & \Gamma\text{-alg} \\ & \searrow & \swarrow \\ & C & \end{array}$$

Such a structure defines an equational arity  $(D, a, u, v)$  over the category  $\Sigma\text{-alg}$  as follows.

- $D$  is the category of objects  $c \in C$  equipped with an algebra structure for both  $\Sigma$  and  $\Gamma$ . More formally,  $D$  is defined as the pullback

$$\begin{array}{ccc} D & \xrightarrow{\quad} & \Gamma\text{-alg} \\ \downarrow a & \lrcorner & \downarrow \\ \Sigma\text{-alg} & \xrightarrow{\quad} & C \end{array}$$

- $a : D \rightarrow \Sigma\text{-alg}$  is induced by the definition of  $D$  as a pullback.
- $u$  maps a  $\Sigma$ -algebra  $c$  to the underlying object of  $c$  equipped with the same  $\Sigma$ -algebra structure and the  $\Gamma$ -algebra given by  $L(c)$ .
- $v$  maps a  $\Sigma$ -algebra  $c$  to the underlying object of  $c$  equipped with the same  $\Sigma$ -algebra structure and the  $\Gamma$ -algebra given by  $R(c)$ .

Then, the category of  $\mathbb{S}$ -algebras [FH09, Definition 3.4] is retrieved as the category of objects of  $\Sigma\text{-alg}$  equipped with an action of this arity and compatible morphisms. More concretely, it is the full subcategory of algebras of  $\Sigma$  that have the same image by  $L$  and  $R$ .

### 2.1.3 Family of arities and their models

**Definition 10.** Let  $E$  be a family of arities over  $C$ . The **category  $C^E$  of models of  $E$**  is defined as follows:

- objects are objects  $c$  of  $C$  equipped with an action of each arity in  $E$ ;
- morphisms are those which are compatible with the action of any arity in  $E$ .
- composition and identities are the obvious ones.

It is equipped with a forgetful functor  $U^E : C^E \rightarrow C$ .

## 2.2 Signatures for a category

Here we define signatures and their models.

**Definition 11.** A **signature for a category  $C$**  is a finite list  $E_1, \dots, E_n$  consisting of families of arities over categories such that:

- $E_1$  is a family of arities over  $C$ .
- each  $E_i$  is a family of arities over the category of models of  $E_{i-1}$ , for  $i > 1$ .

The empty list is called the **empty signature**. The **category of models  $C^S$  of a signature  $S = E_1, \dots, E_n$**  is defined as the category of models of  $E_n$ , if  $n > 0$ , or as  $C$  otherwise. Given a signature  $S$ , a  **$S$ -arity** is an arity over the category of models  $C^S$ . Each signature  $S = E_1, \dots, E_n$  is equipped with a forgetful functor  $U^S$  from its category of models to  $C$  defined as the composition:

$$C^{E_n} \xrightarrow{U^{E_n}} C^{E_{n-1}} \xrightarrow{U^{E_{n-1}}} \dots \longrightarrow C^{E_1} \xrightarrow{U^{E_1}} C$$

**Example 12.** An endofunctor  $\Sigma : C \rightarrow C$  induces a signature consisting of a single arity constructed in Example 3: it is easily verified that the category of models of this signature is the category of algebras of  $\Sigma$ .

**Example 13.** An equational system  $C : \Sigma \triangleright \Gamma \vdash L = R$  yields a signature extending the one induced by the endofunctor  $\Sigma$  (as explained in Example 12) with a singleton family consisting of the equational arity induced by the equational system (Example 9).

The associated category of models coincides with the category of algebras of this equational system, as defined in [FH09, Definition 3.4]. More concretely, as explained in Example 9, models are objects of  $\mathcal{C}$  equipped with a  $\Sigma$ -algebra structure that is mapped to the same image by  $L$  and  $R$ , and morphisms are algebra morphisms.

**Definition 14.** A signature  $S$  is said **effective** if its category of models has an initial model  $\hat{S}$ .

In the following chapters, we consider different instances of signatures:

- In Chapter 3, signatures consist of a single arity over the category of monads.
- In Chapter 4, these signatures are extended with a family of equational arities.
- In Chapters 5 and 6, we introduce the category of reduction monads, and then of their generalization, the operational monads. They are monads with additional structure. Thus, signatures over the category of monads are relevant for specifying the monadic part: we explain how to make it formal in the next section. In Chapter 5, we construct arities specifying the additional structure of reduction monads. Signatures for operational monads of Chapter 6 extend them with arities for specifying the (specifying the *state functor*).

## 2.3 Pulling back arities and signatures

In this section, we explain how a functor  $F : \mathcal{B} \rightarrow \mathcal{C}$  allows to turn any signature  $S$  over  $\mathcal{C}$  into a *pullback signature*  $F^*S$  over  $\mathcal{B}$ . This will be used in Chapters 5 and 6 to upgrade signatures for monads into signatures for reduction or operational monads. We explain how it works for arities.

### 2.3.1 Pullback of arities

In this section, we fix a functor  $F : \mathcal{B} \rightarrow \mathcal{C}$ .

**Definition 15.** Let  $A = (D, a, u, v)$  be an arity over  $\mathcal{C}$ . The **pullback of  $A$  along  $F$**  is the arity  $F^*A = (D', a', u', v')$  over  $\mathcal{B}$  as follows:

- $a' : D' \rightarrow B$  is the pullback of  $a$  along  $F$ , considering the pullback diagram

$$\begin{array}{ccc} D' & \xrightarrow{d} & D \\ \downarrow a' & \lrcorner & \downarrow a \\ B & \xrightarrow{F} & C \end{array}$$

- $u'$  and  $v'$  are the universal morphisms from  $E$  to  $D'$  factorizing the cones

$$\begin{array}{ccc} B & \xrightarrow{u \cdot F} & D \\ \downarrow \text{id}_B & & \downarrow a \\ B & \xrightarrow{F} & C \end{array} \quad \begin{array}{ccc} B & \xrightarrow{v \cdot F} & D \\ \downarrow \text{id}_B & & \downarrow a \\ B & \xrightarrow{F} & C \end{array}$$

**Definition 16.** Let  $E$  be a family of arities over  $C$ . The **pullback of  $E$  along  $F$**  is the family of arities  $F^*E$  over  $B$  consists of the pullbacks of arities of  $E$  along  $F$ .

The process of taking the category of models commutes with pullback:

**Proposition 17.** Let  $E$  be a family of arities over  $C$ . Then there is a functor  $p_{F,E} : B^{F^*E} \rightarrow C^E$  inducing a pullback diagram

$$\begin{array}{ccc} B^{F^*E} & \xrightarrow{p_{F,E}} & C^E \\ \downarrow U^{F^*E} & \lrcorner & \downarrow U^E \\ B & \xrightarrow{F} & C \end{array}$$

### 2.3.2 Pullback of signatures

In this section, we fix a functor  $F : B \rightarrow C$ .

**Definition 18.** The **pullback  $F^*S$  of a signature  $S = E_1, \dots, E_n$  over  $C$  along  $F$**  is the signature  $p_1^*E_1, p_2^*E_2, \dots, p_n^*E_n$  over  $B$ , where the functors  $p_i$  are defined inductively as follows:

- $p_1 : B \rightarrow C$  is  $F$ ;
- $p_i : B^{p_{i-1}^*E_{i-1}} \rightarrow C^{E_{i-1}}$  is  $p_{p_{i-1}, E_{i-1}}$  for  $i > 1$ .

We denote by  $p_{F,S} : B^{F^*S} \rightarrow C^S$  the functor  $F$  if  $n = 0$ , or  $p_{p_n, E_n}$  otherwise.

Again, the process of taking the category of models commutes with pullback:

**Proposition 19.** *Let  $S$  be a signature over  $C$ . Then we have the following pullback diagram:*

$$\begin{array}{ccc} B^{F^*S} & \xrightarrow{p_{F,S}} & C^S \\ U^{F^*S} \downarrow & \lrcorner & \downarrow U^S \\ B & \xrightarrow{F} & C \end{array}$$

PART I

# Syntax

---

# PRESENTABLE SIGNATURES FOR MONADS

---

In the present chapter extracted from [Ahr+19a], we consider a particular class of signatures over the category of monads (in the sense of Chapter 2) that organize into a category. They allow for the specification of the syntax of some untyped programming languages with variable binding.

On the one hand, these 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, that is, to be effective. 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 3.6). Examples are given in Section 3.8.

Our notions of signature and syntax enjoy modularity in the sense introduced by [GUH06]: indeed, we define a “total” category of models where objects are pairs consisting of a signature together with one of its models; and in this total category of models, merging two extensions of a syntax corresponds to building 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 3.1 gives a succinct account of the notion of module over a monad, which is the crucial tool underlying our definition of signatures. Our categories of signatures and models are described in Sections 3.2 and 3.3 respectively. In Section 3.4, we give our definition of a syntax, and we present our first main result, a modularity result about merging extensions of syntax. In Section 3.5, we show through examples how recursion can be recovered from initiality. Our notion of *presentation of a signature* appears in Section 3.6. There, we also state our second main result: presentable signatures generate a syntax. The proof of that result is given in Section 3.7. Finally, in Section 3.8, we give examples of presentable signatures.

## 3.1 Categories of modules over monads

The main mathematical notion underlying our signatures is that of module over a monad. In this section, we recall the definition and some basic facts about modules over a monad in the specific case of the category  $\mathbf{Set}$  of sets, although most definitions are generalizable. See [HM10] for a more extensive introduction on this topic.

### 3.1.1 Modules over monads

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

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

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

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

Let  $R$  be a monad.

**Definition 20** (Modules). A left  $R$ -module is given by a functor  $M: \mathbf{Set} \rightarrow \mathbf{Set}$  equipped with a natural transformation  $\rho^M: M \cdot R \rightarrow M$ , called *module substitution*, which is



compatible with the monad composition and identity:

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

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 21.** • Every monad  $R$  is a module over itself, which we call the *tautological* module.

- For any functor  $F: \text{Set} \rightarrow \text{Set}$  and any  $R$ -module  $M: \text{Set} \rightarrow \text{Set}$ , the composition  $F \cdot M$  is an  $R$ -module (in the evident way).
- For every set  $W$  we denote by  $\underline{W}: \text{Set} \rightarrow \text{Set}$  the constant functor  $\underline{W} := X \mapsto W$ . Then  $\underline{W}$  is trivially an  $R$ -module since  $\underline{W} = \underline{W} \cdot R$ .
- Let  $M_1, M_2$  be two  $R$ -modules. Then the product functor  $M_1 \times M_2$  is an  $R$ -module (see Proposition 23 for a general statement).

**Definition 22** (Linearity). We say that a natural transformation of  $R$ -modules  $\tau: M \rightarrow N$  is *linear*<sup>1</sup> 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  $\text{Mod}(R)$ .

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

**Proposition 23** (LModule\_Colims\_of\_shape, LModule\_Lims\_of\_shape).  $\text{Mod}(R)$  is complete and cocomplete.

---

1. Given a monoidal category  $\mathcal{C}$ , there is a notion of (left or right) module over a monoid object in  $\mathcal{C}$  (see, e.g., [Bra14, Section 4.1] for details). The term “module” comes from the case of rings: indeed, a ring is just a monoid in the monoidal category of Abelian groups. Similarly, our monads are just the monoids in the monoidal category of endofunctors on  $\text{Set}$ , and our modules are just modules over these monoids. Accordingly, the term “linear(ity)” for morphisms among modules comes from the paradigmatic case of rings.

### 3.1.2 The total category of modules

We already introduced the category  $\text{Mod}(R)$  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 24** (Pullback). Let  $f: R \rightarrow S$  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  $f^*M$ .<sup>2</sup>

**Definition 25** (The total module category). We define the *total module category*  $\int^R \text{Mod}(R)$ , or  $\int \text{Mod}$  for short, as follows<sup>3</sup>:

- 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  $f: R \rightarrow S$  is a morphism of monads, and  $m: M \rightarrow f^*N$  is a morphism of  $R$ -modules.

Composition and identity morphisms are the expected ones. The category  $\int \text{Mod}$  comes equipped with a forgetful functor to the category of monads, given by the projection  $(R, M) \mapsto R$ .

**Proposition 26** (`cleaving_bmod`). *The forgetful functor  $\int \text{Mod} \rightarrow \text{Mon}$  is a Grothendieck fibration with fibre  $\text{Mod}(R)$  over a monad  $R$ . In particular, any monad morphism  $f: R \rightarrow S$  gives rise to a functor*

$$f^*: \text{Mod}(S) \rightarrow \text{Mod}(R)$$

*given on objects by Definition 24.*

**Proposition 27** (`pb_LModule_colim_iso`, `pb_LModule_lim_iso`). *For any monad morphism  $f: R \rightarrow S$ , the functor  $f^*: \text{Mod}(S) \rightarrow \text{Mod}(R)$  preserves limits and colimits.*

### 3.1.3 Derivation

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

2. The term “pullback” is standard in the terminology of Grothendieck fibrations (see Proposition 26).

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.

**Definition 28** (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  $\rho^{M'}: M' \cdot R \longrightarrow M'$  defined as in the diagram

$$\begin{array}{ccc} M(R(X) + *) & \xrightarrow{\rho_X^{M'}} & M(X + *) \\ \downarrow [M(R(i_X), \eta_{X+*} \circ *)] & \nearrow \rho_{X+*}^M & \\ M(R(X + *)) & & \end{array} \quad (3.1)$$

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

Derivation is a continuous and cocontinuous endofunctor on the category  $\text{Mod}(R)$  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 29.** Given a list of nonnegative integers  $(a) = (a_1, \dots, a_n)$  and a left module  $M$  over a monad  $R$ , we denote by  $M^{(a)} = M^{(a_1, \dots, a_n)}$  the module  $M^{(a_1)} \times \dots \times M^{(a_n)}$ . Observe that, when  $(a) = ()$  is the empty list,  $M^{()} is the final module  $*$ .$

**Definition 30.** For every monad  $R$  and  $R$ -module  $M$  we have a natural *substitution morphism*  $\sigma: M' \times R \longrightarrow M$  defined by  $\sigma_X = \rho_X^M \circ w_X$ , where  $w_X: M(X + *) \times R(X) \rightarrow M(R(X))$  is the map

$$w_X: (a, b) \mapsto M([\eta_X, \underline{b}])(a), \quad \underline{b}: * \mapsto b.$$

**Lemma 31** (substitution\_laws). *The transformation  $\sigma$  is linear.*

The substitution  $\sigma$  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 32** (deriv\_adj). *The endofunctor of  $\text{Mod}(R)$  mapping  $M$  to the  $R$ -module  $M \times R$  is left adjoint to the derivation endofunctor, the counit being the substitution morphism  $\sigma$ .*

## 3.2 The category of signatures

In this section, we give our notion of signature for specifying monads: they are particular signatures over the category of monads consisting of a single arity in the sense of Chapter 2.

The purpose of a signature is to act on monads. An action of a signature  $\Sigma$  in a monad  $R$  is a morphism from the module  $\Sigma(R)$  to the tautological one  $R$ . For instance, in the case of the signature  $\Sigma$  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 33.** A *signature for monads* (or just signature in the following) is signature over the category of monads consisting of a single arity of the shape  $(\int \text{Mod}, U, \Sigma, \Theta)$ , where  $U : \int \text{Mod} \rightarrow \text{Mon}$ . More concretely, it is given by a section  $\Sigma$  of the forgetful functor from the category  $\int \text{Mod}$  to the category  $\text{Mon}$ , that is, a functor  $\Sigma : \int \text{Mod} \rightarrow \text{Mon}$  making the following diagram commute:

$$\begin{array}{ccc} \text{Mon} & \xrightarrow{\Sigma} & \int \text{Mod} \\ & \searrow & \swarrow \\ & \text{Mon} & \end{array}$$

Now we give some basic examples of signatures.

- Example 34.**
1. The assignment  $R \mapsto R$  yields a signature, which we denote by  $\Theta$ .
  2. For any functor  $F : \text{Set} \rightarrow \text{Set}$  and any signature  $\Sigma$ , 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  $\Sigma$  and  $\Upsilon$ , 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,  $\Theta^n$  is the signature of  $n$ -ary operations.
  5. Given two signatures  $\Sigma$  and  $\Upsilon$ , 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.

**Remark 35.** Any signature  $\Sigma$  induces an arity  $A_\Sigma = (\int \text{Mod}, U, \Sigma, \Theta)$ , where  $U : \int \text{Mod} \rightarrow \text{Mon}$ , and thus a signature  $S_\Sigma$  in the sense of Chapter 2, consisting of this single arity.

The last example above explains how we can combine multiple signatures into a single one. Hence we do not need to distinguish between such arities—used to specify a single syntactic construction—and families of such arities—used to specify a family of syntactic constructions. Our notion of signatures that we present here allow us to do both (via Proposition 39 for families that are not necessarily finitely indexed).

*Elementary* signatures are of a particularly simple shape:

**Definition 36.** For each sequence of nonnegative integers  $s = (s_1, \dots, s_n)$ , the assignment  $R \mapsto R^{(s_1)} \times \dots \times R^{(s_n)}$  (see Definition 29) is a signature, which we denote by  $\Theta^{(s)}$ , or by  $\Theta'$  in the specific case of  $s = (1)$ . Signatures of this form are said *elementary*.

**Remark 37.** The product of two elementary signatures is elementary.

**Definition 38.** A *morphism between two signatures*  $\Sigma_1, \Sigma_2 : \text{Mon} \rightarrow \int \text{Mod}$  is a natural transformation  $m : \Sigma_1 \rightarrow \Sigma_2$  which, post-composed with the projection  $\int \text{Mod} \rightarrow \text{Mon}$ , becomes the identity. Signatures form a subcategory  $\text{Sig}$  of the category of functors from  $\text{Mon}$  to  $\int \text{Mod}$ .

Limits and colimits of signatures can be easily constructed pointwise:

**Proposition 39** (`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  $\Sigma$  and family of signatures  $(S_o)_{o \in O}$ , the canonical morphism  $\coprod_{o \in O} (S_o \times \Sigma) \rightarrow (\coprod_{o \in O} S_o) \times \Sigma$  is an isomorphism.*

**Definition 40.** 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  $\Sigma_{\text{LC}} = \Theta^2 + \Theta'$ .

To conclude this section, we explain the connection between *signatures with strength* (on the category  $\text{Set}$ ) 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  $H : [\mathcal{C}, \mathcal{C}] \rightarrow [\mathcal{C}, \mathcal{C}]$  together with a strength-like datum. Here, we only consider signatures with strength over the base category  $\mathcal{C} := \text{Set}$ . Given a signature with strength  $H$ , we also refer to the underlying endofunctor on the functor category  $[\text{Set}, \text{Set}]$  as  $H : [\text{Set}, \text{Set}] \rightarrow [\text{Set}, \text{Set}]$ .

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  $\text{SigStrength}$  [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  $\alpha : H_1 \rightarrow H_2$  of signatures with strength, we associate to it a morphism of signatures  $\tilde{\alpha} : \tilde{H}_1 \rightarrow \tilde{H}_2$ . This morphism sends a monad  $R$  to a module morphism  $\tilde{\alpha}(R) : \tilde{H}_1(R) \rightarrow \tilde{H}_2(R)$  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 41** (`sigWithStrength_to_sig_functor`). *The maps sketched above yield a functor  $(\tilde{-}) : \text{SigStrength} \rightarrow \text{Sig}$ .*

### 3.3 Categories of models

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

**Definition 42** (Models and actions). Given a signature  $\Sigma$ , the *category  $\text{Mon}^\Sigma$  of models of  $\Sigma$*  is the category of models of  $S_\Sigma$  (see Remark 35). More precisely, its objects are pairs  $(R, r)$  of a monad  $R$  equipped with a module morphism  $r : \Sigma(R) \rightarrow R$ , called *action of  $\Sigma$  in  $R$* . In other words, a model of  $\Sigma$  is a monad  $R$  equipped with an action of  $\Sigma$  in  $R^4$ . A *morphism from  $(R, r)$  to  $(S, s)$*  is a morphism of monads  $m : R \rightarrow S$  compatible

---

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  $\nu : T(X) \rightarrow X$  that is compatible with the multiplication and unit of the monad. This morphism is sometimes called an action.

with the actions, in the sense that the following diagram of  $R$ -modules commutes:

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

Here, the horizontal arrows come from the actions, the left vertical arrow comes from the functoriality of signatures, and  $m: R \rightarrow m^*S$  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 43.** The usual  $\text{app}: \text{LC}^2 \rightarrow \text{LC}$  is an action of the elementary signature  $\Theta^2$  in the monad  $\text{LC}$  of syntactic lambda calculus. The usual  $\text{abs}: \text{LC}' \rightarrow \text{LC}$  is an action of the elementary signature  $\Theta'$  in the monad  $\text{LC}$ . Then  $[\text{app}, \text{abs}]: \text{LC}^2 + \text{LC}' \rightarrow \text{LC}$  is an action of the algebraic signature of the lambda calculus  $\Theta^2 + \Theta'$  in the monad  $\text{LC}$ .

In the formalisation, the category of models of a signature  $\Sigma$  is recovered as the fiber category over  $\Sigma$  of the displayed category `[AL17]` of models, see `rep_disp`. We have also formalized a direct definition (`rep_fiber_category`) and shown that the two definitions yield isomorphic categories: `catiso_modelcat`.

**Definition 44** (Pullback). Let  $f: \Upsilon \rightarrow \Sigma$  be a morphism of signatures and  $(R, r)$  a model of  $\Sigma$ . The linear morphism  $\Upsilon(R) \xrightarrow{f(R)} \Sigma(R) \xrightarrow{r} R$  defines an action of  $\Upsilon$  in  $R$ . The induced model of  $\Upsilon$  is called *pullback*<sup>5</sup> of  $(R, r)$  along  $f$  and denoted by  $f^*(R, r)$ .

## 3.4 Syntax

We are primarily interested in the existence of an initial object in the category  $\text{Mon}^\Sigma$  of models of a signature  $\Sigma$ . We call such an essentially unique object *the syntax generated by  $\Sigma$* .

5. Following the terminology introduced in Definition 24, the term “pullback” is justified by Lemma 52.

### 3.4.1 Representations of a signature

**Definition 45.** If  $\text{Mon}^\Sigma$  has an initial object, this object is essentially unique; we say that it is a *representation*<sup>6</sup> of  $\Sigma$  and call it the *syntax generated by  $\Sigma$* , 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  $\Sigma$  exists, we say that  $\Sigma$  is *effective*: this is equivalent to the assertion that  $S_\Sigma$  of Remark 35 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 46.** Let  $\mathcal{P}$  denote the powerset functor and consider the signature  $\mathcal{P} \cdot \Theta$  (see Example 34, 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 49.

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

**Lemma 47.** *Let  $(R, r)$  be a model of a signature  $\Sigma$ . Let  $\eta : \text{Id} \rightarrow R$  be the unit of the monad  $R$ , and let  $\rho^{\Sigma(R)} : \Sigma(R) \cdot R \rightarrow \Sigma(R)$  be the module substitution of the  $R$ -module  $\Sigma(R)$ .*

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

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

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

---

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



- Moreover, this monad can be given the following  $\Sigma$ -action:

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

- The natural transformation  $[r, \eta] : \Sigma(R) + \text{Id} \rightarrow R$  is a model morphism, that is, it commutes suitably with the  $\Sigma$ -actions of Diagram (3.2) in the source and  $r : \Sigma(R) \rightarrow R$  in the target.

**Definition 48.** Given a model  $M$  of  $\Sigma$ , we denote by  $M^\#$  the  $\Sigma$ -model constructed in Lemma 47, and by  $\epsilon_M : M^\# \rightarrow M$  the morphism of models defined there.

**Lemma 49** (`iso_mod_id_model`). *If  $\Sigma$  is effective, then the morphism of  $\Sigma$ -models*

$$\epsilon_{\hat{\Sigma}} : \hat{\Sigma}^\# \rightarrow \hat{\Sigma}$$

*is an isomorphism.*

We go back to considering the signature  $\Sigma := \mathcal{P} \cdot \Theta$ . Suppose that  $\Sigma$  is effective. From Lemma 49 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 50** (`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  $\text{Set}$ . Its initial algebra (constructed as the colimit of the initial chain) is given the structure of a monad with an action of the algebraic signature, and then a routine verification shows that it is actually initial in the category of models. The computer-checked proof uses the construction of a monad from an algebraic signature formalized in [AMM18].

In Section 3.6, we show a more general effectiveness result: Theorem 54 states that *presentable* signatures, which form a superclass of algebraic signatures, are effective.

### 3.4.2 Modularity

In this section, we study the problem of how to merge two syntax extensions. Our answer, a “modularity” result (Theorem 51), 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  $\Sigma_1$  and  $\Sigma_2$  specify two extensions of the signature  $\Sigma_0$ , and  $\Sigma$  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  $f^\Sigma \text{Mon}^\Sigma$  of models:

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

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

**Theorem 51** (`pushout_in_big_rep`). *Modularity holds in  $\int^\Sigma \text{Mon}^\Sigma$ , in the sense that given a pushout square of effective signatures as above, the associated square in  $\int^\Sigma \text{Mon}^\Sigma$  is a pushout again.*

The proof uses, in particular, the following fact:

**Lemma 52** (`rep_cleaving`). *The projection  $\pi : \int^\Sigma \text{Mon}^\Sigma \rightarrow \text{Sig}$  is a Grothendieck fibration. In particular, given a morphism  $f : \Upsilon \rightarrow \Sigma$  of signatures, the pullback map defined in Definition 44 extends to a functor*

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

Note that Theorem 51 does *not* say that a pushout of effective signatures is effective again; it only tells us that if all of the signatures in a pushout square are effective, then the syntax generated by the pushout is the pushout of the syntaxes. In general, we do not know whether a colimit (or even a binary coproduct) of effective signatures is effective again.

In Section 3.6 we study *presentable* signatures, which we show to be effective and closed under colimits.

## 3.5 Recursion

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

### 3.5.1 Example: Translation of intuitionistic logic into linear logic

We start with an elementary example of translation of syntaxes using initiality, namely the translation of second-order intuitionistic logic into second-order linear logic [Gir87, page 6]. The syntax of second-order intuitionistic logic can be defined with one unary operator  $\neg$ , three binary operators  $\vee$ ,  $\wedge$  and  $\Rightarrow$ , and two binding operators  $\forall$  and  $\exists$ . The associated (algebraic) signature is  $\Sigma_{LJ} = \Theta + 3 \times \Theta^2 + 2 \times \Theta'$ . As for linear logic, there are four constants  $\top$ ,  $\perp$ ,  $0$ ,  $1$ , two unary operators  $!$  and  $?$ , five binary operators  $\&$ ,  $\wp$ ,  $\otimes$ ,  $\oplus$ ,  $\multimap$  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  $\Sigma_{LJ}$  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  $\Sigma_{LL}$  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  $\alpha$  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  $o(r_\alpha(t)) = r'_\alpha(o(t))$  for each  $\alpha$ .

The definition of  $r'_\alpha$  is then straightforward to devise, following the recursive clauses given on the right:

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

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

### 3.5.2 Example: Computing the set of free variables

As above, we denote by  $\mathcal{P}X$  the powerset of  $X$ . Union gives a composition operator  $\mathcal{P}(\mathcal{P}X) \rightarrow \mathcal{P}X$  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  $\Sigma_{LC}$  in the monad  $\mathcal{P}$ . We take the binary union operator  $\cup : \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$  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  $\text{Maybe}_X^{-1}(S) = S \cap X$ . This defines a morphism of modules  $\text{Maybe}^{-1}: \mathcal{P}' \rightarrow \mathcal{P}$ ; 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  $\Theta'$  in  $\mathcal{P}$ .

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

### 3.5.3 Example: Computing the size of a term

We now consider the problem of computing the “size” of a  $\lambda$ -term, that is, for any set  $X$ , a function  $s_X: \text{LC}(X) \rightarrow \mathbb{N}$  such that

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

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

$$gs: \prod_{X: \text{Set}} \left( \text{LC}(X) \rightarrow (X \rightarrow \mathbb{N}) \rightarrow \mathbb{N} \right)$$

Here, unsurprisingly, we recognize the continuation monad (see also [JG07] for the use of continuation for implementing complicated recursion schemes using initiality)

$$\text{Cont}_{\mathbb{N}} := X \mapsto (X \rightarrow \mathbb{N}) \rightarrow \mathbb{N}$$

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

Now we can define  $gs$  through initiality by endowing the monad  $\text{Cont}_{\mathbb{N}}$  with a struc-

ture of  $\Sigma_{\text{LC}}$ -model as follows.

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

$$c_{\text{app}} : \text{Cont}_{\mathbb{N}} \times \text{Cont}_{\mathbb{N}} \longrightarrow \text{Cont}_{\mathbb{N}}$$

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

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

$$\begin{aligned} c_{\text{abs}}(X) : \text{Cont}'_{\mathbb{N}}(X) &\longrightarrow \text{Cont}_{\mathbb{N}}(X) \\ t &\mapsto (k \mapsto 1 + t(\hat{k})) \end{aligned}$$

which is the desired action of the abstraction signature  $\Theta'$ .

Altogether, the transformations  $c_{\text{app}}$  and  $c_{\text{abs}}$  form the desired action of  $\Sigma_{\text{LC}}$  in  $\text{Cont}_{\mathbb{N}}$  and thus give an initial morphism, i.e., a natural transformation  $\iota : \text{LC} \rightarrow \text{Cont}_{\mathbb{N}}$  which respects the  $\Sigma_{\text{LC}}$ -model structure. Now let  $0_X$  be the function that is constantly zero on  $X$ . Then the sought “size” map  $s : \prod_{X:\text{Set}} \text{LC}(X) \rightarrow \mathbb{N}$  is given by  $s_X(t) = \iota_X(t, 0_X)$ .

### 3.5.4 Example: Counting the number of redexes

We now consider an example of recursive computation: a function  $r$  such that  $r(t)$  is the number of redexes of the  $\lambda$ -term  $t$  of  $\text{LC}(X)$ . Informally, the equations defining  $r$  are

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

In order to compute recursively the number of  $\beta$ -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  $\Sigma_{\text{LC}}$ -action on the monad  $W := \text{Cont}_{\mathbb{N} \times \{0,1\}}$ . We denote by  $\pi_1, \pi_2$  the projections that access the two components of the product  $\mathbb{N} \times \{0,1\}$ .

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

function

$$\begin{aligned} c_{\text{abs}}(X): W'(X) &\longrightarrow W(X) \\ t &\mapsto (k \mapsto (\pi_1(t(\hat{k})), 1)). \end{aligned}$$

Then  $c_{\text{abs}}$  is an action of the abstraction signature  $\Theta'$  in  $W$ .

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

$$\begin{aligned} c_{\text{app}}(X): W(X) \times W(X) &\longrightarrow W(X) \\ (t, u) &\mapsto (k \mapsto (\pi_1(t(k)) + \pi_1(u(k)) + \pi_2(t(k)), 0)). \end{aligned}$$

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

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

## 3.6 Presentations of signatures and syntaxes

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

**Definition 53.** Given a signature  $\Sigma$ , a *presentation*<sup>7</sup> of  $\Sigma$  is given by an algebraic signature  $\Upsilon$  and an epimorphism of signatures  $p: \Upsilon \rightarrow \Sigma$ . In that case, we say that  $\Sigma$  is *presented by*  $p: \Upsilon \rightarrow \Sigma$ .

A signature for which a presentation exists is called *presentable*.

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

*Remark.* By definition, any construction which can be encoded through a presentable signature  $\Sigma$  can alternatively be encoded through any algebraic signature “presenting”  $\Sigma$ . 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:

---

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

**Theorem 54** (`PresentableIsEffective`). *Any presentable signature is effective.*

The proof is discussed in Section 3.7.

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

**Theorem 55** (`is_right_adjoint_functor_of_reps_from_pw_epi_choice`). *We assume the axiom of choice. Let  $\Sigma$  be a signature, and let  $p : \Upsilon \rightarrow \Sigma$  be a presentation of  $\Sigma$ . Then the functor  $p^* : \text{Mon}^\Sigma \rightarrow \text{Mon}^\Upsilon$  has a left adjoint.*

In the proof of Theorem 55, the axiom of choice is used to show that endofunctors on `Set` preserve epimorphisms.

Theorem 54 follows from Theorem 55 since the left adjoint  $p^! : \text{Mon}^\Upsilon \rightarrow \text{Mon}^\Sigma$  preserves colimits, in particular, initial objects. However, Theorem 54 is proved in Section 3.7 without appealing to the axiom of choice: there, only some specific endofunctor on `Set` is considered, for which preservation of epimorphisms can be proved without using the axiom of choice.

**Definition 56.** We call a syntax *presentable* if it is generated by a presentable signature.

Next, we give important examples of presentable signatures:

**Theorem 57.** *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  $\Theta$ , see `has_binprodR_isPresentable`)*

*Proof.* Items 1–2 are easy to prove. For Item 3, if  $\Sigma_1$  and  $\Sigma_2$  are presented by  $\coprod_i \Upsilon_i$  and  $\coprod_j \Phi_j$  respectively, then  $\Sigma_1 \times \Sigma_2$  is presented by  $\coprod_{i,j} \Upsilon_i \times \Phi_j$ .  $\square$

**Corollary 58.** *Any colimit of algebraic signatures is effective.*

*Proof.* A colimit of an algebraic is presentable, by Theorem 57, hence effective, by Theorem 54  $\square$



### 3.7 Proof of Theorem 54

In this section, we prove Theorem 54. 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 54 rests on the more technical Lemma 63 below.

**Proposition 59** (`epiSig_equiv_pwEpi_SET`). *Epimorphisms of signatures are exactly pointwise epimorphisms.*

*Proof.* In any category, a morphism  $f : a \rightarrow b$  is an epimorphism if and only if the following diagram is a pushout diagram ([ML98, Exercise III.4.4]) :

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

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

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  $X : \text{Set}_0$ ,  $\sim_X$  is an equivalence relation on  $RX$  such that, for any  $f : X \rightarrow Y$ , the function  $R(f)$  maps related elements in  $RX$  to related elements in  $RY$ . Taking the pointwise quotient, we obtain a quotient  $\pi : R \rightarrow \bar{R}$  in the functor category, satisfying the usual universal property. We want to equip  $\bar{R}$  with a monad structure that upgrades  $\pi : R \rightarrow \bar{R}$  into a quotient in the category of monads. In particular, this means that we need to fill in the square

$$\begin{array}{ccc} R \cdot R & \xrightarrow{\mu} & R \\ \pi \cdot \pi \downarrow & & \downarrow \pi \\ \bar{R} \cdot \bar{R} & \xrightarrow{\bar{\mu}} & \bar{R} \end{array}$$

with a suitable  $\bar{\mu} : \bar{R} \cdot \bar{R} \rightarrow \bar{R}$  satisfying the monad laws. But  $\pi$  is epi, and hence so is  $\pi \cdot \pi = \pi \bar{R} \circ R \pi$  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 60** (`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  $\pi : R \rightarrow \bar{R}$  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 61.** An *epi-signature* is a signature  $\Sigma$  that preserves the epimorphicity in the category of endofunctors on `Set`: for any monad morphism  $f : R \rightarrow S$ , 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 62** (`BindingSigAreEpiSig`). Any algebraic signature is an epi-signature.

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

**Lemma 63** (`push_initiality`). *Let  $\Upsilon$  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  $F : \Upsilon \rightarrow \Sigma$  be a morphism of signatures. Suppose that  $\Upsilon$  is an epi-signature and  $F$  is an epimorphism. Then  $\Sigma$  is effective.*

*Proof sketch.* As before, we denote by  $\hat{\Upsilon}$  the initial  $\Upsilon$ -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  $\Upsilon$ -models  $i : \hat{\Upsilon} \rightarrow F^*S$  with  $S$  a  $\Sigma$ -model and  $F^*S$  the  $\Upsilon$ -model induced by  $F : \Upsilon \rightarrow \Sigma$ .

By Lemma 60, as  $\hat{\Upsilon}$  preserves epimorphisms, we obtain the quotient monad, which we call  $\hat{\Upsilon}/F$ , and the epimorphic projection  $\pi : \hat{\Upsilon} \rightarrow \hat{\Upsilon}/F$ . We now equip  $\hat{\Upsilon}/F$  with a  $\Sigma$ -action, and show that the induced model is initial, in four steps:

- (i) We equip  $\hat{\Upsilon}/F$  with a  $\Sigma$ -action, i.e., with a morphism of  $\hat{\Upsilon}/F$ -modules  $m_{\hat{\Upsilon}/F} : \Sigma(\hat{\Upsilon}/F) \rightarrow \hat{\Upsilon}/F$ . We define  $u : \Upsilon(\hat{\Upsilon}) \rightarrow \Sigma(\hat{\Upsilon}/F)$  as  $u = F_{\hat{\Upsilon}/F} \circ \Upsilon(\pi)$ . Then  $u$  is epimorphic, by composition of epimorphisms and by using Proposition 59. Let  $m_{\hat{\Upsilon}} : \Upsilon(\hat{\Upsilon}) \rightarrow \hat{\Upsilon}$  be the action of the initial model of  $\Upsilon$ . We define  $m_{\hat{\Upsilon}/F}$  as

the unique morphism making the following diagram commute in the category of endofunctors on Set:

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

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  $(\Sigma(\pi) \circ F_{\hat{\Upsilon}}) \cdot \pi : \Upsilon(\hat{\Upsilon}) \cdot \hat{\Upsilon} \rightarrow \Sigma(\hat{\Upsilon}/F) \cdot \hat{\Upsilon}/F$  (this is where the preservation of epimorphisms by  $\Upsilon(\hat{\Upsilon})$  is required) and unfolding the definitions.

- (ii) Now,  $\pi$  can be seen as a morphism of  $\Upsilon$ -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  $\Sigma$ -models.

- (iii) Given a  $\Sigma$ -model  $(S, m_s)$ , the initial morphism of  $\Upsilon$ -models  $i_S : \hat{\Upsilon} \rightarrow F^*S$  induces a monad morphism  $\iota_S : \hat{\Upsilon}/F \rightarrow S$ . We need to show that the morphism  $\iota$  is a morphism of  $\Sigma$ -models. Pre-composing the involved diagram by the epimorphism  $\Sigma(\pi) \circ F_{\hat{\Upsilon}} : \Upsilon(\hat{\Upsilon}) \rightarrow \Sigma(\hat{\Upsilon}/F)$  and unfolding the definitions shows that  $\iota_S : \hat{\Upsilon}/F \rightarrow S$  is a morphism of  $\Sigma$ -models.
- (iv) We show that  $\iota_S$  is the only morphism  $\hat{\Upsilon}/F \rightarrow S$ . Let  $g$  be such a morphism. Then  $g \circ \pi : \hat{\Upsilon} \rightarrow S$  defines a morphism in the category of  $\Upsilon$ -models. Uniqueness of  $i_S$  yields  $g \circ \pi = i_S$ , and by uniqueness of the diagram defining  $\iota_S$  it follows that  $g = \iota'_S$ .  $\square$

**Lemma 64** (`algebraic_model_Epi` and `BindingSig_on_model_isEpi`). *Let  $\Sigma$  be an algebraic signature. Then  $\hat{\Sigma}$  and  $\Sigma(\hat{\Sigma})$  preserve epimorphisms.*

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

*Proof of Theorem 54.* Let  $p : \Upsilon \rightarrow \Sigma$  be a presentation of  $\Sigma$ . We need to construct a representation for  $\Sigma$ .

As the signature  $\Upsilon$  is algebraic, it is effective (by Theorem 50) and is an epi-signature (by Example 62). We can thus instantiate Lemma 63 to see that  $\Sigma$  is effective, thanks to Lemma 64.  $\square$

## 3.8 Constructions of presentable signatures

Complex signatures are naturally built as the sum of basic components, generally referred as “arities” (which in our settings are signatures themselves, see remark after Example 34). Thanks to Theorem 57, Item 2, direct sums (or, indeed, any colimit) of presentable signatures are presentable, hence effective by Theorem 54.

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

### 3.8.1 Post-composition with a presentable functor

A functor  $F : \mathbf{Set} \rightarrow \mathbf{Set}$  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 65.** Let  $G : \mathbf{Set} \rightarrow \mathbf{Set}$  be a functor. A *presentation of  $G$*  is a pair consisting of a polynomial functor  $F : \mathbf{Set} \rightarrow \mathbf{Set}$  and an epimorphism  $p : F \rightarrow G$ . The functor  $G$  is called *presentable* if there is a presentation of  $G$ .

**Proposition 66.** *Given a presentable functor  $G$ , the signature  $G \cdot \Theta$  is presentable.*

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

**Proposition 67.** *Here we assume the axiom of excluded middle. An endofunctor on  $\mathbf{Set}$  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  $\omega$ -accessible functors are exactly the finitary ones.  $\square$

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  $\text{square} : \text{Set} \rightarrow \text{Set}$  mapping a set  $X$  to  $X \times X$ ; it is polynomial. The associated signature  $\text{square} \cdot \Theta$  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  $\pi$ -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  $\mathcal{S}_2 : \text{Set} \rightarrow \text{Set}$  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  $\text{square} \rightarrow \mathcal{S}_2$ . By Proposition 66, the signature  $\mathcal{S}_2 \cdot \Theta$  is presentable; it encodes a commutative binary operator.

**Example: Adding a maximum operator**

Let  $\text{list} : \text{Set} \rightarrow \text{Set}$  be the functor associating, to any set  $X$ , the set  $\text{list}(X)$  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:

$$\text{max} : \text{list}(\mathbb{N}) \rightarrow \mathbb{N}$$

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

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  $\mathcal{P}_{\text{fin}} : \text{Set} \rightarrow \text{Set}$  associating, to a set  $X$ , the set  $\mathcal{P}_{\text{fin}}(X)$  of its finite subsets. This functor is presented by the epimorphism  $\text{list} \rightarrow \mathcal{P}_{\text{fin}}$ .

By Proposition 66, the signature  $\mathcal{P}_{\text{fin}} \cdot \Theta$  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  $R\langle \_ \rangle : \text{Set} \rightarrow \text{Set}$  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  $\text{list} : \text{Set} \rightarrow \text{Set}$ , and the epimorphism

$$p : \text{list} \longrightarrow \mathbb{N}\langle \_ \rangle$$

$$p_X([x_1, \dots, x_n]) := x_1 + \dots + x_n .$$

By Proposition 66, this yields a presentable signature, which we call  $\mathbb{N}\langle \Theta \rangle$ .

The Differential Lambda Calculus (DLC) [ER03] of Ehrhard and Regnier is a lambda calculus with operations suitable to express differential constructions. The calculus is parametrized by a semiring  $R$ ; again we restrict to  $R = (\mathbb{N}, +, \times)$ .

DLC has a binary “application” operator, written  $(s)t$ , where  $s \in T$  is an element of the inductively defined set  $T$  of terms and  $t \in \mathbb{N}\langle T \rangle$  is an element of the free  $(\mathbb{N}, +, \times)$ -module. This operator is thus specified by the presentable signature  $\Theta \times \mathbb{N}\langle \Theta \rangle$ .

### 3.8.2 Example: Adding a syntactic closure operator

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

Let us start with binding a fixed number  $k$  of fresh variables. The elementary signature  $\Theta^{(k)}$  already specifies an operation that binds  $k$  variables. However, this encoding does not reflect invariance under variable permutation. To enforce this invariance, it

suffices to quotient the signature  $\Theta^{(k)}$  with respect to the action of the group  $\mathfrak{S}_k$  of permutations of the set  $k$ , that is, to consider the colimit of the following one-object diagram:

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

where  $\sigma$  ranges over the elements of  $\mathfrak{S}_k$ . We denote by  $\mathcal{S}^{(k)}\Theta$  the resulting signature presented by the projection  $\Theta^{(k)} \rightarrow \mathcal{S}^{(k)}\Theta$ . By universal property of the quotient, a model of it consists of a monad  $R$  with an action  $m : R^{(k)} \rightarrow R$  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 co-product  $\coprod_k \mathcal{S}^{(k)}\Theta$ . 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  $\text{colim}_k \mathcal{S}^{(k)}\Theta$  which, as a colimit of presentable signatures, is presentable (Theorem 57, Item 2).

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

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

### 3.8.3 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 sig-

nature 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 31) the (unary) substitution  $\sigma_R : R' \times R \rightarrow R$ . More generally, we have the sequence of substitution operations

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

We say that  $\text{subst}_p$  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  $\text{subst}_1$  is the original  $\sigma_R$ .)

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  $\text{subst}_p$  is invariant under this action.

Thus, if we fix an integer  $p$ , there are two ways to internalise  $\text{subst}_p$  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  $m : R^{(p)} \times R^p \rightarrow R$  satisfying the desired invariance.

Before turning to the encoding of the entire series  $(\text{subst}_p)_{p \in \mathbb{N}}$ , 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  $u : I_p \rightarrow I_q$ , where  $I_n$  denotes the set  $\{1, \dots, n\}$ . 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  $\text{subst}_q$ ; while letting  $u$  act contravariantly on the second factor leads us into  $R^{(p)} \times R^p$  where we can apply  $\text{subst}_p$ . The good news is that we obtain the same result. More precisely, the



following diagram is commutative:

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

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

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

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

Abstracting over  $R$ , we obtain the following:

**Definition 68.** *Integrated substitution*

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

is the signature morphism obtained by abstracting over  $R$  the linear morphisms  $\text{isubst}_R$ .

Thus, if we want to internalise the whole sequence  $(\text{subst}_p)_{p:\mathbb{N}}$  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

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

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

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

### 3.8.4 Example: Adding a coherent fixed-point operator

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

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

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

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

Let us start with the unary case.

**Definition 69.** A *unary fixed-point operator* for a monad  $R$  is a module morphism  $f$  from  $R'$  to  $R$  that makes the following diagram commute,

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

where  $\sigma$  is the substitution morphism defined in Lemma 31.

Accordingly, the signature for a syntactic unary fixpoint operator is  $\Theta'$ , ignoring the commutation requirement (which we address later in Section 4.4.4, 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  $LC_{\beta\eta}$  of the lambda-calculus modulo  $\beta$ - and  $\eta$ -equivalence. How can we relate the above notion to the classical notion of fixed-point combinator? Terms are built out of two constructions,  $\text{app} : LC_{\beta\eta} \times LC_{\beta\eta} \rightarrow LC_{\beta\eta}$  and  $\text{abs} : LC'_{\beta\eta} \rightarrow LC_{\beta\eta}$ . A fixed-point combinator is a term  $Y$  satisfying, for any (possibly open) term  $t$ , the equation

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

Given such a combinator  $Y$ , we define a module morphism  $\hat{Y} : LC'_{\beta\eta} \rightarrow LC_{\beta\eta}$ . It associates, to any term  $t$  depending on an additional variable  $*$ , the term  $\hat{Y}(t) := \text{app}(Y, \text{abs } t)$ . This term satisfies  $t\{* := \hat{Y}(t)\} = \hat{Y}(t)$ , which is precisely the diagram of Definition 69 for a unary fixed-point operator. Thus,  $\hat{Y}$  is a unary fixed-point operator for the monad  $LC_{\beta\eta}$ . Conversely, we have:

**Proposition 70.** *Any fixed-point combinator in  $LC_{\beta\eta}$  comes from a unique fixed-point operator.*

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

A closed lambda term  $t$  is mapped to the morphism  $u \mapsto \hat{t} u := \text{app}(t, \text{abs } u)$ . 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  $LC'_{\beta\eta}$  to  $LC_{\beta\eta}$  induces a closed term  $Y_f := \text{abs}(f_1(\text{app}(*, **)))$  where  $f_1 : LC_{\beta\eta}(\{*, **\}) \rightarrow LC_{\beta\eta}(\{*\})$ .

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  $t \in LC_{\beta\eta}X$ , then we have

$$\text{app}(Y_f, t) = \text{app}(\text{abs } f_X(\text{app}(*, **)), t) \tag{3.5}$$

$$= f_X(\text{app}(t, **)) \tag{3.6}$$

$$= \text{app}(t, f_X(\text{app}(t, **))) \tag{3.7}$$

$$= \text{app}(t, \text{app}(Y_f, t)) \tag{3.8}$$

where (3.5) comes from the definition of  $Y_f$  (and naturality of  $f$ ). Equality (3.6) follows from  $\beta$ -reduction, Equality 3.7 from the definition of a fixed-point operator. Finally,

Equality 3.8 comes from the equality  $\text{app}(Y_f, t) = f_X(\text{app}(t, **))$ , which is obtained by chaining the equalities from (3.5) to (3.6). This concludes the construction of the bijection.  $\square$

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

**Definition 71.** • A *rough  $n$ -ary fixed-point operator* for a monad  $R$  is a module morphism  $f : (R^{(n)})^n \rightarrow R^n$  making the following diagram commute:

$$\begin{array}{ccc} (R^{(n)})^n & \xrightarrow{id_{(R^{(n)})^n}, f, \dots, f} & (R^{(n)})^n \times (R^n)^n \\ f \downarrow & & \parallel \\ R^n & \xleftarrow{(\text{subst}_n)^n} & (R^{(n)} \times R^n)^n \end{array}$$

where  $\text{subst}_n$  is the  $n$ -substitution as in Section 3.8.3.

- An  *$n$ -ary fixed-point operator* is just a rough  $n$ -ary fixed-point operator which is furthermore invariant under the natural action of the permutation group  $\mathfrak{S}_n$ .

The type of  $f$  above is canonically isomorphic to

$$(R^{(n)})^n + (R^{(n)})^n + \dots + (R^{(n)})^n \rightarrow R,$$

which we abbreviate to<sup>8</sup>  $n \times (R^{(n)})^n \rightarrow R$ .

Accordingly, a natural signature for encoding a *syntactic*<sup>9</sup> rough  $n$ -ary fixpoint operator is  $n \times (\Theta^{(n)})^n$ .

Similarly, a natural signature for encoding a syntactic  $n$ -ary fixpoint operator is  $(n \times (\Theta^{(n)})^n) / \mathfrak{S}_n$  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 3.8.3, which we now introduce.

---

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

9. The adjective *syntactic* means here that we do not deal with the equation.

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

$$\begin{array}{ccc}
 p \times (R^{(p)})^q & \xrightarrow{p \times (R^{(p)})^u} & p \times (R^{(p)})^p \\
 u \times (R^{(u)})^q \downarrow & & \downarrow \text{fix}_p \\
 q \times (R^{(q)})^q & \xrightarrow{\text{fix}_q} & R
 \end{array} \tag{3.9}$$

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

**Definition 72.** Given a monad  $R$ , we define a *coherent fixed-point operator* on  $R$  to be a module morphism from  $\int^{n:\mathbb{N}} \underline{n} \times (R^{(\underline{n})})^{\overline{n}}$  to  $R$  where, for every  $n \in \mathbb{N}$ , the  $n$ -th component is a (rough)<sup>10</sup>  $n$ -ary fixpoint operator.

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

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

to  $\Sigma$ .

10. As in Section 3.8.3, invariance follows from coherence.



# ALGEBRAIC 2-SIGNATURES FOR MONADS

---

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

The presentable signatures of Chapter 3 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 3 are now referred to as 1-signatures and 1-models. The 2-signatures can now be thought of as signatures over the category of monads, in the sense of Chapter 2. More precisely, they consist of two components: the first one is a single arity (corresponding to the underlying 1-signature of the 2-signature), and the second one is a family of equational arities.

In this chapter, we also adopt an alternative viewpoint on this work: our 2-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 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.

## 4.1 Introduction

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

If  $p : \hat{\Sigma} \rightarrow R$  is a monad epimorphism, we understand that  $R$  is generated by a family of operations whose binding arities are given by  $\Sigma$ , 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 3.2), and we have an obvious epimorphism  $p : \hat{\Sigma} \rightarrow \text{LC}_{\beta\eta}$ , where  $\text{LC}_{\beta\eta}$  is the monad of lambda-terms modulo  $\beta$  and  $\eta$ . In order to cover such equations, the approach in the first-order case suggests to identify  $p$  as the coequalizer of a pair of parallel 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  $\beta$ -equation of the monad  $\text{LC}_{\beta\eta}$ . It should say that for each set  $X$ , the following two maps from  $\hat{\Sigma}(X + \{*\}) \times \hat{\Sigma}(X)$  to  $\hat{\Sigma}(X)$ ,

- $(t, u) \mapsto \text{app}(\text{abs}(t), u)$
- $(t, u) \mapsto t\{* \mapsto u\}$

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

$$T \rightrightarrows^f \hat{\Sigma} \xrightarrow{p} R \quad (4.1)$$

where  $\Sigma$  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  $f$ <sup>1</sup>. To this end, recall that we introduced the category  $\text{Mon}^\Sigma$  “of models of  $\Sigma$ ”, whose objects are monads “equipped with an action of  $\Sigma$ ” (Definition 42). 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 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  $(\Sigma, E)$ , where  $E$  is a family of parametric equations as above, does the subcategory of models of  $\Sigma$  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 104).

This provides a construction of a monad from an algebraic 2-signature, and we prove furthermore (see Theorem 99) 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 51 for 1-signatures.

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

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

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

---

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

## 4.2 2-Signatures and their models

In this section we study *2-signatures* and *models of 2-signatures*. A 2-signature is a pair of a 1-signature and a family of *equations* over it. Alternatively, in the terminology of Chapter 2, a 2-signature can be thought of as a signature  $S_1, E_2$  over the category of monads where  $S_1$  is a signature consisting of a single arity, and  $E_2$  is a family of equational  $S_1$ -arities.

### 4.2.1 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. They can be thought of as particular arities in the sense of Chapter 2 (Remark 79). 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  $\Sigma$ , that we instantiate in the examples.

**Definition 73.** We define a  $\Sigma$ -**module** to be a functor  $T$  from the category of models of  $\Sigma$  to the category  $\int \text{Mod}$  commuting with the forgetful functors to the category  $\text{Mon}$  of monads,

$$\begin{array}{ccc} \text{Mon}^\Sigma & \xrightarrow{T} & \int \text{Mod} \\ & \searrow & \swarrow \\ & \text{Mon} & \end{array}$$

**Example 74.** To each 1-signature  $\Psi$  is associated, by precomposition with the projection from  $\text{Mon}^\Sigma$  to  $\text{Mon}$ , a  $\Sigma$ -module still denoted  $\Psi$ . All the  $\Sigma$ -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  $\Sigma$ -module**  $\Theta$ , and, more generally, for any natural number  $n \in \mathbb{N}$ , a  $\Sigma$ -module  $\Theta^{(n)}$ . Also we have another fundamental  $\Sigma$ -module (arising in this way from)  $\Sigma$  itself.

**Definition 75.** Let  $S$  and  $T$  be  $\Sigma$ -modules. We define a **morphism of  $\Sigma$ -modules** from  $S$  to  $T$  to be a natural transformation from  $S$  to  $T$  which becomes the identity when postcomposed with the forgetful functor  $\int \text{Mod} \rightarrow \text{Mon}$ .

**Example 76.** Each 1-signature morphism  $\Psi \rightarrow \Phi$  upgrades into a morphism of  $\Sigma$ -modules. Further in that vein, there is a morphism of  $\Sigma$ -modules  $\tau^\Sigma : \Sigma \rightarrow \Theta$ . It is given, on a model  $(R, m)$  of  $\Sigma$ , by  $m : \Sigma(R) \rightarrow R$ . (Note that it does not arise from a morphism of 1-signatures.) When the context is clear, we write simply  $\tau$  for this morphism, and call it the **tautological morphism of  $\Sigma$ -modules**.

**Proposition 77.** *Our  $\Sigma$ -modules and their morphisms, with the obvious composition and identity, form a category.*

**Definition 78.** We define a  $\Sigma$ -equation to be a pair of parallel morphisms of  $\Sigma$ -modules. We also write  $e_1 = e_2$  for the  $\Sigma$ -equation  $e = (e_1, e_2)$ .

**Remark 79.** Any  $\Sigma$ -equation  $e_1 = e_2 : \Psi \rightarrow \Phi$  induces an equational  $S_\Sigma$ -arity  $A_{e_1, e_2} := (D, a, u, v)$  in the sense of Chapter 2, where

- $D$  is the category whose objects are models  $R$  of  $\Sigma$  equipped with a  $R$ -module morphism  $h : \Psi(R) \rightarrow \Phi(R)$ , and morphisms between  $(R, h)$  and  $(S, i)$  are model morphisms  $f : R \rightarrow S$  such that the following diagram commutes in the category of functors

$$\begin{array}{ccc} \Psi(R) & \xrightarrow{h} & \Phi(R) \\ \Psi(f) \downarrow & & \downarrow \Phi(f) \\ \Psi(S) & \xrightarrow{i} & \Phi(S) \end{array}$$

- $a : D \rightarrow \text{Mon}^\Sigma$  is the canonical forgetful functor;
- $u$  maps a model  $R$  to  $(R, e_{1,R})$ ;
- $v$  maps a model  $R$  to  $(R, e_{2,R})$ .

**Example 80** (Commutativity of a binary operation). Here we instantiate our fixed 1-signature as follows:  $\Sigma := \Theta \times \Theta$ . In this case, we say that  $\tau$  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  $\text{swap} : \Theta^2 \rightarrow \Theta^2$  that exchanges the two components of the direct product. Again by Example 76, we have an induced morphism of  $\Sigma$ -modules, still denoted  $\text{swap}$ .

Then, the  $\Sigma$ -equation for commutativity is given by the two morphisms of  $\Sigma$ -modules

$$\begin{array}{ccc} \Theta^2 & \xrightarrow{\text{swap}} & \Theta^2 \xrightarrow{\tau} \Theta \\ \Theta^2 & \xrightarrow{\tau} & \Theta \end{array}$$

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

For the example of the lambda calculus with  $\beta$ - and  $\eta$ -equality (given in Example 82), we need to introduce *currying*:

**Definition 81.** By abstracting over the base monad  $R$  the adjunction in the category of  $R$ -modules of Proposition 32, we can perform **currying** of morphisms of 1-signatures: given a morphism of signatures  $\Sigma_1 \times \Theta \rightarrow \Sigma_2$  it produces a new morphism  $\Sigma_1 \rightarrow \Sigma'_2$ . By Example 74, currying acts also on morphisms of  $\Sigma$ -modules.

Conversely, given a morphism of 1-signatures (resp.  $\Sigma$ -modules)  $\Sigma_1 \rightarrow \Sigma'_2$ , we can define the **uncurried** map  $\Sigma_1 \times \Theta \rightarrow \Sigma_2$ .

**Example 82** ( $\beta$ - and  $\eta$ -conversions). Here we instantiate our fixed 1-signature as follows:  $\Sigma_{\text{LC}} := \Theta \times \Theta + \Theta'$ . This is the 1-signature of the lambda calculus. We break the tautological  $\Sigma$ -module morphism into its two pieces, namely  $\text{app} := \tau \circ \text{inl} : \Theta \times \Theta \rightarrow \Theta$  and  $\text{abs} := \tau \circ \text{inr} : \Theta' \rightarrow \Theta$ . Applying currying to  $\text{app}$  yields the morphism  $\text{app}_1 : \Theta \rightarrow \Theta'$  of  $\Sigma_{\text{LC}}$ -modules. The usual  $\beta$  and  $\eta$  relations are implemented in our formalism by two  $\Sigma_{\text{LC}}$ -equations that we call  $e_\beta$  and  $e_\eta$  respectively:

$$e_\beta : \begin{array}{ccc} \Theta' & \xrightarrow{\text{abs}} & \Theta \xrightarrow{\text{app}_1} \Theta' \\ \Theta' & \xrightarrow{1} & \Theta' \end{array} \quad \text{and} \quad e_\eta : \begin{array}{ccc} \Theta & \xrightarrow{\text{app}_1} & \Theta' \xrightarrow{\text{abs}} \Theta \\ \Theta & \xrightarrow{1} & \Theta \end{array}.$$

## 4.2.2 2-signatures and their models

**Definition 83.** A **2-signature** is a pair  $(\Sigma, E)$  of a 1-signature  $\Sigma$  and a family  $E$  of  $\Sigma$ -equations.

**Remark 84.** Any 2-signature  $(\Sigma, E)$  induces a signature  $S_\Sigma, \mathcal{E}_E$  over the category of monads, where  $S_\Sigma$  is defined in Remark 35, and  $\mathcal{E}_E = (A_{e_1, e_2})_{e_1 = e_2 \in E}$  (see Remark 79).

**Example 85.** The 2-signature for a commutative binary operation is  $(\Theta^2, \tau \circ \text{swap} = \tau)$  (cf. Example 80).

**Example 86.** The 2-signature of the lambda calculus modulo  $\beta$ - and  $\eta$ -equality is  $\Upsilon_{\text{LC}_{\beta\eta}} = (\Theta \times \Theta + \Theta', \{e_\beta, e_\eta\})$ , where  $e_\beta, e_\eta$  are the  $\Sigma_{\text{LC}}$ -equations defined in Example 82.

**Definition 87** (*satisfies\_equation*). We say that a model  $M$  of  $\Sigma$  **satisfies the  $\Sigma$ -equation**  $e = (e_1, e_2)$  if  $e_1(M) = e_2(M)$ . If  $E$  is a family of  $\Sigma$ -equations, we say that a model  $M$  of  $\Sigma$  **satisfies**  $E$  if  $M$  satisfies each  $\Sigma$ -equation in  $E$ .

**Definition 88.** Given a monad  $R$  and a 2-signature  $\Upsilon = (\Sigma, E)$ , an **action of  $\Upsilon$  in  $R$**  is an action of  $\Sigma$  in  $R$  such that the induced 1-model satisfies all the equations in  $E$ .

**Definition 89** (*category\_model\_equations*). For a 2-signature  $(\Sigma, E)$ , we define the **category  $\text{Mon}^{(\Sigma, E)}$  of models of  $(\Sigma, E)$**  to be the category of models of  $S_\Sigma, \mathcal{E}_E$  in the sense of Chapter 2 (see Remark 84): it is the full subcategory of the category of models of  $\Sigma$  whose objects are models of  $\Sigma$  satisfying  $E$ , or equivalently, monads equipped with an action of  $(\Sigma, E)$ .

**Example 90.** A model of the 2-signature  $\Upsilon_{\text{LC}_{\beta\eta}} = (\Theta \times \Theta + \Theta', \{e_\beta, e_\eta\})$  is given by a model  $(R, \text{app}^R : R \times R \rightarrow R, \text{abs}^R : R' \rightarrow R)$  of the 1-signature  $\Sigma_{\text{LC}}$  such that  $\text{app}_1^R \cdot \text{abs}^R = 1_{R'}$  and  $\text{abs}^R \cdot \text{app}_1^R = 1_R$  (see Example 82).

**Definition 91.** A 2-signature  $(\Sigma, E)$  is said to be **effective** if its category of models  $\text{Mon}^{(\Sigma, E)}$  has an initial object, denoted  $(\widehat{\Sigma, E})$ .

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

### 4.2.3 Modularity for 2-signatures

In this section, we define the category  $2\text{Sig}$  of 2-signatures and the category  $2\text{Mod}$  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

$$\begin{array}{ccc}
 2\text{Mod} & \begin{array}{c} \xrightarrow{U_{\text{Mod}}} \\ \top \\ \xleftarrow{F_{\text{Mod}}} \end{array} & \text{Mod} \\
 \downarrow 2\pi & & \downarrow \pi \\
 2\text{Sig} & \begin{array}{c} \xrightarrow{U_{\text{Sig}}} \\ \top \\ \xleftarrow{F_{\text{Sig}}} \end{array} & \text{Sig}
 \end{array}$$

where

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

As a simple consequence of this data, we obtain, in Theorem 99, 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<sup>2</sup>.

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

**Definition 92** (`TwoSig_category`). Given 2-signatures  $(\Sigma_1, E_1)$  and  $(\Sigma_2, E_2)$ , a **morphism of 2-signatures from  $(\Sigma_1, E_1)$  to  $(\Sigma_2, E_2)$**  is a morphism of 1-signatures  $m : \Sigma_1 \rightarrow \Sigma_2$  such that for any model  $M$  of  $\Sigma_2$  satisfying  $E_2$ , the  $\Sigma_1$ -model  $m^*M$  satisfies  $E_1$ .

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

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

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

**Lemma 93** (`TwoSig_OneSig_is_right_adjoint, OneSig_TwoSig_fully_faithful`). We have  $F_{\text{Sig}} \dashv U_{\text{Sig}}$ . Furthermore,  $U_{\text{Sig}}$  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 94** (`TwoSig_PushoutsSET`). The category 2Sig has pushouts.

---

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

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 95.** *The category  $2\text{Sig}$  is cocomplete and  $U_{\text{Sig}}$  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 the 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 96.** The category  $\int^{(\Sigma, E)} \text{Mon}^{(\Sigma, E)}$ , or  $2\text{Mod}$  for short, has, as objects, pairs  $((\Sigma, E), M)$  of a 2-signature  $(\Sigma, E)$  and a model  $M$  of  $(\Sigma, E)$ .

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  $m : (\Sigma_1, E_1) \rightarrow (\Sigma_2, E_2)$  of 2-signatures and a morphism  $f : M_1 \rightarrow m^* M_2$  of  $(\Sigma_1, E_1)$ -models (or, equivalently, of  $\Sigma_1$ -models).

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

**Lemma 97** (`TwoMod_OneMod_is_right_adjoint`, `OneMod_TwoMod_fully_faithful`). *We have  $F_{\text{Mod}} \dashv U_{\text{Mod}}$ . Furthermore,  $U_{\text{Mod}}$  is a coreflection.*

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

**Proposition 98** (`TwoMod_cleaving`). *The forgetful functor  $2\pi$  from  $2\text{Mod}$  to  $2\text{Sig}$  is a Grothendieck fibration.*

The *modularity result* below is analogous to the modularity result for 1-signatures (Theorem 51):

**Theorem 99** (`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  $2\text{Mod}$  as on the right below, where we only write the second components, omitting the (morphisms of) signatures. This square is a pushout square.

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

Intuitively, the 2-signatures  $\Upsilon_1$  and  $\Upsilon_2$  specify two extensions of the 2-signature  $\Upsilon_0$ , and  $\Upsilon$  is the smallest extension containing both these extensions. By Theorem 99 the initial model of  $\Upsilon$  is the “smallest model containing both the languages generated by  $\Upsilon_1$  and  $\Upsilon_2$ ”.

#### 4.2.4 Initial Semantics for 2-Signatures

We now turn to the problem of constructing the initial model of a 2-signature  $(\Sigma, E)$ . More specifically, we identify sufficient conditions for  $(\Sigma, E)$  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  $\text{Mon}^\Sigma$ .

This leads immediately to our first requirement on  $(\Sigma, E)$ , which is that  $\Sigma$  must be an effective 1-signature. (For instance, we can assume that  $\Sigma$  is an algebraic 1-signature, see Theorem 50.) This is a very natural hypothesis, since in the case where  $E$  is the empty family of  $\Sigma$ -equations, it is obviously a necessary and sufficient condition.

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

**Definition 100.** Given a 1-signature  $\Sigma$ , a  $\Sigma$ -module  $S$  is **nice** if  $S$  sends pointwise epimorphic  $\Sigma$ -model morphisms to pointwise epimorphic module morphisms.

**Definition 101** (*elementary\_equation*). Given a 1-signature  $\Sigma$ , an **elementary  $\Sigma$ -equation** is a  $\Sigma$ -equation such that



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

**Example 102.** Any algebraic 1-signature is nice (Example 62). Thus, any  $\Sigma$ -equation between an algebraic 1-signature and  $\Theta^{(n)}$ , for some natural number  $n$ , is elementary.

**Definition 103.** A 2-signature  $(\Sigma, E)$  is said **algebraic** if  $\Sigma$  is algebraic and  $E$  is a family of elementary equations.

**Theorem 104** (`elementary_equations_on_alg_preserve_initiality`). *Any algebraic 2-signature has an initial model.*

The proof of Theorem 104 is given in Section 4.3.

**Example 105.** The 2-signature of lambda calculus modulo  $\beta$  and  $\eta$  equations given in Example 86 is algebraic. Its initial model is precisely the monad  $\text{LC}_{\beta\eta}$  of lambda calculus modulo  $\beta\eta$  equations.

The instantiation of the formalized Theorem 104 to this 2-signature is done<sup>3</sup> in `LCBetaEta`.

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

**Proposition 106** (`ModEq_Mod_is_right_adjoint`, `ModEq_Mod_fully_faithful`). *Here we assume the axiom of choice. The forgetful functor from the category  $\text{Mon}^{(\Sigma, E)}$  of 2-models of  $(\Sigma, E)$  to the category  $\text{Mon}^\Sigma$  of  $\Sigma$ -models has a left adjoint. Moreover, the left adjoint is a reflector.*

## 4.3 Proof of Theorem 104

Our main technical result on effectiveness is the following Lemma 107. In Theorem 104, we give a much simpler criterion that encompasses all the examples we give.

**Lemma 107** (`elementary_equations_preserve_initiality`). *Let  $(\Sigma, E)$  be a 2-signature such that:*

---

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

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

Then, the category of 2-models of  $(\Sigma, E)$  has an initial object.

Before tackling the proof of Lemma 107, we discuss how to derive Theorem 104 from it, and we prove some auxiliary results.

The “epimorphism” hypotheses of Lemma 107 are used to transfer structure from the initial model  $\hat{\Sigma}$  of the 1-signature  $\Sigma$  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  $\mathbf{Set}$  is split and thus preserved by any functor.
- Condition 5 is a consequence of Condition 4 if  $\Sigma$  sends monads preserving epimorphisms to modules preserving epimorphisms.
- If  $\Sigma$  is algebraic, then Conditions 1, 3, 4 and 5 are satisfied (Example 62 and Lemma 63).

From the remarks above, we derive the simpler and weaker statement of Theorem 104 that covers all our examples, which are algebraic.

This section is dedicated to the proof of the main technical result, Lemma 107. 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 107 uses some quotient constructions that we present now:

**Proposition 108** (`u_monad_def`). *Given a monad  $R$  preserving epimorphisms and a collection of monad morphisms  $(f_i : R \rightarrow S_i)_{i \in I}$ , there exists a quotient monad  $R/(f_i)$  together with a projection  $p^R : R \rightarrow R/(f_i)$ , 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 60.  $\square$

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  $\Sigma$  even without resorting to the axiom of choice.

The above construction can be transported to  $\Sigma$ -models:

**Proposition 109** (`u_rep_def`). *Let  $\Sigma$  be a 1-signature sending epimorphic natural transformations to epimorphic natural transformations, and let  $R$  be a  $\Sigma$ -model such that  $R$  and  $\Sigma(R)$  preserve epimorphisms. Let  $(f_i : R \rightarrow S_i)_{i \in I}$  be a collection of  $\Sigma$ -model morphisms. Then the monad  $R/(f_i)$  has a natural structure of  $\Sigma$ -model and the quotient map  $p^R : R \rightarrow R/(f_i)$  is a morphism of  $\Sigma$ -models. Any morphism  $f_i$  factors through  $p^R$  in the category of  $\Sigma$ -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 108.

Now we are ready to prove the main technical lemma:

*Proof of Lemma 107.* Let  $\Sigma$  be an effective 1-signature, and let  $E$  be a set of elementary  $\Sigma$ -equations. The plan of the proof is as follows:

1. Start with the initial model  $(\hat{\Sigma}, \sigma)$ , with  $\sigma : \Sigma(\hat{\Sigma}) \rightarrow \hat{\Sigma}$ .
2. Construct the quotient model  $\hat{\Sigma}/(f_i)$  according to Proposition 109 where  $(f_i : \hat{\Sigma} \rightarrow S_i)_i$  is the collection of all initial  $\Sigma$ -morphisms from  $\hat{\Sigma}$  to any  $\Sigma$ -model satisfying the equations. We denote by  $\sigma/(f_i) : \Sigma(\hat{\Sigma}/(f_i)) \rightarrow \hat{\Sigma}/(f_i)$  the action of the quotient model.
3. Given a model  $M$  of the 2-signature  $(\Sigma, E)$ , we obtain a morphism  $i_M : \hat{\Sigma}/(f_i) \rightarrow M$  from Proposition 109. Uniqueness of  $i_M$  is shown using epimorphicity of the projection  $p : \hat{\Sigma} \rightarrow \hat{\Sigma}/(f_i)$ . For this, it suffices to show uniqueness of the composition  $i_M \circ p : \hat{\Sigma} \rightarrow M$  in the category of 1-models of  $\Sigma$ , which follows from initiality of  $\hat{\Sigma}$ .
4. The verification that  $(\hat{\Sigma}/(f_i), \sigma/(f_i))$  satisfies the equations is given below. Actually, it follows the same line of reasoning as in the proof of Proposition 108 that  $\hat{\Sigma}/(f_i)$  satisfies the monad equations.

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

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

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 108), 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  $e_1 = e_2$ .  $\square$

## 4.4 Examples of algebraic 2-signatures

We already illustrated our theory by looking at the paradigmatic case of lambda calculus modulo  $\beta$ - and  $\eta$ -equations (Examples 82 and 105). 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 104.

### 4.4.1 Monoids

We begin with an example of monad for a first-order syntax with equations. Given a set  $X$ , we denote by  $M(X)$  the free monoid built over  $X$ . This is a classical example of monad over the category of (small) sets. The monoid structure gives us, for each set  $X$ , two maps  $m_X : M(X) \times M(X) \longrightarrow M(X)$  and  $e_X : 1 \longrightarrow M(X)$  given by the product and the identity respectively. It can be easily verified that  $m : M^2 \longrightarrow M$  and

$e: 1 \rightarrow M$  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  $\Sigma$ -modules (cf. Example 76) into constituent pieces, defining  $m := \tau \circ \text{inl} : \Theta \times \Theta \rightarrow \Theta$  and  $e := \tau \circ \text{inr} : 1 \rightarrow \Theta$ .

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

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

and we denote by  $E$  the family consisting of these three  $\Sigma$ -equations. All are elementary since their codomain is  $\Theta$ , and their domain a product of  $\Theta$ s.

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

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

## 4.4.2 Colimits of algebraic 2-signatures

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

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

**Proposition 110.** *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  $\Xi$  of algebraic 2-signatures can be expressed as a coequalizer of two morphisms  $f, g$  between some algebraic 2-signatures  $(\Sigma_1, E_1)$  and  $(\Sigma_2, E_2)$ ,

$$(\Sigma_1, E_1) \xrightarrow[f]{g} (\Sigma_2, E_2) \xrightarrow{p} \Xi = (\Sigma_3, E_2) .$$

where  $\Sigma_3$  is the coequalizer of the 1-signature morphisms  $f$  and  $g$ . Note that the set of equations of  $\Xi$  is  $E_2$ , by construction of the coequalizer in the category of 2-signatures. Now, consider the algebraic 2-signature  $\Xi' = (\Sigma_2, E_2 + (4.2))$  consisting of the 1-signature  $\Sigma_2$  and the equations of  $E_2$  plus the following elementary equation (see Examples 76 and 102):

$$\begin{array}{c} \Sigma_1 \xrightarrow{f} \Sigma_2 \xrightarrow{\tau^{\Sigma_2}} \Theta . \\ \Sigma_1 \xrightarrow{g} \Sigma_2 \xrightarrow{\tau^{\Sigma_2}} \Theta \end{array} \quad (4.2)$$

We show that  $\text{Mon}^\Xi$  and  $\text{Mon}^{\Xi'}$  are isomorphic. A model of  $\Xi'$  is a monad  $R$  together with an  $R$ -module morphism  $r : \Sigma_2(R) \rightarrow R$  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  $\Sigma_3(R) \rightarrow R$  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  $\Xi$  and the category of models of  $\Xi'$ .  $\square$

This proposition, together with the following corollary, allow us to recover all the examples presented in Chapter 3, 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 111.** *If  $F$  is a finitary endofunctor on  $\text{Set}$ , 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  $\text{Set}$  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.  $\square$

However, we do not know whether we can recover Theorem 54 stating that any presentable 1-signature is effective.

### 4.4.3 Algebraic theories

From the categorical point of view, several fundamental algebraic structures in mathematics can be conveniently and elegantly described using finitary monads. For instance, the category of monoids can be seen as the category of Eilenberg–Moore algebras of the monad of lists. Other important examples, like groups and rings, can be treated analogously. A classical reference on the subject is the work of Manes, where such monads are significantly called *finitary algebraic theories* [Man76, 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 111 says how to encode the 1-signature  $T \cdot \Theta$  as an algebraic 2-signature  $(\Sigma_T, E_T)$ . Models are monads  $R$  together with an  $R$ -linear morphism  $r : T \cdot R \rightarrow R$ .

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  $\tau$  denotes the tautological morphism of  $T \cdot \Theta$ -modules:

$$\begin{array}{ccc} \Theta \xrightarrow{\eta_{T \cdot \Theta}} T \cdot \Theta \xrightarrow{\tau} \Theta & & T \cdot T \cdot \Theta \xrightarrow{\mu_{T \cdot \Theta}} T \cdot \Theta \xrightarrow{\tau} \Theta \\ \Theta \xrightarrow{1} \Theta & & T \cdot T \cdot \Theta \xrightarrow{T\tau} T \cdot \Theta \xrightarrow{\tau} \Theta \end{array} \quad (4.3)$$

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

**Lemma 112.** *Let  $F$  be a finitary endofunctor on  $\text{Set}$ . Then  $F$  preserves epimorphisms.*

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

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

$F(f)$  is epimorphic. By naturality, the following diagram commutes:

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

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  $(\Sigma_T, E'_T)$ , where  $E'_T$  extends the family  $E_T$  with the two elementary equations of Diagram 4.3. This signature allows to enrich any other algebraic 2-signature with the operations of the algebraic theory  $T$ , subject to the relevant equations.

#### 4.4.4 Fixpoint operator

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

Let us start by recalling Definition 69: a **unary fixpoint operator for a monad**  $R$  is a module morphism  $f$  from  $R'$  to  $R$  that makes the following diagram commute, where  $\sigma$  is the substitution morphism defined as the uncurrying (see Definition 81) of the identity morphism on  $\Theta'$ :

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



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

$$e_{\text{fix}} : \begin{array}{ccc} \Theta' & \xrightarrow{\langle 1, \tau \rangle} & \Theta' \times \Theta \xrightarrow{\sigma} \Theta \\ \Theta' & \xrightarrow{\tau} & \Theta \end{array} \quad (4.4)$$

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

## 4.5 Recursion

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

### 4.5.1 Principle of recursion

In our context, the recursion principle is a recipe for constructing a morphism from the monad underlying the initial model of a 2-signature  $\Upsilon = (\Sigma, E)$  to an arbitrary monad  $T$ .

**Proposition 113** (Recursion principle). *Let  $S$  be the monad underlying the initial model of the 2-signature  $\Upsilon$ . To any action  $a$  of  $\Upsilon$  in  $T$  is associated a monad morphism  $\hat{a} : S \rightarrow T$ .*

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

Hence the recipe consists in the following two steps:

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

In the next section, we illustrate this principle.

## 4.5.2 Translation of lambda calculus with fixpoint to lambda calculus

In this section, we consider the 2-signature  $\Upsilon_{LC_{\beta\eta, \text{fix}}} := \Upsilon_{LC_{\beta\eta}} + \Upsilon_{\text{fix}}$  where the two components have been introduced above (see Example 90 and Section 4.4.4).

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

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

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

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

PART II

# Semantics

---

# REDUCTION MONADS AND THEIR SIGNATURES

---

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*. They are, in particular, signatures in the sense of Chapter 2 consisting of a signature over the category of monads (for example, a 2-signature, as introduced in Chapter 4) extended with a family of arities specifying reduction rules.

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, there is 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.

## 5.1 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  $\beta$ -reduction, while the second addresses substitution but incorporates only the  $\beta$ -equality. Merging these two perspectives led L  th 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.

### 5.1.1 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 4. Sometimes they also refer to 1-signatures of Chapter 3, as particular cases of 2-signatures without equations. The letter  $\Sigma$  is usually associated with a signature for monads. Our work generalizes straightforwardly to the case of signatures over the category of monads, in the sense of Chapter 2.

**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  $\Sigma$  for monads as  $\Sigma$ -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  $t\{y := u\}$ .

### 5.1.2 Plan of the chapter

In Section 5.2, we define reduction monads. In Section 5.3, we present our take on reduction rules. This enables us to define signatures for reduction monads—*reduction signatures*—in Section 5.4. Section 5.5 is devoted to the proof of our theorem of effectivity for these signatures. Then, in Section 5.6, we give a detailed example of a reduction signature specifying the lambda calculus with explicit substitutions of [Kes09]. Finally, in Section 5.7, we explain the recursion principle which, as usual, can be derived from initiality in our categories of models.

## 5.2 Reduction monads

Here below, we define *the category of reduction monads* in Section 5.2.1. We also consider some examples of reduction monads, in Section 5.2.2.

### 5.2.1 The category of reduction monads

**Definition 114.** 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  $\text{Red}(R)$  (the module of *reductions*);
3. a morphism of  $R$ -modules  $\text{red}_R : \text{Red}(R) \rightarrow R \times R$  (*source* and *target* of rules).

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

**Notation 115.** For a reduction monad  $R$ , a set  $X$ , and elements  $s, t \in R(X)$ , we think of the fiber  $\text{red}_R(X)^{-1}(s, t)$  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  $m \in \text{red}_R(X)^{-1}(s, t)$ .

**Remark 116.** 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 117.** Let  $R$  be a reduction monad,  $X$  and  $Y$  two sets,  $f : X \rightarrow R(Y)$  a substitution, and  $u$  and  $v$  two elements of  $R(X)$  related by  $m : u \blacktriangleright v$ . The module structure on  $\text{Red}(R)$  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 5.2.2.

**Definition 118.** A **morphism of reduction monads** from  $R$  to  $S$  is given by a pair  $(f, \alpha)$  of

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

satisfying the following two conditions:

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

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

commutes in the category of natural transformations.

In Section 5.7 we specify morphisms of reduction monads via a recursion principle.

Intuitively, a morphism  $(f, \alpha)$  as above maps terms of  $R$  to terms of  $S$  via  $f$ , and reductions of  $R$  to reductions of  $S$  via  $\alpha$ . 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  $\alpha$  to a reduction  $\alpha(m) : f(u) \blacktriangleright f(v)$ .

**Proposition 119** (Category of reduction monads). *Reduction monads and their morphisms, with the obvious composition and identity, form a category  $\text{RedMon}$ , 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 120.** *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. □

## 5.2.2 Examples of reduction monads

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

**Example 121** (Lambda calculus with head- $\beta$ -reduction). Consider the reduction monad  $\text{LC}_{\text{head-}\beta}$  given as follows:

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

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

**Example 122** (Lambda calculus with head- $\eta$ -expansion). Consider the reduction monad  $\text{LC}_{\text{head-}\eta}$  given as follows:

1. the underlying monad is  $\text{LC}$ ;
2.  $\text{Red}(R)$  is the module  $\text{LC}$ ;



3.  $\text{red}_R(X)$  is the morphism  $t \mapsto (t, \text{abs}(\text{app}(\iota t, *)))$ , where  $\iota : \text{LC}(X) \rightarrow \text{LC}'(X)$  is the natural injection (weakening).

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

Obviously, to get the analogous reduction rule for the  $\eta$ -contraction, it is enough to swap, in the previous example, the components of the output of the morphism  $\text{red}_R$ .

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

**Definition 123.** 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  $\text{Red}(R \amalg S)$  is the coproduct  $\text{Red}(R) \amalg \text{Red}(S)$ .
3. the module morphism  $\text{red}_{R \amalg S}$  is induced by  $\text{red}_R$  and  $\text{red}_S$ .

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

**Example 124.** The reduction monad  $\text{LC}_{\text{head-}\beta/\eta} := \text{LC}_{\text{head-}\beta} \amalg \text{LC}_{\text{head-}\eta}$  has, as reductions,  $\beta$ -reductions and  $\eta$ -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 125.** Let  $R$  be a reduction monad over the monad  $\text{LC}$  on sets. We define the reduction monad  $R^{\text{cong}}$  as follows:

1. the underlying monad is  $\text{LC}$ ;
2.  $\text{Red}(R^{\text{cong}})(X)$  is generated by the following constructions:
  - (a) for  $m : u \blacktriangleright v$  in  $\text{Red}(R)$ ,  $m$  is also in  $\text{Red}(R^{\text{cong}})$
  - (b) for  $m : u \blacktriangleright v : \text{LC}(X)$  in  $\text{Red}(R^{\text{cong}})$  and  $t \in \text{LC}(X)$ , we have  $\text{app-cong}_1(m, t) : \text{app}(u, t) \blacktriangleright \text{app}(v, t)$
  - (c) for  $m : u \blacktriangleright v : \text{LC}(X)$  in  $\text{Red}(R^{\text{cong}})$  and  $t \in \text{LC}(X)$ , we have  $\text{app-cong}_2(t, m) : \text{app}(t, u) \blacktriangleright \text{app}(t, v)$

(d) for  $m : u \blacktriangleright v : LC'(X)$  in  $\text{Red}(R^{\text{cong}})$  we have  $\text{abs-cong}(m) : \text{abs}(u) \blacktriangleright \text{abs}(v)$

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

**Example 126.** A reduction in the reduction monad  $LC_{\beta/\eta} := (LC_{\text{head-}\beta/\eta})^{\text{cong}}$  is “one step” of  $\beta$ -reduction or  $\eta$ -expansion, anywhere in the source term.

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

**Definition 127.** 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  $\text{Red}(R^*)$  is defined as follows. For  $n \in \mathbb{N}$  we define the module  $\text{Red}(R)^n$  of “ $n$  composable reductions”, namely as the limit of the diagram

$$\begin{array}{ccccccc}
 & & \text{Red}(R) & & \text{Red}(R) & & \dots \\
 & \swarrow \text{source}_R & & \searrow \text{target}_R & \swarrow \text{source}_R & \searrow \text{target}_R & \\
 R & & & R & & R & 
 \end{array}$$

with  $n$  copies of  $\text{Red}(R)$  (and hence  $n + 1$  copies of  $R$ ). We obtain  $n + 1$  projections  $\pi_i : \text{Red}(R)^n \rightarrow R$ , and we call  $p_n := (\pi_0, \pi_n) : \text{Red}(R)^n \rightarrow R \times R$ . We set  $\text{Red}(R^*) := \coprod_n \text{Red}(R)^n$ .

3. the module morphism is  $\text{red}_{R^*} := [p_n]_{n \in \mathbb{N}} : \coprod_n \text{Red}(R)^n \rightarrow R \times R$  the universal morphism induced by the family  $(p_n)_{n \in \mathbb{N}}$ .

**Example 128** (The reduction monad of the lambda calculus). The *reduction monad of the lambda calculus* is defined to be the reduction monad  $LC_{\beta/\eta}^*$ .

In Section 5.4 we introduce signatures that allow for the specification of reduction monads. The signature specifying the reduction monad of lambda calculus of Example 126 is given in Example 157.

## 5.3 Reduction rules

In this section, we define an abstract notion of reduction rule over a signature for monads  $\Sigma$  (Section 5.3.2). We first focus, in Section 5.3.1, on the example of the congruence rule for the application construction in the signature  $\Sigma_{LC}$  for the monad of the

lambda calculus, in order to motivate the definitions. The destiny of a reduction rule over  $\Sigma$  is to be “validated” in a reduction monad equipped with an action of  $\Sigma$  (this is what we will call a reduction  $\Sigma$ -monad in Section 5.3.3). We make this notion of validation precise in Section 5.3.4, as an action of the reduction rule in the reduction  $\Sigma$ -monad. Finally, we give a protocol for specifying reduction rules in Section 5.3.5 that we apply in Section 5.3.6 to some examples.

### 5.3.1 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:

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

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  $\text{app} : R \times R \rightarrow R$ , 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  $c_R : \mathcal{V}(R) \rightarrow R$  maps a set  $X$  and a quadruple  $(T, T', U, U')$  to the pair  $(\text{app}(T, U), \text{app}(T', U'))$ . These assignments are actually functorial in  $R$ , and abstracting over  $R$  yields our notion of *term-pair* over the  $\Sigma$ -module  $\mathcal{V}$ , as morphisms from  $\mathcal{V}$  to  $\Theta \times \Theta$ , where  $\Sigma$  is any signature including a single first-order binary operation  $\text{app}$  (for example,  $\Sigma_{\text{LC}}$ ). 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 quadruple  $(T, T', U, U') \in \mathcal{V}(R)$ , any reductions  $s : T \blacktriangleright T'$  and  $t : U \blacktriangleright U'$ , there is a reduction  $\text{app-cong}(s, t) : \text{app}(T, U) \blacktriangleright \text{app}(T', U')$ . 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  $\Sigma_{\text{app}}$ . We call such a structure a *reduction  $\Sigma_{\text{app}}$ -monad*.

### 5.3.2 Definition of reduction rules

In this subsection,  $\Sigma$  is a signature for monads. We present our notion of *reduction rule over  $\Sigma$* , from which we build *reduction signatures* in Section 5.4.

We begin with the definition of *term-pair*, alluded to already in Section 5.3.1:

**Definition 129.** Given a  $\Sigma$ -module  $\mathcal{V}$ , a **term-pair from  $\mathcal{V}$**  is a pair  $(n, p)$  of a natural number  $n$  and a morphism of  $\Sigma$ -modules  $p : \mathcal{V} \rightarrow \Theta^{(n)} \times \Theta^{(n)}$ .

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 130.** Let  $n_1, \dots, n_p$  be a list of natural numbers. For  $i, j \in \{1, \dots, p\}$ , we define the pair projection  $\pi_{i,j}$  and the projection  $\pi_i$  as the following  $\Sigma$ -module morphisms, for any signature  $\Sigma$ :

$$\begin{aligned} \pi_{i,j} : \Theta^{(n_1)} \times \dots \times \Theta^{(n_p)} &\rightarrow \Theta^{(n_i)} \times \Theta^{(n_j)} & \pi_{i,j,R,X}(T_1, \dots, T_p) &= (T_i, T_j) \\ \pi_i : \Theta^{(n_1)} \times \dots \times \Theta^{(n_p)} &\rightarrow \Theta^{(n_i)} & \pi_{i,j,R,X}(T_1, \dots, T_p) &= T_i \end{aligned}$$

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 131** (term-pair of the conclusion of the congruence for application). The term-pair corresponding to the conclusion  $\text{app}(T, U) \rightsquigarrow \text{app}(T', U')$  of congruence for application (Section 5.3.1) is given by  $(0, c)$ , on the  $\Sigma_{\text{LC}}$ -module  $\Theta^4$ . Here, we have

$$\begin{aligned} c : \mathcal{V} &\rightarrow \Theta \times \Theta \\ c_{R,X}(T, T', U, U') &:= \left( \text{app}_{R,X}(T, U), \text{app}_{R,X}(T', U') \right) \end{aligned}$$

More schematically:

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

We now give our definition of *reduction rule*, making precise the intuition developed in Section 5.3.1.

**Definition 132.** A **reduction rule**  $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c)) **over  $\Sigma$**  is given by:$

- **Metavariables:** a  $\Sigma$ -module  $\mathcal{V}$  of metavariables, that we sometimes denote by  $\text{MVar}_{\mathcal{A}}$ ;
- **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 133** (Reduction rule for congruence of application). The reduction rule  $\mathcal{A}_{\text{app-cong}}$  for congruence of application (Section 5.3.1) 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  $(0, h_2)$ :

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

- **Conclusion:** Given by the term-pair  $(0, c)$  of Example 131.

More examples of reduction rules are given in Section 5.3.6.

### 5.3.3 Reduction $\Sigma$ -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  $\Sigma$  for monads, this reduction monad  $R$  must be equipped with an action of  $\Sigma$ , hence the following definition:

**Definition 134.** Let  $\Sigma$  be a signature for monads. The **category**  $\text{RedMon}^{\Sigma}$  **of reduction  $\Sigma$ -monads** is defined as the following pullback:

$$\begin{array}{ccc} \text{RedMon}^{\Sigma} & \xrightarrow{\quad} & \text{RedMon} \\ \downarrow & \lrcorner & \downarrow \\ \text{Mon}^{\Sigma} & \xrightarrow{\quad} & \text{Mon} \end{array}$$

More concretely,

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

- a **morphism of reduction  $\Sigma$ -monads**  $R \rightarrow S$  is a morphism  $f : R \rightarrow S$  of reduction monads compatible with the action of  $\Sigma$ , i.e, whose underlying monad morphism is a  $\Sigma$ -monad morphism.

**Remark 135.** The pullback of a signature  $\Sigma$  for monads along the forgetful functor  $U : \text{RedMon} \rightarrow \text{Mon}$  defines a signature  $U^*\Sigma$  over the category of reduction monads (see Chapter 2). Then,  $\text{RedMon}^\Sigma$  is retrieved as the category of models of  $U^*\Sigma$ .

### 5.3.4 Action of a reduction rule

Let  $\Sigma$  be a signature for monads. In this section, we introduce the notion of *action of a reduction rule over  $\Sigma$  in a reduction  $\Sigma$ -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 136.** Let  $(n, p)$  be a term-pair from a  $\Sigma$ -module  $\mathcal{V}$ , and  $R$  be a reduction  $\Sigma$ -monad. We denote by  $p^*(\text{Red}(R)^{(n)})$  the pullback of  $\text{red}_R^{(n)} : \text{Red}(R)^{(n)} \rightarrow R^{(n)} \times R^{(n)}$  along  $p_R : \mathcal{V}(R) \rightarrow R^{(n)} \times R^{(n)}$ :

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

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

**Definition 137.** Let  $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$  be a reduction rule, and  $R$  be a reduction  $\Sigma$ -monad. The  $R$ -**module**  $\text{Hyp}_{\mathcal{A}}(R)$  **of hypotheses of  $\mathcal{A}$**  is  $\prod_{i \in I / \mathcal{V}(R)} h_i^* \text{Red}(R)^{(n_i)}$  the fiber product of all the  $R$ -modules  $h_i^* \text{Red}(R)^{(n_i)}$  along their projection to  $\mathcal{V}(R)$ . It thus comes with a projection  $\text{hyp}_{\mathcal{A}}(R) : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \mathcal{V}(R)$

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

**Example 138.** Let  $R$  be a reduction  $\Sigma_{\text{LC}}$ -monad. The  $R$ -module of conclusion of the congruence reduction rule for application (Example 133) 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  $m : \text{app}(T, U) \blacktriangleright \text{app}(T', U')$ .

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

$$\tau : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \text{Con}_{\mathcal{A}}(R)$$

making the following diagram commute:

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

**Example 140** (Action of the congruence rule for application). Consider the reduction rule of the congruence for application of Example 133. Let  $R$  be a reduction  $\Sigma_{\text{LC}}$ -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  $m : \text{app}(A, B) \blacktriangleright \text{app}(A', B')$ . The commutation of the triangle (5.1) ensures that  $(A, A', B, B') = (T, T', U, U')$ .

Alternatively (as justified formally by Lemma 161), an action is a morphism mapping the same sextuple to a reduction  $m : \text{app}(T, U) \blacktriangleright \text{app}(T', U')$ .

**Remark 141.** Any reduction rule  $\mathcal{A}$  over  $\Sigma$  induces an arity  $(D, a, u, v)$  over  $\text{RedMon}^{\Sigma}$  defined as follows:

- $a : D \rightarrow \text{RedMon}^{\Sigma}$  is the Grothendieck fibration corresponding (through the Grothendieck construction) to the functor mapping a reduction  $\Sigma$ -monad  $R$  to the category  $\text{Mod}(R)/\text{MVar}_{\mathcal{A}}(R)$ ;
- $u$  maps a reduction  $\Sigma$ -monad  $R$  to  $\text{hyp}_{\mathcal{A}}(R) : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \text{MVar}_{\mathcal{A}}(R)$
- $v$  maps a reduction  $\Sigma$ -monad  $R$  to  $\text{con}_{\mathcal{A}}(R) : \text{Con}_{\mathcal{A}}(R) \rightarrow \text{MVar}_{\mathcal{A}}(R)$

Then, the notion of action of this induced arity (see Chapter 2) coincides with the one introduced in this section.

### 5.3.5 Protocol for specifying reduction rules

In Section 5.3.6, we adopt the following schematic presentation of a reduction rule over a signature  $\Sigma$ :

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

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

$$\begin{aligned} p_i &: M_1 \times \dots \times M_q \rightarrow \Theta^{(m_i)} \times \Theta^{(m_i)} \\ p_{i,R,X}(T_1, \dots, T_q) &:= (s_i(T_1, \dots, T_q), t_i(T_1, \dots, T_q)) \end{aligned} \quad (5.2)$$

where the  $\Sigma$ -modules  $M_1, \dots, M_q$ , and the natural numbers  $m_0, \dots, m_n$  are inferred for Equation (5.2) to be well defined for all  $i \in \{0, \dots, n\}$ .

The induced reduction rule is:

- **Metavariables:** the  $\Sigma$ -module of metavariables is  $\mathcal{V} = M_1 \times \dots \times M_q$ ;
- **Hypotheses:** the hypotheses are the term-pairs  $(m_i, p_i)_{i \in \{1, \dots, n\}}$ ;
- **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.

### 5.3.6 Examples of reduction rules

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

For the rest of this section, we assume that we have fixed a signature for monads  $\Sigma$ . Figure 5.1 shows some notable examples of reduction rules. In order, they are: reflexivity, transitivity, congruence for abs,  $\beta$ -reduction,  $\eta$ -expansion, and expansion of the fixpoint operator.



$$\begin{array}{c}
 \frac{}{T \rightsquigarrow T} \text{Refl} \quad \frac{T \rightsquigarrow U \quad U \rightsquigarrow W}{T \rightsquigarrow W} \text{Trans} \quad \frac{T \rightsquigarrow U}{\text{abs}(T) \rightsquigarrow \text{abs}(U)} \text{abs-Cong} \\
 \\
 \frac{}{\text{app}(\text{abs}(T), U) \rightsquigarrow T\{* := U\}} \beta\text{-Red} \quad \frac{}{T \rightsquigarrow \text{abs}(\iota(T))} \eta\text{-Exp} \quad \frac{}{\text{fix}(T) \rightsquigarrow T\{* := \text{fix}(T)\}} \text{fix-Exp}
 \end{array}$$

Figure 5.1: Examples of reduction rules.

Rule	Signature	Metavariables	Hypothesis	Conclusion
Refl	any	$\Theta$		$(0, (\text{id}, \text{id}))$
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	$\Sigma_{\text{LC}}$	$\Theta' \times \Theta'$ for $(T, U)$	$(1, \text{id})$	$(0, \text{abs} \times \text{abs})$
$\beta$ -Red	$\Sigma_{\text{LC}}$	$\Theta' \times \Theta$ for $(T, U)$		$(0, c_{\beta\text{-Red}})$
$\eta$ -Exp	$\Sigma_{\text{LC}}$	$\Theta$		$(0, (\text{id}, \text{abs} \circ \iota))$
fix-Exp	$\Sigma_{\text{fix}}$	$\Theta'$		$(0, c_{\text{fix-Exp}})$

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

Figure 5.2: Modules and term pairs relative to the reduction rules of Figure 5.1.

For the example of the fixpoint operator (rule fix-Exp), we consider the 1-signature  $\Sigma_{\text{fix}}$ , as described in Section 4.4.4. (without enforcing the fixpoint equation, which is replaced here by the reduction rule under consideration). A  $\Sigma_{\text{fix}}$ -monad is a monad  $R$  equipped with an  $R$ -module morphism  $\text{fix} : R' \rightarrow R$ .

Figure 5.2 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 130.

## 5.4 Signatures for reduction monads and Initiality

In this section, we define the notion of *reduction signature*, consisting of a signature for monads  $\Sigma$  and a family of reduction rules over  $\Sigma$  (see Section 5.4.1). A reduction signature induces a signature over the category of reduction monads, in the sense of Chapter 2. Thus, for each of them there is an associated category of models. Our main result, Theorem 155 (see Section 5.4.3), states that a reduction signature is effective as soon as its underlying signature for monads is effective.

### 5.4.1 Signatures and their models

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

**Definition 142.** A **reduction signature** is a pair  $(\Sigma, \mathfrak{R})$  of a signature  $\Sigma$  for monads and a family  $\mathfrak{R}$  of reduction rules over  $\Sigma$ .

**Remark 143.** A reduction signature  $(\Sigma, \mathfrak{R})$  defines a signature  $S_{\Sigma, \mathfrak{R}} = S, \mathfrak{R}'$  over the category of reduction monads, in the sense of Chapter 2, as follows:

- $S$  is the pullback of  $S_{\Sigma}$  along the forgetful functor  $U : \text{RedMon} \rightarrow \text{Mon}$ , with  $\text{RedMon}^{\Sigma}$  as associated category of models (Remark 135);
- $\mathfrak{R}'$  is the family of arities induced by the reduction rules of  $\mathfrak{R}$ , as explained in Remark 141.

**Definition 144.** Given a reduction monad  $R$  and a reduction signature  $S = (\Sigma, \mathfrak{R})$ , an **action of  $S$  in  $R$**  consists of an action of  $\Sigma$  in its underlying monad  $\underline{R}$  and an action of each reduction rule of  $\mathfrak{R}$  in  $R$ .

**Definition 145.** Let  $S = (\Sigma, \mathfrak{R})$  be a reduction signature. A **model of  $S$**  is a reduction monad equipped with an action of  $S$ , or equivalently, a reduction  $\Sigma$ -monad equipped with an action of each reduction rule of  $\mathfrak{R}$ .

**Remark 146** (Continuation of Remark 116). Just as our reduction monads are “proof-relevant” (cf. Remark 116), our notion of reduction signature allows for the specification of multiple reductions between terms. As a trivial example, duplicating the  $\beta$ -rule in the signature  $S_{\text{LC}}$  yields two distinct  $\beta$ -reductions in the initial model.

**Remark 147.** A model of a reduction signature  $\Sigma, \mathfrak{R}$  is the same thing as a model of  $S_{\Sigma, \mathfrak{R}}$  (see Remark 143).

### 5.4.2 The functors $\text{Hyp}_{\mathcal{A}}$ and $\text{Con}_{\mathcal{A}}$

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

**Definition 148.** Let  $\Sigma$  be a signature for monads, and  $\mathcal{A}$  be a reduction rule over  $\Sigma$ . Definition 137 assigns to each  $\Sigma$ -monad  $R$  the  $R$ -modules  $\text{Hyp}_{\mathcal{A}}(R)$  and  $\text{Con}_{\mathcal{A}}(R)$ . These assignments extend to functors  $\text{Hyp}_{\mathcal{A}}, \text{Con}_{\mathcal{A}} : \text{RedMon}^{\Sigma} \rightarrow \int \text{Mod}$ .

**Proposition 149.** *Given the same data, the functors  $\text{Hyp}_{\mathcal{A}}$  and  $\text{Con}_{\mathcal{A}}$  commute with the forgetful functors to  $\text{Mon}$ :*

$$\begin{array}{ccc} \text{RedMon}^{\Sigma} & \begin{array}{c} \xrightarrow{\text{Hyp}_{\mathcal{A}}} \\ \xrightarrow{\text{Con}_{\mathcal{A}}} \end{array} & \int \text{Mod} \\ & \searrow \quad \swarrow & \\ & \text{Mon} & \end{array}$$

### 5.4.3 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 155, which gives a sufficient condition for  $\mathcal{S}$  to admit an initial model.

**Definition 150.** 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  $\Sigma$ -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:

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

**Example 151** (Example 140 continued). Consider the reduction signature consisting of the signature  $\Sigma_{\text{app}}$  of a binary operation  $\text{app}$  and a single reduction rule of congruence for application (Example 133).

Let  $R$  and  $T$  be models for this signature: they are reduction  $\Sigma_{\text{app}}$ -monads equipped with an action  $\rho$  and  $\tau$ , in the alternative sense of Example 140. A  $\Sigma_{\text{app}}$ -monad morphism  $(f, \alpha)$  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  $\rho(A, A', B, B') : \text{app}(A, B) \blacktriangleright \text{app}(A', B')$  is mapped to the reduction  $\tau(f(A), f(A'), f(B), f(B'))$  by  $\alpha : \text{Red}(R) \rightarrow \text{Red}(T)$ .

**Proposition 152.** *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  $\text{RedMon}^{\mathcal{S}}$ , equipped with a forgetful functor to  $\text{RedMon}^{\Sigma}$ .*

**Remark 153.** The category of models of a reduction signature  $(\Sigma, \mathfrak{R})$  is isomorphic to the category of models of  $\mathcal{S}_{\Sigma, \mathfrak{R}}$  (see Remark 143).

**Definition 154.** A reduction signature  $\mathcal{S}$  is said to be **effective** if its category of models  $\text{RedMon}^{\mathcal{S}}$  has an initial object, denoted  $\hat{\mathcal{S}}$ . In this case, we say that  $\hat{\mathcal{S}}$  is **generated (or specified) by  $\mathcal{S}$** .

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

**Theorem 155.** *Let  $(\Sigma, \mathfrak{R})$  be a reduction signature. If  $\Sigma$  is effective, then so is  $(\Sigma, \mathfrak{R})$ .*

The proof of this theorem is given in Section 5.5.

Theorem 104 entails the following corollary:

**Corollary 156.** *Let  $(\Sigma, \mathfrak{R})$  be a reduction signature. If  $\Sigma$  is algebraic (in the sense of Definition 103), then  $(\Sigma, \mathfrak{R})$  is effective.*

All the examples of reduction signatures considered here satisfy the algebraicity condition of Corollary 156.

**Example 157** (Reduction signature for Example 126). The reduction monad of Example 126 is generated by the reduction signature  $\mathcal{S}_{\text{LC}}$  that is given by the signature  $\Sigma_{\text{LC}}$  together with the following reduction rules (see Section 5.3.6):

- the reduction rule for  $\beta$ -reduction;
- the reduction rule for  $\eta$ -expansion;
- the congruence rule for abstraction;
- two unary congruence rules for application:

$$\frac{T \rightsquigarrow T'}{\text{app}(T, U) \rightsquigarrow \text{app}(T', U)} \quad \frac{U \rightsquigarrow U'}{\text{app}(T, U) \rightsquigarrow \text{app}(T, U')}.$$

**Example 158** (Reduction signature of lambda calculus with a fixpoint operator). The signature  $\mathcal{S}_{\text{LC}_{\text{fix}}}$  specifying the reduction monad  $\text{LC}_{\text{fix}}$  of the lambda calculus with a fixpoint operator extends the signature  $\mathcal{S}_{\text{LC}}$  of Example 157 with:

- a new operation  $\text{fix} : \Theta' \rightarrow \Theta$  (thus extending the signature for monads  $\Sigma_{\text{LC}}$ );
- the reduction rule for the fixpoint reduction (cf. Section 5.3.6);
- a congruence rule for  $\text{fix}$ :

$$\frac{T \rightsquigarrow T'}{\text{fix}(T) \rightsquigarrow \text{fix}(T')}.$$

## 5.5 Proof of Theorem 155

This section details the proof of Theorem 155.

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  $\Sigma$ -monads to the category of  $\Sigma$ -monads.

In Section 5.5.1, 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 5.5.2, we give an alternative definition of the category of models that we make use of in the proof of effectivity, in Section 5.5.3.

### 5.5.1 Normalizing reduction rules

**Definition 159.** A reduction rule  $(\mathcal{V}, (n_j, h_j)_{j \in J}, (n, c))$  is said **normalized** if  $n = 0$ .

**Lemma 160.** *Let  $\mathcal{A} = (\mathcal{V}, (n_j, h_j)_{j \in J}, (n, c))$  be a reduction rule over  $\Sigma$ . Then there exists a normalized reduction rule  $\mathcal{A}'$  over  $\Sigma$  such that the induced notion of action is equivalent, in the sense that:*

- *given a reduction  $\Sigma$ -monad  $R$ , there is a bijection between actions of  $\mathcal{A}$  in  $R$  and actions of  $\mathcal{A}'$  in  $R$ ;*
- *a morphism between reduction  $\Sigma$ -monads equipped with an action of  $\mathcal{A}$  preserves the action (in the sense of Definition 150) 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 161.** Let  $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$  be a reduction rule over  $\Sigma$ . By universal property of the pullback  $\text{Con}_{\mathcal{A}}(R) = c^* \text{Red}(R)^{(n)}$ , an action can be alternatively be defined as an  $R$ -module morphism  $\sigma : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \text{Red}(R)^{(n)}$  making the following diagram commute

$$\begin{array}{ccc} \text{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\sigma} & \text{Red}(R)^{(n)} \\ \downarrow & & \downarrow \text{red}_R^{(n)} \\ \mathcal{V}(R) & \xrightarrow{c} & R^{(n)} \times R^{(n)} \end{array} . \quad (5.3)$$

**Lemma 162.** 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  $\Sigma$ -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  $\mathfrak{R}$ :

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

We now prove Lemma 160 using these alternative definitions:

*Proof of Lemma 160.* 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  $j \in J$ ,  $h'_j : \mathcal{V}' \rightarrow \Theta^{(n_j)} \times \Theta^{(n_j)}$  is defined as the composition of  $\pi_1 : \mathcal{V} \times \Theta^n \rightarrow \mathcal{V}$  with  $h_j : \mathcal{V} \rightarrow \Theta^{(n_j)} \times \Theta^{(n_j)}$ .
- Conclusion: The morphism  $c' : \mathcal{V} \times \Theta^n \rightarrow \Theta \times \Theta$  is the  $n^{\text{th}}$  uncurrying (see Definition 81) of  $c : \mathcal{V} \rightarrow \Theta^{(n)} \times \Theta^{(n)}$ .

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

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

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

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

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.  $\square$

**Corollary 163.** *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 160.  $\square$

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

### 5.5.2 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  $\Sigma$ -monad  $R$  is given by a  $R$ -module morphism  $\tau : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \text{Red}(R)$  such that the following square commutes:

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

We can rephrase this commutation by stating that this morphism  $\tau$  is a morphism in the slice category  $\text{Mod}(\underline{R})/\underline{R}^2$  from an object that we denote by  $F_{\mathcal{A}|\underline{R}}(\text{Red}(R), \text{red}_R)$ , to  $(\text{Red}(R), \text{red}_R)$ . Here, we use the notation  $\underline{R}$  to refer explicitly to the underlying monad

of  $R$ . Actually, the domain is functorial in its argument, and thus the action  $\tau$  can be thought of as an algebra structure on  $(\text{Red}(R), \text{red}_R)$ :

**Lemma 164.** *Given any  $\Sigma$ -monad  $R$ , the assignment  $(M, p : M \rightarrow R \times R) \mapsto F_{\mathcal{A}|R}(M, c)$  yields an endofunctor  $F_{\mathcal{A}|R}$  on  $\text{Mod}(R)/R^2$ . An action of  $\mathcal{A}$  in a reduction  $\Sigma$ -monad  $R$  is exactly the same as an algebra structure for this endofunctor on  $(\text{Red}(R), \text{red}_R) \in \text{Mod}(R)/R^2$ .*

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

*Proof.* This is a consequence of the functoriality of  $\text{Hyp}_{\mathcal{A}}$ , as noticed in Section 5.4.2. □

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

**Proposition 165.** *Let  $F_S : \text{RedMon}^\Sigma \rightarrow \text{RedMon}^\Sigma$  be the coproduct  $\coprod_i F_{\mathcal{A}_i}$ . Then, the category of models of  $S$  is isomorphic to the **category of vertical algebras** of  $F_S$  defined as follows:*

- *an object is an algebra  $r : F_S(R) \rightarrow R$  such that  $r$  is mapped to the identity by  $\mathcal{U}^\Sigma$*
- *morphisms are the usual  $F_S$ -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 166.** *The forgetful functor from the category of models of  $S$  to the category of  $\Sigma$ -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  $\mathcal{U}^\Sigma : \text{RedMon}^\Sigma \rightarrow \text{Mon}^\Sigma$  is a fibration) and  $F = F_S$ :

**Lemma 167.** *Let  $p : E \rightarrow B$  be a fibration and  $F$  an endofunctor on  $E$  satisfying  $p \cdot F = p$ . Then the category of vertical algebras of  $F$  is fibered over  $B$ .*

*Proof.* Let  $r : F(R) \rightarrow R$  be an algebra over  $X \in B$ . Let  $a : Y \rightarrow X$  be a morphism in  $B$ . Let  $\bar{a} : a^*R \rightarrow R$  the associated cartesian morphism in  $E$ . We define the pullback of  $r$  along  $\bar{a}$  as follows: the base object is  $a^*R$ , and the algebra structure  $\rho : F(a^*R) \rightarrow R$



is given by the unique morphism which factors  $F(a^*R) \xrightarrow{F(\bar{a})} F(R) \xrightarrow{r} R$  through the cartesian morphism  $\bar{a} : a^*R \rightarrow R$ . Thus, the square

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

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

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

As  $\bar{a}$  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  $\bar{a}$ . The fact that  $v$  is an algebra morphism allows to conclude.  $\square$

We want to apply this lemma for proving Lemma 166. We thus need to show that  $\mathcal{U}^\Sigma : \text{RedMon}^\Sigma \rightarrow \text{Mon}^\Sigma$  is a fibration:

**Lemma 168.** *The forgetful functors  $\text{RedMon} \rightarrow \text{Mon}$  and  $\mathcal{U}^\Sigma : \text{RedMon}^\Sigma \rightarrow \text{Mon}^\Sigma$  are fibrations.*

*Proof.* We have the two following pullbacks:

$$\begin{array}{ccccc} \text{RedMon}^\Sigma & \longrightarrow & \text{RedMon} & \longrightarrow & V(f \text{ Mod}) \\ \mathcal{U}^\Sigma \downarrow & & \downarrow & & \downarrow \text{cod} \\ \text{Mon}^\Sigma & \longrightarrow & \text{Mon} & \xrightarrow{\Theta \times \Theta} & f \text{ Mod} \end{array}$$

where  $V(f \text{ Mod})$  is the full subcategory of arrows of  $f \text{ Mod}$  which are vertical (that is,

they are mapped to the identity monad morphism by the functor from  $\int \text{Mod}$  to  $\text{Mon}$ ), and  $\text{cod}$  maps such an arrow to its codomain. By Propositions 23 and 27, the category  $\int \text{Mod}$  has fibred finite limits **TODO maybe formulate this in Chapter 3 ?**, so that  $\text{cod}$  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  $\text{RedMon} \rightarrow \text{Mon}$  is a fibration, and then,  $\mathcal{U}^\Sigma : \text{RedMon}^\Sigma \rightarrow \text{Mon}^\Sigma$  also is.  $\square$

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

*Proof of Lemma 166.* Apply Lemma 167 with the fibration  $p = \mathcal{U}^\Sigma$  (Lemma 168) and  $F = F_{\mathcal{S}}$ .  $\square$

### 5.5.3 Effectivity

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

**Lemma 169.** *Let  $p : E \rightarrow B$  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  $\Sigma$ -monad  $R$ . This fiber category can be characterized as a category of algebras:

**Lemma 170.** *The fiber category over a given  $\Sigma$ -monad  $R$  through the fibration from models of  $\mathcal{S}$  (Lemma 166) is the category of algebras of the endofunctor  $F_{\mathcal{S}|R} = \coprod_i F_{\mathcal{A}_i|R}$  on the slice category  $\text{Mod}(R)/R^2$ .*

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 171** (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 172.** *The category  $\text{Mod}(R)/R^2$  is cocomplete for any monad  $R$ .*

*Proof.* The category of modules  $\text{Mod}(R)$  over a given monad  $R$  is cocomplete by Proposition 23, so any of its slice categories is, by the dual of [ML98, Exercise V.1.1], in particular  $\text{Mod}(R)/R^2$ .  $\square$

Let us show that the finitariness requirement of Lemma 171 is also satisfied for the case of a signature with a single reduction rule:

**Lemma 173.** *Let  $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (0, c))$  be a normalized reduction rule over  $\Sigma$ , and  $R$  be a  $\Sigma$ -monad. Then,  $F_{\mathcal{A}|R}$  is finitary.*

*Proof.* In this proof, we denote by  $F$  the endofunctor  $F_{\mathcal{A}|R}$  on  $\text{Mod}(R)/R^2$ , by  $\pi : D/d \rightarrow D$  the projection for a general slice category, and by  $\alpha : \pi \rightarrow d$  the natural transformation induced by the underlying morphism of a slice object:  $\alpha_p : \pi(p) \rightarrow d$ . Note that  $\pi$  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  $\pi$  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  $\pi \cdot F : \text{Mod}(R)/R^2 \rightarrow \text{Mod}(R)$  is finitary.

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

$$\begin{array}{ccccc}
 & & \mathcal{V}(R) & & \\
 & \swarrow h_{i,R} & & \searrow h_{i',R} & \\
 R^{(n_i)} \times R^{(n_i)} & & & & R^{(n_{i'})} \times R^{(n_{i'})} \rightarrow \dots \\
 \uparrow \alpha_q^{(n_i)} & & & & \uparrow \alpha_q^{(n_{i'})} \\
 \pi(q)^{(n_i)} & & & & \pi(q)^{(n_{i'})}
 \end{array}$$

Let  $J : \mathcal{C} \rightarrow \text{Mod}(R)/R^2$  be a filtered diagram. As  $\pi$  preserves colimits (since it

creates them),  $\pi(F(\text{colim } J))$  is the limit of the following diagram:

$$\begin{array}{ccc}
 & \mathcal{V}(R) & \\
 h_{i,R} \swarrow & & \searrow h_{i',R} \\
 R^{(n_i)} \times R^{(n_i)} & & R^{(n_{i'})} \times R^{(n_{i'})} \rightarrow \dots \\
 \uparrow \alpha_{\text{colim } J}^{(n_i)} & & \uparrow \alpha_{\text{colim } J}^{(n_{i'})} \\
 \text{colim } \pi(J)^{(n_i)} & & \text{colim } \pi(J)^{(n_{i'})}
 \end{array}$$

Now, as limits and colimits are computed pointwise in the category of modules, and as finite limits commute with filtered colimits in  $\text{Set}$  ([ML98, Section IX.2, Theorem 1]), we have that  $\pi(F(\text{colim } J))$ , 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  $\Sigma$ -monad  $R$ :

**Lemma 174.** *For any  $\Sigma$ -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 173), 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 155.* We assume that  $\Sigma$  is effective; let  $R$  be the initial  $\Sigma$ -monad. We want to show that  $\mathcal{S}$  has an initial model. We apply Lemma 169 with  $p$  the fibration from models to  $\Sigma$ -monads (Lemma 166): we are left with providing an initial object in the fiber category over  $R$ . By Lemma 170, this boils down to constructing an initial algebra for the endofunctor  $F_{\mathcal{S}|R}$  on the category  $\text{Mod}(R)/R^2$ . We apply Lemma 171:  $\text{Mod}(R)/R^2$  is indeed cocomplete by Lemma 172, and  $F_{\mathcal{S}|R}$  is finitary by Lemma 174). □

## 5.6 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 be-

cause it involves operations subject to some equations, and on top of this syntax with equations, a “graph of reductions”.

In Section 5.6.1, we present the underlying signature for monads, and in Section 5.6.2, we list the reduction rules of the signature.

### 5.6.1 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  $\text{esubst}_X : \text{LC}'(X) \times \text{LC}(X) \rightarrow \text{LC}(X)$ . It is thus specified by the signature  $\Theta' \times \Theta$ . An action of this signature in a monad  $R$  yields a map  $\text{esubst}_X : R(X + \{*\}) \times R(X) \rightarrow R(X)$  for each set  $X$ , where  $\text{esubst}_X(t, u)$  is meant to model the explicit substitution  $t[* / u]$ .

**Definition 175.** The signature  $\Upsilon_{\text{LC}_{\text{ex}}}$  for the monad of the lambda calculus with explicit substitutions without equations is the coproduct of  $\Theta' \times \Theta$  and  $\Sigma_{\text{LC}}$ .

#### Equation

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

$$t[x/u][y/v] = t[y/v][x/u] \quad \text{if } y \notin \text{fv}(u) \text{ and } x \notin \text{fv}(v) . \quad (5.4)$$

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

$$\begin{array}{ccc} \Theta'' \times \Theta \times \Theta & \xrightarrow{\Theta'' \times \iota \times \Theta} \Theta'' \times \Theta' \times \Theta & \xrightarrow{\text{esubst} \times \Theta} \Theta' \times \Theta \xrightarrow{\text{esubst}} \Theta \\ \Theta'' \times \Theta \times \Theta & \xrightarrow{\Theta'' \times \Theta \times \iota} \Theta'' \times \Theta \times \Theta' & \xrightarrow{\text{esubst}' \circ \pi_{1,3}, \pi_2} \Theta' \times \Theta \xrightarrow{\text{esubst}} \Theta \end{array} \quad (5.5)$$

Here,  $\iota$  denotes the inclusion  $\Theta \rightarrow \Theta'$ .

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

**Definition 176.** The signature  $\Sigma_{\text{LC}_{\text{ex}}}$  of the lambda calculus monad with explicit substitutions consists of  $\Upsilon_{\text{LC}_{\text{ex}}}$  and the single  $\Sigma_{\text{LC}_{\text{ex}}}$ -equation stating the equality between the two morphisms of Equation 5.5.

**Lemma 177.** *The signature  $\Sigma_{\text{LC}_{\text{ex}}}$  for monads is effective*

*Proof.* This is a direct consequence of Corollary 156. □

## 5.6.2 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  $\Sigma_{\text{LC}_{\text{ex}}}$  of Definition 176; 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]):

$$\begin{array}{c} \frac{T \rightsquigarrow T'}{\text{app}(T, U) \rightsquigarrow \text{app}(T', U)} \quad \frac{U \rightsquigarrow U'}{\text{app}(T, U) \rightsquigarrow \text{app}(T, U')} \quad \frac{T \rightsquigarrow T'}{\text{abs}(T) \rightsquigarrow \text{abs}(T')} \\[10pt] \frac{T \rightsquigarrow T'}{\text{esubst}(T, U) \rightsquigarrow \text{esubst}(T', U)} \quad \frac{U \rightsquigarrow U'}{\text{esubst}(T, U) \rightsquigarrow \text{esubst}(T, U')} \end{array}$$

They are translated into reduction rules through the protocol described in Section 5.3.5.

Figure 5.3 gives Kesner’s rules. Five out of six of Kesner’s rules translate straightforwardly, see Figure 5.4. Note how the explicit weakening  $\iota : \Theta \rightarrow \Theta'$  accounts for the side condition  $x \notin \text{fv}(t)$  of the Gc-rule in Figure 5.4.

The side condition  $y \in \text{fv}(u)$  of  $\llbracket \rrbracket$ -rule is more involved and necessitates the definition of the  $\Sigma_{\text{LC}_{\text{ex}}}$ -module  $\Theta_*$  such that  $\text{LC}_{\text{ex}*}$  is the submodule of  $\text{LC}_{\text{ex}}'$  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  $\Sigma_{\text{LC}_{\text{ex}}}$ -monad  $R$ :

$$\begin{array}{c}
 \frac{}{(\lambda x.t)u \rightsquigarrow t[x/u]} \beta\text{-red} \quad \frac{x \notin \text{fv}(t)}{t[x/u] \rightsquigarrow t} \text{Gc} \quad \frac{}{x[x/u] \rightsquigarrow u} \text{var}[] \\
 \\
 \frac{}{(tu)[x/v] \rightsquigarrow t[x/v]u[x/v]} \text{app}[] \quad \frac{}{(\lambda y.t)[x/v] \rightsquigarrow \lambda y.t[x/v]} \text{abs}[] \\
 \\
 \frac{y \in \text{fv}(u)}{t[x/u][y/v] \rightsquigarrow t[y/v][x/u[y/v]]} []
 \end{array}$$

Figure 5.3: Reduction rules of lambda calculus with explicit substitutions.

$$\begin{array}{c}
 \frac{}{\text{app}(\text{abs}(T), U) \rightsquigarrow \text{esubst}(T, U)} \beta\text{-red} \quad \frac{}{\text{esubst}(\iota(T), U) \rightsquigarrow T} \text{Gc} \\
 \\
 \frac{}{\text{esubst}(\text{app}(T, U), V) \rightsquigarrow \text{app}(\text{esubst}(T, V), \text{esubst}(U, V))} \text{app-esubst} \\
 \\
 \frac{}{\text{esubst}(*, T) \rightsquigarrow T} \text{var-esubst} \quad \frac{}{\text{esubst}(\text{abs}(T), V) \rightsquigarrow \text{abs}(\text{esubst}'(T, \iota(V)))} \text{abs-esubst} \\
 \\
 \frac{}{\text{esubst}(\text{esubst}(T, \kappa(U)), V) \rightsquigarrow \text{esubst}(\text{esubst}'(T, \iota(V)), \text{esubst}(\kappa(U), V))} \text{esubst-esubst}
 \end{array}$$

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

where  $\Theta_*$  is the 1-hole context  $\Sigma_{\text{LC}_{\text{ex}}}$ -submodule of  $\Theta'$  (Definition 178)

Figure 5.4: Reduction rules of Figure 5.3 reformulated in our setting.

- $\eta(*) \in R_*(X)$ , for any set  $X$ ;
- (application)
  - if  $t \in R(X)$  and  $u \in R_*(X)$ , then  $\text{app}(\iota(t), u) \in R_*(X)$
  - if  $t \in R_*(X)$  and  $u \in R(X)$ , then  $\text{app}(t, \iota(u)) \in R_*(X)$
  - if  $t \in R_*(X)$  and  $u \in R_*(X)$ , then  $\text{app}(t, u) \in R_*(X)$
- 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  $\Sigma_{\text{LC}_{\text{ex}}}$ -module  $\Theta_*$  equipped with a morphism  $\kappa : \Theta_* \rightarrow \Theta'$ .

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

- the terminal  $\Sigma_{\text{LC}_{\text{ex}}}$ -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  $M'$  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 171). Unfortunately, the resulting  $\Sigma_{\text{LC}_{\text{ex}}}$ -module does not yield the module that we are expecting in the case of the monad  $\text{LC}_{\text{ex}}$ : it does not satisfy Equation 5.5, and thus contains more terms than necessary. To obtain the desired  $\Sigma_{\text{LC}_{\text{ex}}}$ -module, we equip  $\Theta'$  with its canonical algebra structure, inducing a morphism from the initial algebra, and we define  $\Theta_*$  as the image of this morphism, thus equipped with an inclusion  $\kappa : \Theta_* \rightarrow \Theta'$ .



**Definition 178.** We define the  $\Sigma_{\text{LC}_{\text{ex}}}$ -**module of “one-hole contexts”** to be  $\Theta_*$ , equipped with an inclusion  $\kappa : \Theta_* \rightarrow \Theta'$ .

**Remark 179.** 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 180.** The reduction signature  $\mathcal{S}_{\text{LC}_{\text{ex}}}$  of the lambda calculus reduction monad with explicit substitutions consists of the signature  $\Sigma_{\text{LC}_{\text{ex}}}$  of Definition 176 and all the reduction rules specified in this section.

**Lemma 181.** *The reduction signature  $\mathcal{S}_{\text{LC}_{\text{ex}}}$  is effective.*

*Proof.* We apply Theorem 155—the underlying signature for monads is effective by Lemma 177.  $\square$

## 5.7 Recursion

In this section, we derive, for any effective reduction signature  $\mathcal{S}$ , a recursion principle from initiality. In Section 5.7.1, we state this recursion principle, then we give an example of application in Section 5.7.2, by translating lambda calculus with a fix-point operator to lambda calculus. In Section 5.7.2, 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 5.7.4.

### 5.7.1 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 182** (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  $\tau$  of  $\mathcal{S}$  in  $T$  induces a reduction monad morphism  $\hat{\tau} : R \rightarrow T$ .*

*Proof.* The action  $\tau$  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.  $\square$

In the next sections, we illustrate this principle.

## 5.7.2 Translation of lambda calculus with fixpoint to lambda calculus

In this section, we consider the signature  $\mathcal{S}_{\text{LC}_{\text{fix}}}$  of Example 158 for the lambda calculus with an explicit fixpoint operator.

We build, by recursion, a reduction monad morphism from the initial model  $\text{LC}_{\text{fix}}$  of this signature to  $\text{LC}_{\beta/\eta}^*$ , the “closure under identity and composition of reductions” (Definition 127) of the initial model  $\text{LC}_{\beta/\eta}$  of the signature  $\mathcal{S}_{\text{LC}}$  (Example 157).

As explained in Section 5.7.1, we need to define an action of  $\mathcal{S}_{\text{LC}_{\text{fix}}}$  in  $\text{LC}_{\beta/\eta}^*$ . Note that  $\mathcal{S}_{\text{LC}_{\text{fix}}}$  is an extension of  $\mathcal{S}_{\text{LC}}$  (Example 158). First, we focus on the core  $\mathcal{S}_{\text{LC}}$  part: we show that the reduction monad  $\text{LC}_{\beta/\eta}^*$  inherits the canonical action of  $\mathcal{S}_{\text{LC}_{\beta/\eta}}$  in  $\text{LC}_{\beta/\eta}$ .

**Lemma 183.** *There is an action of  $\mathcal{S}_{\text{LC}_{\beta/\eta}}$  in  $\text{LC}_{\beta/\eta}^*$ .*

*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  $\text{abs}(t_0)$  and  $\text{abs}(t_n)$ . For each  $i$ , we have a reduction between  $\text{abs}(t_{i-1})$  and  $\text{abs}(t_i)$ . By composing the corresponding sequence, we obtain the desired reduction.  $\square$

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

- an operation  $\text{fix} : \text{LC}_{\beta/\eta}^* \rightarrow \text{LC}_{\beta/\eta}^*$ : for this, we choose a fixpoint combinator  $Y$  and set  $\text{fix}_X(t) = \text{app}(Y, \text{abs}(t))$ , in accordance with Section 3.8.4;
- an action of the reduction rule

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

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

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

- an action of the congruence rule

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

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  $\text{fix}(t)$  with  $\text{app}(Y, \text{abs}(t))$ , and each fixpoint reduction with a composite of  $\beta$ -reductions.

### 5.7.3 Translation of lambda calculus with explicit substitutions into lambda calculus with $\beta$ -reduction

In this section, we consider the reduction signature  $\mathcal{S}_{\text{LC}_{\text{ex}}} = (\Sigma_{\text{LC}_{\text{ex}}}, \mathfrak{R}_{\text{LC}_{\text{ex}}})$  introduced in Definition 180. The underlying monad of the initial model  $\text{LC}_{\text{ex}}$  is the monad of lambda calculus with an application and abstraction operation, and an explicit substitution operator  $\text{LC}_{\text{ex}}' \times \text{LC}_{\text{ex}} \rightarrow \text{LC}_{\text{ex}}$  satisfying Equation 5.5, for  $R = \text{LC}_{\text{ex}}$ . The associated reduction monad has all the rules specified in Section 5.6.

We build, by recursion, a reduction monad morphism from the initial model  $\text{LC}_{\text{fix}}$  of this signature to  $\text{LC}_{\text{trans, refl}}$ , the initial model of the signature  $\mathcal{S}_{\text{LC}}$  (Example 157) extended with the reflexivity and reflexivity reduction rules (see Section 5.3.6):

$$\frac{}{T \rightsquigarrow T} \text{Refl} \quad \frac{T \rightsquigarrow U \quad U \rightsquigarrow W}{T \rightsquigarrow W} \text{Trans.}$$

The underlying monad is the lambda calculus monad  $\text{LC}$ .

As explained in Section 5.7.1, we need to define an action of  $\mathcal{S}_{\text{LC}_{\text{ex}}}$  in  $\text{LC}_{\text{trans,refl}}$ . For the action of the underlying 1-signature, we give the canonical application and abstraction operations of  $\text{LC}$ , as the initial  $\Sigma_{\text{LC}}$ -monad. The explicit substitution operation  $\text{LC}' \times \text{LC} \rightarrow \text{LC}$  is defined using the monadic substitution, mapping a pair  $(t, u) \in \text{LC}(X + \{*\}) \times \text{LC}(X)$  to the monadic substitution  $t\{* := u\}$ . One checks that this operation satisfies Equation 5.5, thanks to the usual monadic equations.

The action of the underlying signature for the congruence rules of application and abstraction is canonically given by the model structure of  $\text{LC}_{\text{trans,refl}}$ . The other actions are all given by reflexivity, except the  $\beta$ -rule which already has an action in  $\text{LC}_{\text{trans,refl}}$ .

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

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

The left one is obtained from the substitution of the module of reductions (see Remark 117). The action of the right one is constructed by induction from congruences of  $\text{LC}_{\text{trans,refl}}$ : it is a consequence of Lemma 186, 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  $\text{LC}_{\text{trans,refl}}$ . Finally, by the recursion principle, we get a reduction monad morphism from  $\text{LC}_{\text{ex}}$  to  $\text{LC}_{\text{trans,refl}}$ . 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 reflexivity except for the ones for the  $\beta$ -reduction and congruence for the substitution.

## 5.7.4 Induction principle

**TODO déplacer dans chapitre FSCD ?** Let  $\Sigma$  be an algebraic signature for monads. In this section, we state an induction principle based on initiality in the category of  $\Sigma$ -monads. This is a variant of the recursion principle of Section 4.5. Although a similar one can be stated for reduction monads, here we focus on the mere monadic case as we have a use case: congruence for substitution (Lemma 186), that is actually used in translating the reduction monad of lambda calculus with explicit substitutions into the reduction monad of the lambda calculus with  $\beta$ -reduction (Section 5.7.3).

Let  $\hat{\Sigma}$  be the initial model of  $\Sigma$  and  $P_X(t)$  be predicates on  $t \in \hat{\Sigma}(X)$  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  $\Sigma$ -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 184.** *Let  $R$  be a monad. For a given set  $X$  and element  $t \in R(X)$ , let  $P_X(t)$  be a predicate on  $t$ . Then, the mapping  $R_{|P} : X \mapsto \{t \in R(X) | P_X(t)\}$  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  $f : X \rightarrow R(Y)$ , 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  $u : X \rightarrow R_{|P}(Y)$  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  $R_{|P} \rightarrow R$  is a monad morphism.  $\square$

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

**Lemma 185** (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 184, 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  $\sigma : \Sigma(\widehat{\Sigma}) \rightarrow \widehat{\Sigma}$  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:*

- $p : \hat{\Sigma}_{|P} \rightarrow \hat{\Sigma}$  is the monadic inclusion;
- $\Sigma(\_) : \text{Mon} \rightarrow \int \text{Mod}$  denotes the underlying 1-signature functor of the signature  $\Sigma$ .

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

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

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

The equations specified by the signature  $\Sigma$  automatically hold for  $\hat{\Sigma}_{|P}$ , because they do for  $\hat{\Sigma}$ . This exploits the fact that  $\Sigma$  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 hypothesis.

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

We give now an example of use: giving  $\text{LC}_{\text{trans,refl}}$  (introduced in the previous section) an “action” of the following informal reduction rule (which is made precise in the upcoming Remark 188),

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

**Lemma 186.** Consider  $\text{LC}_{\text{trans,refl}}$ , the initial model of the signature  $\mathcal{S}_{\text{LC}}$  of lambda calculus extended with the reflexivity and transitivity reduction rules. Then, monadic substitution lifts congruence, in the following sense: for any sets  $X$  and  $Y$ , any element  $t$  in  $\text{LC}_{\text{trans,refl}}(X)$ , any families  $(u_x)_{x \in X}$  and  $(v_x)_{x \in X}$  of elements of  $\text{LC}_{\text{trans,refl}}(Y)$ , 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 185 taking  $P_X(t)$  the following property, for  $t \in \text{LC}_{\text{trans,refl}}(X)$ :

$$\begin{aligned} & \forall Y (r : X \rightarrow \text{Red}(\text{LC}_{\text{trans,refl}})(Y)), \\ & \exists m : t\{\text{source}_{\text{LC}_{\text{trans,refl}}} \circ \text{red}_{\text{LC}_{\text{trans,refl}}}(Y)\} \blacktriangleright t\{\text{target}_{\text{LC}_{\text{trans,refl}}} \circ \text{red}_{\text{LC}_{\text{trans,refl}}}(Y)\} . \end{aligned}$$

□

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

**Remark 187.** 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 117).

**Remark 188.** If, in the definition of reduction rules, we allow that the  $\Sigma$ -module of metavariables depends on the module of reductions, then this statement is equivalent to saying that  $\text{LC}_{\text{trans,refl}}$  is equipped with an action of the following reduction rule:

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

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

**Remark 189.** 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.





# OPERATIONAL MONADS AND THEIR SIGNATURES

---

In this chapter, we extend the notion of *reduction monads* of Chapter 5. Our goal is to deal with programming languages whose terms do not form a monad, but rather into a module over a monad. The idea is that reductions are compatible with the monadic substitution only. Consider the example of the call-by-value lambda calculus: 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 form 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  $T : \text{Set} \rightarrow \text{Set}$ . 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 of binary trees with leaves in its argument.

We also generalize to the case of *heterogeneous reductions*. This allows us to cover the call-by-value lambda calculus with big-step operational semantics, where terms reduce to values.

We adopt the same terminology and notation as in Chapter 5. We are quicker here, as all the definitions and proofs are straightforward generalizations.

## Plan of the chapter

In Section 6.1, we define operational monads, a generalization of the reduction monads of Chapter 5. In Section 6.2, we adapt the definition of reduction rules to the new setting. This enables us to define signatures for operational monads—*operational signatures*—in Section 6.3. Then, in Section 6.4, we give some examples of operational

signatures.

## 6.1 Operational monads

We define *operational monads* in Section 6.1.1, and then consider some examples of operational monads, in Section 6.1.2.

### 6.1.1 Operational monads

Before defining operational monads, we need the intermediate notion of  $T$ -reduction monads, when  $T$  is a pair of endofunctors on  $\text{Set}$ .

**Definition 190.** Let  $T = (T_1, T_2)$  be a pair of endofunctors on  $\text{Set}$ . 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  $\text{Red}(R)$  (the module of *reductions*);
3. a morphism of  $R$ -modules  $\text{red}_R : \text{Red}(R) \rightarrow (T_1 \cdot R) \times (T_2 \cdot R)$  (*source and target of rules*).

We set  $\text{source}_R := \pi_1 \circ \text{red}_R : \text{Red}(R) \rightarrow T_1 \cdot R$ , and  $\text{target}_R := \pi_2 \circ \text{red}_R : \text{Red}(R) \rightarrow T_2 \cdot R$ .

**Notation 191.** For a  $T$ -reduction monad  $R$ , a set  $X$ , and elements  $s \in T_1(R(X))$ ,  $t \in T_2(R(X))$ , we think of the fiber  $\text{red}_R(X)^{-1}(s, t)$  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  $m \in \text{red}_R(X)^{-1}(s, t)$ .

**Remark 192.** We recover Chapter 5 by taking  $T = (\text{Id}_{\text{Set}}, \text{Id}_{\text{Set}})$ .

**Definition 193.** Let  $T = (T_1, T_2)$  be a pair of endofunctors on  $\text{Set}$ . A **morphism of  $T$ -reduction monads** from  $R$  to  $S$  is given by a pair  $(f, \alpha)$  of

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

satisfying the following two conditions:

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

$$\begin{array}{ccc} \text{Red}(R) & \xrightarrow{\alpha} & \text{Red}(S) \\ \text{red}_R \downarrow & & \downarrow \text{red}_S \\ (T_1 \cdot R) \times (T_2 \cdot R) & \xrightarrow{T_1 f \times T_2 f} & (T_1 \cdot S) \times (T_2 \cdot S) \end{array}$$

commutes in the category of natural transformations.

**Proposition 194** (Category of  $T$ -reduction monads). *Let  $T = (T_1, T_2)$  be a pair of endofunctors on  $\text{Set}$ . Then,  $T$ -reduction monads and their morphisms, with the obvious composition and identity, form a category  $T\text{-RedMon}$ , equipped with a forgetful functor to the category of monads.*

We get an analogue of Theorem 120:

**Theorem 195.** *Let  $T = (T_1, T_2)$  be a pair of endofunctors on  $\text{Set}$ . 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  $\text{Set}/(T_1 \times T_2)$ .*

**Definition 196.** An **operational monad** is a pair  $(T, R)$  where  $R$  is a  $T$ -reduction monad. The pair  $T = (T_1, T_2)$  of endofunctors on  $\text{Set}$  is called the pair of **state functors** of the operational monad.

Operational monads may be organized into a category:

**Definition 197** (Morphism of operational monads). A **morphism between operational monads**  $((T_1, T_2), R)$  and  $((T'_1, T'_2), R')$  is a triple  $(\alpha_1, \alpha_2, f)$  consisting of

- a natural transformation  $\alpha_1 : T_1 \rightarrow T'_1$ ;
- a natural transformation  $\alpha_2 : T_2 \rightarrow T'_2$ ;
- a  $(T'_1, T'_2)$ -reduction monad morphism  $f$  between  $(\alpha_1, \alpha_2)_! R$  and  $R'$ , where the  $(T'_1, T'_2)$ -reduction monad  $(\alpha_1, \alpha_2)_! R$  is defined as follows:
  - the underlying monad is  $R$ ;

- $\text{Red}((\alpha_1, \alpha_2)_! R)$  is defined as  $\text{Red}(R)$ ;
- $\text{red}_{(\alpha_1, \alpha_2)_! R}$  is defined as the composition

$$\text{Red}(R) \xrightarrow{\text{red}_R} (T_1 \cdot R) \times (T_2 \cdot R) \xrightarrow{(\alpha_1 \cdot R) \times (\alpha_2 \cdot R)} (T'_1 \cdot R) \times (T'_2 \cdot R) .$$

**Proposition 198** (Category of operational monads). *Operational monads and their morphisms with the obvious composition and identity morphisms form a category  $\text{OpMon}$ .*

### 6.1.2 Examples of operational monads

Any example of reduction monad  $R$  of Chapter 5 yields an operational monad whose state functors are the identity endofunctors.

**Example 199** (Call-by-value lambda calculus). In call-by-value, reductions between terms are not stable under substitution of variables with terms in general. However, they are stable under substitution of variables 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  $\text{LC}_v = \text{Id} + \text{LC}'$  equipped with a monad morphism  $i : \text{LC}_v \rightarrow \text{LC}$ .

Now, reductions are between terms rather than between values, so we need to devise a functor  $B : \text{Set} \rightarrow \text{Set}$  such that  $B \cdot \text{LC}_v$  is isomorphic to  $\text{LC}$ : then, the call-by-value lambda calculus is a  $(B, B)$ -reduction monad. 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  $B$  to be the functor underlying the monad of binary trees specified by the 1-signature  $\Theta \times \Theta$ : then, there is an isomorphism  $\alpha : \text{LC} \rightarrow B \cdot \text{LC}_v$  in the category of  $\text{LC}_v$ -modules ( $\text{LC}$  is indeed equipped with a structure of  $\text{LC}_v$ -module thanks to the inclusion  $i$  of  $\text{LC}_v$  in  $\text{LC}$  as a monad morphism). The operational monad of call-by-value lambda calculus is the pair  $((B, B), \text{LC}_v)$  where the  $(B, B)$ -reduction monad  $\text{LC}_v$  is defined as follows:

1. the underlying monad is  $\text{LC}_v = \text{Id} + \text{LC}'$ ;
2.  $\text{red}_{\text{LC}_v} : \text{Red}(\text{LC}_v) \rightarrow (B \cdot \text{LC}_v) \times (B \cdot \text{LC}_v)$  is generated by the following constructions:
  - (a) for  $t \in \text{LC}'(X)$  and  $u \in \text{LC}_v(X)$ , we have  $\beta(t, u) : \alpha(\text{app}(\text{abs}(t), i(u))) \blacktriangleright \alpha(t)\{* := u\}$ ;

- (b) for  $m : \alpha(u) \blacktriangleright \alpha(v) : B(\text{LC}_v(X))$  in  $\text{Red}(\text{LC}_v)$  and  $t \in \text{LC}(X)$ , we have  $\text{app-cong}_1(m, t) : \alpha(\text{app}(u, t)) \blacktriangleright \alpha(\text{app}(v, t))$ ;
- (c) for  $m : \alpha(u) \blacktriangleright \alpha(v) : B(\text{LC}_v(X))$  in  $\text{Red}(\text{LC}_v)$  and  $t \in \text{LC}(X)$ , we have  $\text{app-cong}_2(t, m) : \alpha(\text{app}(t, u)) \blacktriangleright \alpha(\text{app}(t, v))$ ;
- (d) for  $m : \alpha(u) \blacktriangleright \alpha(v) : B(\text{LC}'_v(X))$  in  $\text{Red}(\text{LC}_v)$  we have  $\text{abs-cong}(m) : \alpha(\text{abs}(u)) \blacktriangleright \alpha(\text{abs}(v))$ .

We call this operational monad *the operational monad of the call-by-value lambda calculus*. A signature for it is given in Section 6.4.1.

**Example 200** (Big-step call-by-value lambda calculus). In the call-by-value lambda calculus with big-step operational semantics, reductions happen between a term and a value. The underlying monad is  $\text{LC}_v$ , and the pair of state functors is  $(B, \text{Id})$ , where  $B$  is defined in Example 199. We denote  $\text{abs}_v : \text{LC}' \rightarrow \text{LC}_v$  the inclusion of lambda abstractions into values. The module of reductions is defined inductively as follows:

- for each value  $v \in \text{LC}_v(X)$ , the induced term  $\alpha(i(v))$  reduces to  $v$ ;
- given a reduction  $m_t : \alpha(t) \blacktriangleright \text{abs}_v(t')$ , a reduction  $m_u : \alpha(u) \blacktriangleright u'$ , and a reduction  $m : \alpha(t')\{* := u'\} \blacktriangleright v$ , we get a reduction  $\beta(m_t, m_u, m) : \alpha(\text{app}(t, u)) \blacktriangleright v$ .

We specify more formally this operational monad in Section 6.4.2.

**Example 201.** Let us recall the following simple variant of  $\pi$ -calculus. The syntax for *processes* is given by

$$P, Q ::= 0 \mid (P|Q) \mid !P \mid \nu a.P \mid \bar{a}\langle b \rangle.P \mid a(b).P,$$

where  $a$  and  $b$  range over a fixed, countable set of *channel names*. In  $\nu a.P$ , the channel name  $a$  is bound, and in  $a(b).P$ , the channel name  $b$  is bound. Processes will be considered equivalent up to *structural congruence*, the smallest equivalence relation  $\equiv$  stable under context and such that  $0|P \equiv P$ ,  $P|(Q|R) \equiv (P|Q)|R$ ,  $(\nu a.P)|Q \equiv \nu a.(P|Q)$  when  $a$  does not occur free in  $Q$ , and  $!P \equiv P|!P$ . Reduction is then given by the following

inductive rules, the last one enforcing the quotient by structural congruence.

$$\begin{array}{c}
 \frac{}{\overline{a}\langle b \rangle . P | a(c) . Q \rightsquigarrow P | (Q\{c := b\})} \qquad \frac{P \rightsquigarrow Q}{P | R \rightsquigarrow Q | R} \qquad \frac{P \rightsquigarrow Q}{\nu a . P \rightsquigarrow \nu a . Q} \\
 \\
 \frac{P \equiv P' \quad P' \rightsquigarrow Q' \quad Q' \equiv Q}{P \rightsquigarrow Q}
 \end{array}$$

In  $\pi$ -calculus all we substitute is channel names, so the monad is just the identity. There is no need for process variables: no binding of process variable is involved, nor substitution of variables with processes.

The state functor  $T$  maps any  $X \in \text{Set}$  to the set of processes with free channel names in  $X$ , considered equivalent up to structural congruence. Finally, let  $\text{Red}(R)(X)$  denote the set of reductions between (equivalence classes of) processes in  $T(X)$ .

We specify more formally this operational monad in Section 6.4.3.

## 6.2 $\mathbb{S}$ -reduction rules

This section is a straightforward generalization of the notion of reduction rules of Section 5.3 to the setting of operational monads, when  $\mathbb{S}$  is a pair of signatures over the category of endofunctors on  $\text{Set}$  (in the sense of Chapter 2):  $\mathbb{S}$ -reduction rules are defined in Section 6.2.1, and the associated notion of action in a  $\mathbb{S}$ -reduction monad is explained in Section 6.2.2, after introducing  $\mathbb{S}$ -reduction  $\Sigma$ -monads in Section 6.2.2, when  $\Sigma$  is a signature for monads.

### 6.2.1 Definition of $\mathbb{S}$ -reduction rules

In this subsection,  $\Sigma$  is a signature for monads, and  $\mathbb{S} = (\mathbb{S}_1, \mathbb{S}_2)$  is a pair of signatures over the category of endofunctors on  $\text{Set}$ . We present our notion of  $\mathbb{S}$ -reduction rule over  $\Sigma$ , from which we build *operational signatures* in Section 6.3.

We need first to generalize the notion of  $\Sigma$ -module from Definition 73 to that of  $(\Sigma, \mathbb{S})$ -module:

**Definition 202.** We define a  $(\Sigma, \mathbb{S})$ -**module** to be a functor  $T$  from the product of the categories of models of  $\Sigma$ ,  $\mathbb{S}_1$ , and  $\mathbb{S}_2$  to the category  $\int \text{Mod}$  commuting with the forgetful

functors to the category  $\text{Mon}$  of monads,

$$\begin{array}{ccc} \text{Mon}^\Sigma \times [\text{Set}, \text{Set}]^{\mathbb{S}_1} \times [\text{Set}, \text{Set}]^{\mathbb{S}_2} & \xrightarrow{T} & \int \text{Mod} \\ & \searrow & \swarrow \\ & \text{Mon} & \end{array}$$

**Example 203.** To each  $\Sigma$ -module  $\Psi$  is associated, by precomposition with the projection to  $\text{Mon}^\Sigma$ , a  $(\Sigma, \mathbb{S})$ -module still denoted  $\Psi$ .

**Definition 204.** Let  $S$  and  $T$  be  $(\Sigma, \mathbb{S})$ -modules. We define a **morphism of  $(\Sigma, \mathbb{S})$ -modules** from  $S$  to  $T$  to be a natural transformation from  $S$  to  $T$  which becomes the identity when postcomposed with the forgetful functor  $\int \text{Mod} \rightarrow \text{Mon}$ .

Then we define  $\mathbb{S}$ -term-pairs:

**Definition 205.** Given a  $\Sigma$ -module  $\mathcal{V}$ , a  **$\mathbb{S}$ -term-pair from  $\mathcal{V}$**  is a pair  $(n, p)$  of a natural number  $n$  and a morphism of  $(\Sigma, \mathbb{S})$ -modules  $p : \mathcal{V} \rightarrow (\mathcal{T}_1 \cdot \Theta^{(n)}) \times (\mathcal{T}_2 \cdot \Theta^{(n)})$ , where  $(\mathcal{T}_i \cdot \Theta^{(n)})$  is the  $(\Sigma, \mathbb{S})$ -module mapping a triple  $(R, T_1, T_2)$  to the  $R$ -module  $T_i \cdot R$ .

We now give our definition of  $\mathbb{S}$ -reduction rule.

**Definition 206.** A  **$\mathbb{S}$ -reduction rule  $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$  over  $\Sigma$**  is given by:

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

## 6.2.2 $\mathbb{S}$ -Reduction $\Sigma$ -monads

Similarly to Chapter 5, the purpose of a  $\mathbb{S}$ -reduction rule is to be validated in a  $\mathbb{S}$ -reduction monad  $R$ , that is, an operational monad whose underlying state functors are models of  $\mathbb{S}$ . Here again, as the hypotheses or the conclusion of the  $\mathbb{S}$ -reduction rule may refer to some operations specified by a signature  $\Sigma$  for monads, this  $\mathbb{S}$ -reduction monad  $R$  must be equipped with an action of  $\Sigma$ , hence the following definition.

**Definition 207.** Let  $\Sigma$  be a signature for monads, and  $\mathbb{S} = (\mathbb{S}_1, \mathbb{S}_2)$  be a pair of signatures over the category of endofunctors on  $\text{Set}$ . The **category**  $\mathbb{S}\text{-RedMon}^\Sigma$  **of**  **$\mathbb{S}$ -reduction  $\Sigma$ -monads** is defined as the following pullback.

$$\begin{array}{ccc} \mathbb{S}\text{-RedMon}^\Sigma & \xrightarrow{\quad} & \text{OpMon} \\ \downarrow & & \downarrow \\ \text{Mon}^\Sigma \times [\text{Set}, \text{Set}]^{\mathbb{S}_1} \times [\text{Set}, \text{Set}]^{\mathbb{S}_2} & \longrightarrow & \text{Mon} \times [\text{Set}, \text{Set}]^2 \end{array}$$

More concretely,

- a  **$\mathbb{S}$ -reduction  $\Sigma$ -monad** is an operational monad  $(R, T_1, T_2)$  equipped with
  - an action  $\rho$  of  $\Sigma$  in  $R$ , thus inducing a  $\Sigma$ -monad that we denote also by  $R$ , or by  $\underline{R}$  when we want to be explicit;
  - an action of  $\mathbb{S}_i$  in  $T_i$  for  $i \in \{1, 2\}$ , inducing a model of  $\mathbb{S}_i$  that we denote also by  $T_i$ ;
- a **morphism of  $\mathbb{S}$ -reduction  $\Sigma$ -monads**  $R \rightarrow S$  is a morphism  $f : R \rightarrow S$  of operational monads compatible with the actions of  $\Sigma$ ,  $\mathbb{S}_1$ , and  $\mathbb{S}_2$ , i.e, whose underlying monad morphism is a  $\Sigma$ -monad morphism and whose underlying natural transformations between state functors are model morphisms.

### 6.2.3 Action of a $\mathbb{S}$ -reduction rule

Let  $\Sigma$  be a signature for monads and  $\mathbb{S} = (\mathbb{S}_1, \mathbb{S}_2)$  be a pair of signatures over the category of endofunctors on  $\text{Set}$ . In this section, we introduce the notion of *action of a  $\mathbb{S}$ -reduction rule over  $\Sigma$  in a  $\mathbb{S}$ -reduction  $\Sigma$ -monad*. Intuitively, such an action is a “map from the hypotheses to the conclusion” of the  $\mathbb{S}$ -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 208.** Let  $(n, p)$  be a  $\mathbb{S}$ -term-pair from a  $\Sigma$ -module  $\mathcal{V}$ , and  $R$  be a  $\mathbb{S}$ -reduction  $\Sigma$ -monad. We denote by  $p^*(\text{Red}(R)^{(n)})$  the pullback of  $\text{red}_R^{(n)} : \text{Red}(R)^{(n)} \rightarrow (T_1 \cdot R^{(n)}) \times$



$(T_2 \cdot R^{(n)})$  along  $p_R : \mathcal{V}(R) \rightarrow (T_1 \cdot R^{(n)}) \times (T_2 \cdot R^{(n)})$ :

$$\begin{array}{ccc} p^*(\text{Red}(R)^{(n)}) & \xrightarrow{\quad} & \text{Red}(R)^{(n)} \\ \downarrow & \lrcorner & \downarrow \text{red}_R^{(n)} \\ \mathcal{V}(R) & \xrightarrow{p_R} & (T_1 \cdot R^{(n)}) \times (T_2 \cdot R^{(n)}) \end{array} .$$

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

**Definition 209.** Let  $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$  be a  $\mathbb{S}$ -reduction rule, and  $R$  be a  $\mathbb{S}$ -reduction  $\Sigma$ -monad.

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

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

**Definition 210.** Let  $\mathcal{A}$  be a  $\mathbb{S}$ -reduction rule over  $\Sigma$ . An **action of  $\mathcal{A}$  in a  $\mathbb{S}$ -reduction  $\Sigma$ -monad  $R$**  is a morphism, in the slice category  $\text{Mod}(R)/\text{MVar}_{\mathcal{A}}(R)$ , between  $\text{hyp}_{\mathcal{A}}(R)$  and  $\text{con}_{\mathcal{A}}(R)$ , that is, a morphism of  $R$ -modules

$$\tau : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \text{Con}_{\mathcal{A}}(R)$$

making the following diagram commute:

$$\begin{array}{ccc} \text{Hyp}_{\mathcal{A}}(R) & \xrightarrow{\tau} & \text{Con}_{\mathcal{A}}(R) \\ & \searrow & \swarrow \\ & \text{MVar}_{\mathcal{A}}(R) & \end{array} . \quad (6.1)$$

**Remark 211.** Any  $\mathbb{S}$ -reduction rule  $\mathcal{A}$  over  $\Sigma$  defines an arity  $(D, a, u, v)$  over the category of  $\mathbb{S}$ -reduction  $\Sigma$ -monads (in the sense of Chapter 2) as follows:

- $a : D \rightarrow \mathbb{S}\text{-RedMon}^{\Sigma}$  is the fibration corresponding (through the Grothendieck construction) to the functor mapping a  $\mathbb{S}$ -reduction  $\Sigma$ -monad  $R$  to  $\text{Mod}(R)/\text{MVar}_{\mathcal{A}}(R)$ .
- $u$  maps a  $\mathbb{S}$ -reduction  $\Sigma$ -monad to  $\text{hyp}_{\mathcal{A}}(R) : \text{Hyp}_{\mathcal{A}}(R) \rightarrow \text{MVar}_{\mathcal{A}}(R)$ ;
- $v$  maps a  $\mathbb{S}$ -reduction  $\Sigma$ -monad to  $\text{con}_{\mathcal{A}}(R) : \text{Con}_{\mathcal{A}}(R) \rightarrow \text{MVar}_{\mathcal{A}}(R)$ .

This arity yields the same notion of action.

## 6.3 Signatures for operational monads and Initiality

In this section, we define the notion of *operational signature* specifying an operational monad  $(T, R)$  through:

- a specification of the pair  $T$  of state functors, as a pair of signatures  $\mathbb{S} = (\mathbb{S}_1, \mathbb{S}_2)$  over the category of endofunctors on  $\mathbf{Set}$ ,
- a signature for the monad underlying  $R$ ,
- and a family of  $\mathbb{S}$ -reduction rules over  $\Sigma$ .

In Section 6.3.4, we review Fiore and Hur's notion of equational system [FH09] that yield effective signatures for specifying endofunctors. An operational signature induces a signature over the category of operational monads in the sense of Chapter 2. In Sections 6.3.1 and 6.3.3, we describe the *category of models* associated to each operational signature. Our main result, Theorem 221, states that an operational signature is effective as soon as its underlying signature for monads is effective.

### 6.3.1 Operational signatures and their models

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

**Definition 212.** An **operational signature** is a quadruple  $(\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  consisting of:

- signatures  $\mathbb{S}_1$  and  $\mathbb{S}_2$  over the category of endofunctors;
- a signature  $\Sigma$  for monads;
- a family  $\mathfrak{R}$  of  $(\mathbb{S}_1, \mathbb{S}_2)$ -reduction rules over  $\Sigma$ .

Examples are given in Section 6.4.

**Remark 213.** An operational signature  $(\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  defines a signature over the category of operational monads (in the sense of Chapter 2) consisting of four parts.

The first part is given by the pullback (in the sense of Definition 18) of the signature  $S_\Sigma$  (defined in Remark 35) along the forgetful functor from the category of operational

monads to the category of monads. The associated category of models is the category of *operational  $\Sigma$ -monads*, that is, operational monads whose underlying monad is equipped with an action of  $\Sigma$ .

The second part is a family of two arities, which are pullbacks of  $\mathbb{S}_1$  and  $\mathbb{S}_2$  along the forgetful functor to the category of endofunctors on  $\mathbf{Set}$ . The associated category of models is the category of  $(\mathbb{S}_1, \mathbb{S}_2)$ -reduction  $\Sigma$ -monads.

The last part is the family of arities induced by the reduction rules of  $\mathfrak{R}$ , as explained in 211.

**Definition 214.** Let  $\mathcal{O} = (\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  be an operational signature. Let  $\mathbb{S} := (\hat{\mathbb{S}}_1, \hat{\mathbb{S}}_2)$ . A **model of  $\mathcal{O}$**  is a  $\mathbb{S}$ -reduction  $\Sigma$ -monad equipped with an action of each reduction rule of  $\mathfrak{R}$ .

### 6.3.2 The functors $\text{Hyp}_{\mathcal{A}}$ and $\text{Con}_{\mathcal{A}}$

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

**Definition 215.** Let  $\mathbb{S}$  be a pair of signatures over the category of endofunctors on  $\mathbf{Set}$ . Let  $\Sigma$  be a signature for monads, and  $\mathcal{A}$  be a  $\mathbb{S}$ -reduction rule over  $\Sigma$ . Definition 209 assigns to each  $\mathbb{S}$ -reduction  $\Sigma$ -monad  $R$  the  $R$ -modules  $\text{Hyp}_{\mathcal{A}}(R)$  and  $\text{Con}_{\mathcal{A}}(R)$ . These assignments extend to functors  $\text{Hyp}_{\mathcal{A}}, \text{Con}_{\mathcal{A}} : \mathbb{S}\text{-RedMon}^{\Sigma} \rightarrow \int \mathbf{Mod}$ .

**Proposition 216.** *Given the same data, the functors  $\text{Hyp}_{\mathcal{A}}$  and  $\text{Con}_{\mathcal{A}}$  commute with the forgetful functors to  $\mathbf{Mon}$ :*

$$\begin{array}{ccc}
 T\text{-RedMon}^{\Sigma} & \begin{array}{c} \xrightarrow{\text{Hyp}_{\mathcal{A}}} \\ \xrightarrow{\text{Con}_{\mathcal{A}}} \end{array} & \int \mathbf{Mod} \\
 & \searrow \quad \swarrow & \\
 & \mathbf{Mon} &
 \end{array}$$

### 6.3.3 The main result

For an operational signature  $\mathcal{O}$ , we define here the notion of  $\mathcal{O}$ -model morphism, inducing a **category of models of  $\mathcal{O}$** . We then state our main result, Theorem 221, which gives a sufficient condition for  $\mathcal{O}$  to admit an initial model.

**Definition 217.** Let  $\mathcal{O} = (\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  be an operational signature. Let  $T := (\hat{\mathbb{S}}_1, \hat{\mathbb{S}}_2)$ . A morphism between models  $R$  and  $S$  of  $\mathcal{O}$  is a morphism  $f$  of  $T$ -reduction  $\Sigma$ -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:

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

**Proposition 218.** Let  $\mathcal{O} = (\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  be an operational signature. Models of  $\mathcal{O}$  and their morphisms, with the obvious composition and identity, define a category that we denote by  $\text{OpMon}^{\mathcal{O}}$ , equipped with a forgetful functor to  $(\hat{\mathbb{S}}_1, \hat{\mathbb{S}}_2)\text{-RedMon}^{\Sigma}$ .

**Remark 219.** The category of models of an operational signature  $\mathcal{O}$  is the same as the one associated to the induced signature over the category of operational monads, as explained in Remark 213.

**Definition 220.** An operational signature  $\mathcal{O}$  is said to be **effective** if its category of models  $\text{OpMon}^{\mathcal{O}}$  has an initial object, denoted  $\hat{\mathcal{O}}$ . In this case, we say that  $\hat{\mathcal{O}}$  is **generated (or specified) by  $\mathcal{O}$** .

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

**Theorem 221.** Let  $\mathcal{O} = (\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  be an operational signature. If  $\mathbb{S}_1$ ,  $\mathbb{S}_2$ , and  $\Sigma$  are effective, then so is  $\mathcal{O}$ .

*Proof.* This is a close adaptation of the proof detailed in Section 5.5. Therefore, we only sketch the main arguments.

Consider the two following pullbacks, the left one defining  $\text{OpMon}^{\Sigma, \mathbb{S}}$ :

$$\begin{array}{ccccc} \text{OpMon}^{\Sigma, \mathbb{S}} & \xrightarrow{\quad} & \text{OpMon} & \xrightarrow{\quad} & V(f \text{ Mod}) \\ \downarrow & \lrcorner & \downarrow & \lrcorner & \downarrow \text{cod} \\ \text{Mon}^{\Sigma} \times [\text{Set}, \text{Set}]^{\mathbb{S}_1} \times [\text{Set}, \text{Set}]^{\mathbb{S}_2} & \longrightarrow & \text{Mon} \times [\text{Set}, \text{Set}]^2 & \xrightarrow{(R, T_1, T_2) \mapsto (T_1 \cdot R, T_2 \cdot R)} & f \text{ Mod} \end{array}$$

Then, the left vertical arrow is a fibration by the same argument detailed in Lemma 168. Moreover, the category of models of  $\mathcal{O}$  is isomorphic to the category of vertical algebras of an endofunctor  $F_{\mathcal{O}}$  on  $\text{OpMon}^{\Sigma, \mathbb{S}}$ , by the same argument as in Proposition 165. Then,

this category of models is fibered over  $\text{Mon}^\Sigma \times [\text{Set}, \text{Set}]^{\mathbb{S}_1} \times [\text{Set}, \text{Set}]^{\mathbb{S}_2}$  by Lemma 167, and the fiber category over  $(R, T_1, T_2)$  is the category of algebras of the finitary restriction of  $F_O$  as an endofunctor on  $\text{Mod}(R)/(T_1 \cdot R \times T_2 \cdot R)$ , following an argument similar to that of Lemmas 170 and 173.

Now, we apply Lemma 169: we only need to show that the fiber category over  $(\hat{\Sigma}, \hat{\mathbb{S}}_1, \hat{\mathbb{S}}_2)$  has an initial object, that is, we must construct an initial algebra for a finitary endofunctor thanks to the previous remark. Adámek's theorem (Lemma 171) allows to conclude. □

We already know a class of effective signatures for monads: they are the algebraic 2-signatures (Theorem 104). In the next section, we introduce a similar notion of algebraic signature for endofunctors based on equational systems whose effectivity is ensured by [FH09]. This will allow us to state Corollary 237, as a consequence of Theorem 221, that we will use for our examples.

### 6.3.4 Specifying the state functors

We choose to rely on Fiore and Hur's notion of equational system [FH09]. Thanks to Example 13, they can be considered as signatures in the sense of Chapter 2.

A first definition of signature for endofunctors on  $\text{Set}$  that we can think of goes as follows:

**Definition 222.** The **category of 1-signatures for endofunctors** is the category of endofunctors on the category of endofunctors on  $\text{Set}$ , with natural transformations as morphisms. The **category of models of a 1-signature  $\Sigma$  for endofunctors** is defined as the category of models of the induced signature over the category of endofunctors (see Example 12): it is the category  $\Sigma\text{-alg}$  of algebras.

**Proposition 223.** *The category of 1-signatures for endofunctors is complete and co-complete: limits and colimits are computed pointwise.*

**Notation 224.** *The following is reminiscent of 1-signatures for monads:*

- $\Theta$  is the identity endofunctor  $\text{Id}$  on the category of endofunctors on  $\text{Set}$ ;

- if  $\Sigma$  is a 1-signature for endofunctors, we denote by  $\Sigma^{(n)}$  its  $n^{th}$  derivative, where  $\Sigma^{(n)}(F) := \Sigma(F)^{(n)}$  mapping a set  $X$  to the set  $\Sigma(F)(X + n)$ , for any endofunctor  $F$ ;
- $\Sigma' := \Sigma^{(1)}$ , for any 1-signature  $\Sigma$  for endofunctors;
- if  $F$  is an endofunctor on  $\text{Set}$ , the constant 1-signature mapping any functor to  $F$  is denoted by  $\underline{F}$ ;
- if  $\Sigma = \coprod_{i \in I} \Theta^{(n_{j_1})} \times \dots \times \Theta^{(n_{j_i})}$  is an algebraic 1-signature for monads, we denote  $\bar{\Sigma}$  the 1-signature for endofunctors defined by the same formula.

The clash of notations with 1-signatures for monads is justified by the fact that given any monad  $R$  and algebraic 1-signature  $\Sigma$  for monads, the functor underlying the image of  $R$  by  $\Sigma$  is the image of the functor underlying  $R$  by the 1-signature  $\bar{\Sigma}$ . More formally:

**Proposition 225.** *Let  $\Sigma$  be an algebraic 1-signature for monads. Then, the following diagram commutes:*

$$\begin{array}{ccc} \text{Mon} & \xrightarrow{\Sigma} & \int \text{Mod} \\ \downarrow & & \downarrow \\ [\text{Set}, \text{Set}] & \xrightarrow{\bar{\Sigma}} & [\text{Set}, \text{Set}] \end{array}$$

We now introduce algebraic 1-signatures for endofunctors, which satisfy an initiality theorem:

**Definition 226.** A 1-signature  $\Sigma$  for endofunctors is said **algebraic** if it is a coproduct of endofunctors of the shape  $\underline{\text{Id}}_{\text{Set}}^{(i_1)} \times \dots \times \underline{\text{Id}}_{\text{Set}}^{(i_n)} \times \Theta^{(j_1)} \times \dots \times \Theta^{(j_m)}$ .

**Example 227.** If  $\Sigma$  is an algebraic 1-signature for monads, then  $\bar{\Sigma}$  is an algebraic 1-signature for endofunctors.

**Proposition 228.** *Any algebraic 1-signature  $\Sigma$  for endofunctors has an initial model, denoted by  $\hat{\Sigma}$ .*

*Proof.* This follows from Adámek's theorem (Lemma 171), as algebraic 1-signatures are finitary. □

Actually, the functor underlying the initial model of an algebraic 1-signature for monads is specified by an appropriate algebraic 1-signature for endofunctors:

**Proposition 229.** *Let  $\Sigma$  be an algebraic 1-signature for monads. Then the functors underlying  $\widehat{\Sigma}$  and  $\widehat{\Sigma} + \text{Id}_{\text{Set}}$  are isomorphic.*

Next, we would like to specify equations. To this end, we use equational systems [FH09], that we rephrase to make the similarity more explicit with the 2-signatures of Chapter 4. We focus on equations of the shape  $u_R = v_R : \Psi(R) \rightarrow R$ , where

- $R$  is an algebra of a 1-signature  $\Sigma$ ;
- $\Psi(R)$  is an endofunctor on  $\text{Set}$ ;
- $u_R$  and  $v_R$  are parallel natural transformations between  $\Psi(R)$  and  $R$ .

The assignment  $R \mapsto (\Psi(R), u_R, v_R)$  is required to be functorial. This suggests the following definition:

**Definition 230.** Let  $\Sigma$  be a 1-signature for functors. A  $\Sigma$ -**equation** is a triple  $(\Psi, u, v)$  consisting of:

- a functor  $\Psi : \Sigma\text{-alg} \rightarrow [\text{Set}, \text{Set}]$ ;
- a pair of parallel natural transformations  $u, v$  from  $\Psi$  to  $\Theta$ , where  $\Theta : \Sigma\text{-alg} \rightarrow [\text{Set}, \text{Set}]$  maps an algebra to its underlying object.

It is said **algebraic** if  $\Psi$  is.

**Remark 231.** Let  $\Sigma$  be a 1-signature for endofunctors. A  $\Sigma$ -equation  $(\Psi, u, v)$  induces an equational system  $(C : \Sigma \triangleright \Psi \vdash L = R)$ , as introduced in Example 9. Indeed,  $L : \Sigma\text{-alg} \rightarrow \Psi\text{-alg}$  maps a  $\Sigma$ -algebra  $F$  to the  $\Psi$ -algebra  $u_F : \Psi(F) \rightarrow F$ , and similarly,  $R$  maps a  $\Sigma$ -algebra  $F$  to the  $\Psi$ -algebra  $v_F : \Psi(F) \rightarrow F$ .

Thus, by Example 9, a  $\Sigma$ -equation induces an equational arity over the category of algebras of  $\Sigma$ .

**Definition 232.** A **2-signature for endofunctors** (or just **signature for endofunctors**) is a pair of a 1-signature  $\Sigma$  and a set  $E$  of  $\Sigma$ -equations. It is said **algebraic** if  $\Sigma$  is and  $E$  consist of algebraic  $\Sigma$ -equations.

The associated category of models is the full subcategory of models  $R$  of  $\Sigma$  satisfying  $u_R = v_R$  for each  $(\Psi, u, v) \in E$ .

**Remark 233.** A 2-signature  $(\Sigma, E)$  induces a signature over the category of endofunctors on  $\text{Set}$ , in the sense of Chapter 2, which has two components:

- the first one is a singleton family consisting of the arity induced by  $\Sigma$  (Example 3);
- the second one is the family of equational arities induced by each  $\Sigma$ -equation, as explained in Remark 231.

This induced signature yields the same category of models.

**Definition 234.** A signature  $\mathbb{S}$  for endofunctors is said **effective** if it has an initial model, denoted by  $\hat{\mathbb{S}}$ .

**Proposition 235.** Any algebraic signature for endofunctors is effective.

*Proof.* A first step consists in combining all the equations of the 2-signature into a single one, using coproducts. Then, the result follows from [FH09, Theorem 4.7].  $\square$

**Example 236.** Recalling Example 201, we would like to specify the state functor  $T : \text{Set} \rightarrow \text{Set}$ , such that  $T(A)$  denotes the set of  $\pi$ -calculus processes up to structural congruence with free channels in  $A$ . For this, let us consider the endofunctor on  $[\text{Set}, \text{Set}]$  defined for all  $X \in [\text{Set}, \text{Set}]$  and  $\gamma \in \text{Set}$  by

$$\begin{aligned} \Sigma(X)(\gamma) &= 1 + X(\gamma)^2 + X(\gamma) + X(\gamma + 1) + \gamma^2 \times X(\gamma) + \gamma \times X(\gamma + 1) \\ P, Q &::= 0 \mid (P|Q) \mid !P \mid \nu a.P \mid \bar{a}\langle b \rangle.P \mid a(b).P \\ (\Sigma &= 1 + \Theta^2 + \Theta + \Theta' + \text{Id}^2 \times \Theta + \text{Id} \times \Theta') \end{aligned}$$

(with corresponding pieces of syntax below each term of the sum, and using the notations of Definition 226 for the last line). As an example of  $\Sigma$ -equation  $(\Psi, u, v)$ , commutativity of parallel composition would have  $\Psi(X)(\gamma) = X(\gamma)^2$ . The first natural transformation  $u : \Psi \rightarrow \Theta$  maps any  $\Sigma$ -alg, say  $\rho : \Sigma X \rightarrow X$ , to the natural transformation with component  $u_{X,\rho,\gamma} : \Psi(X)(\gamma) = X(\gamma)^2 \rightarrow X(\gamma)$  at  $\gamma$  given by parallel composition, i.e., the composite

$$X(\gamma)^2 \xrightarrow{(\text{inj}_2)_\gamma} \Sigma(X)(\gamma) \xrightarrow{\rho_\gamma} X(\gamma),$$

and  $v_{X,\rho,\gamma}$  given by swapping, and then parallel composition. All equations may be treated similarly, forming a signature  $\mathbb{S}$  for endofunctors which is algebraic, whose initial algebra is the desired  $T$ .

Proposition 235 and Theorem 221 entail the following corollary of Theorem 221.

**Corollary 237.** Let  $\mathcal{O} = (\mathbb{S}_1, \mathbb{S}_2, \Sigma, \mathfrak{R})$  be an operational signature. If  $\mathbb{S}_1$ ,  $\mathbb{S}_2$ , and  $\Sigma$  are algebraic, then  $\mathcal{O}$  is effective.



All the examples of operational signatures considered here satisfy the algebraicity condition of Corollary 237.

## 6.4 Examples of operational signatures

We give examples of operational signatures for the call-by-value lambda calculus in Section 6.4.1, for the big-step operational semantics variant in Section 6.4.2, and for the  $\pi$ -calculus in Section 6.4.3. The reduction rules are given following the schematic presentation described in Section 5.3.5. These signatures all satisfy the hypotheses of Corollary 237, and thus are effective.

### 6.4.1 Call-by-value lambda calculus

We give an operational signature for the operational monad  $((B, B), LC_v)$  of the call-by-value lambda calculus from Example 199. Recall that  $B$  is the monad of binary trees underlying the initial model of the 1-signature  $\Theta \times \Theta$ . By Proposition 229, the corresponding algebraic signature for endofunctor is  $\mathbb{S} = (\Theta \times \Theta + \text{Id}_{\text{Set}}, \emptyset)$ . A model  $T$  of  $\mathbb{S}$  comes equipped with a binary operation  $\text{app} : T \times T \rightarrow T$  and a variable operation  $v : \text{Id}_{\text{Set}} \rightarrow T$ . We denote by  $j_T : B \rightarrow T$  the initial morphism.

Now, we give a signature for the monad  $LC_v$  of values of the lambda calculus. A value is either a variable or a an abstracted lambda term, and we argued in Example 199 that a lambda term can be specified as a binary tree whose leaves are values. Thus, we choose the 1-signature  $B \cdot \Theta'$  to specify the monad of values. Note that this is algebraic as  $B$  is polynomial: we have

$$B \cdot \Theta' = \coprod_{n \in \mathbb{N}} B_n \times (\Theta')^n,$$

where  $B_n$  is the set of binary trees with  $n$  leaves. A model of this signature is a monad  $R$  with a module morphism  $\text{abs}_R : B \cdot R' \rightarrow R$ .

We give now the family  $\mathfrak{R}$  of reduction rules:

$$\begin{array}{c} \text{app}(v(\text{abs}(T)), v(U)) \rightsquigarrow j(T)\{* := U\} \quad \frac{j(T) \rightsquigarrow j(T')}{v(\text{abs}(T)) \rightsquigarrow v(\text{abs}(T'))} \\[10pt] \frac{T \rightsquigarrow T'}{\text{app}(T, U) \rightsquigarrow \text{app}(T', U)} \quad \frac{U \rightsquigarrow U'}{\text{app}(T, U) \rightsquigarrow \text{app}(T, U')}. \end{array}$$

To conclude, the operational signature is  $(\mathbb{S}, \mathbb{S}, B \cdot \Theta', \mathfrak{R})$ .

### 6.4.2 Call-by-value lambda calculus with big-step operational semantics

We give a signature for the operational monad  $((B, \text{Id}), \text{LC}_v)$  of the call-by-value lambda calculus with big-step operational semantics (Example 200). The first state functor  $B$  is specified by  $\mathbb{S}$  as in Section 6.4.1, whereas the second one  $\text{Id}$  is specified by the algebraic 1-signature for functors  $\text{Id}_{\text{Set}}$ : a model  $M$  is an endofunctor with a natural transformation  $w : \text{Id}_{\text{Set}} \rightarrow M$

The underlying signature for monads is the same as the one of the call-by-value lambda calculus (Section 6.4.1). Now, we give the family  $\mathfrak{R}$  of reduction rules, using the same notations as in Section 6.4.1:

$$\frac{}{v(T) \rightsquigarrow w(T)} \quad \frac{T \rightsquigarrow w(\text{abs}(T')) \quad U \rightsquigarrow w(U') \quad j(T')\{* := U'\} \rightsquigarrow V}{\text{app}(T, U) \rightsquigarrow V}.$$

To conclude, the operational signature is  $(\mathbb{S}, \text{Id}_{\text{Set}}, B \cdot \Theta', \mathfrak{R})$ , where  $\mathbb{S}$  is defined in Section 6.4.1.

### 6.4.3 $\pi$ -calculus

We give an operational signature for the operational monad  $((T, T), \text{Id})$  of the  $\pi$ -calculus described in Example 201. In Example 236, we gave an algebraic signature  $\mathbb{S}$  for specifying the endofunctor  $T$ . A model  $M$  of  $\mathbb{S}$  comes equipped with the following natural transformations:

- $\text{par} : M \times M \rightarrow M$  mapping  $(P, Q)$  to  $P|Q$ ;
- $\text{out} : \text{Id} \times \text{Id} \times M \rightarrow M$  mapping  $(a, b, P)$  to  $\bar{a}\langle b \rangle.P$ ;

- $\text{in} : \text{Id} \times M' \rightarrow M$  mapping  $(a, P) \in X \times M(X + \{*\})$  to  $a(*) \cdot P$ ;
- $\text{abs} : M' \rightarrow M$  mapping  $P \in M(X + \{*\})$  to  $\nu * \cdot P$ .

Note that the monad  $\text{Id}_{\text{Set}}$  is the initial model of the empty signature  $0$ .

The family  $\mathfrak{R}$  of reduction rules over  $0$  consists of the following:

Original rule	As a $\mathbb{S}$ -reduction rule
$\overline{a}\langle b \rangle \cdot P \mid a(c) \cdot Q \rightsquigarrow P \mid (Q\{c := b\})$	$\text{par}(\text{out}(a, b, j_1(P)), \text{in}(a, j_1(Q))) \rightsquigarrow \text{par}(j_2(P), j_2(T[\text{id}, \underline{b}](Q)))$
$\frac{P \rightsquigarrow Q}{P \mid R \rightsquigarrow Q \mid R}$	$\frac{P \rightsquigarrow Q}{\text{par}(P, R) \rightsquigarrow \text{par}(Q, R)}$
$\frac{P \rightsquigarrow Q}{\nu a \cdot P \rightsquigarrow \nu a \cdot Q}$	$\frac{P \rightsquigarrow Q}{\text{abs}(P) \rightsquigarrow \text{abs}(Q)}$

where:

- $j_i : T \rightarrow M_i$  is the initial  $\mathbb{S}$ -model morphism to the underlying state functor  $M_i$  of a  $\mathbb{S}$ -reduction  $\Sigma$ -monad, for  $i \in \{1, 2\}$ ;
- $T[\text{id}, \underline{b}]$  is the function  $T[\text{id}, \underline{b}] : T(R(X) + 1) \rightarrow T(R(X))$ , for  $\underline{b} : 1 \rightarrow R(X)$  the function selecting  $b \in R(X)$ .

To conclude, the operational signature is  $(\mathbb{S}, \text{Id}_{\text{Set}}, 0, \mathfrak{R})$ .



# CONCLUSION

---

In this thesis, we have studied different notions of signatures that are all particular instances of the general definition that we give in Chapter 2.

In Chapters 3 and 4, we have presented notions of signatures for monads and their models. More precisely, in Chapter 3, 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 3.7 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 41).

In Chapter 4, 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  $\Sigma$  (that is, a signature in the sense of Chapter 3) and a set of  $\Sigma$ -equations that must be satisfied.

Finally, in Chapter 5, we have introduced the notions of reduction monad and reduction signature: they are meant to model syntax 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. Chapter 6 generalizes the notion of reduction monad to that of operational monad, and adapts the statement of Chapter 5 accordingly. These two chapters is a first step towards a theory for the al-

---

gebraic specification of programming languages and their semantics. In future work, we aim at generalizing 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 146).

We anticipate that our work extends to simply-typed languages, by changing the base category  $\mathbf{Set}$  to a presheaf category  $\mathbf{Set}^T$ , 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.LG].
- [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](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/CB09780511525865.
- [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] Randal 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](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, LFMTTP 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 Declarative 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.





---

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

**Mot clés :** monades, syntaxe, sémantique, signature

**Résumé :** Cette thèse traite de la spécification et la construction de la syntaxe et sémantique opérationnelle d'un langage de programmation. Nous travaillons avec une notion générale de "signature" pour spécifier des objets d'une catégorie donnée comme des objets initiaux dans une catégorie appropriée de modèles. Cette caractérisation, dans l'esprit de la sémantique initiale, donne une justification du principe de récursion.

Les langages avec liaisons, telles que le lambda calcul pur, sont des monades sur la catégorie des ensembles spécifiées 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.

Finalement, nous abordons la spécification de la sémantique opérationnelle d'un langage de programmation tel que le lambda calcul avec  $\beta$ -réduction. Nous introduisons à cette fin la notion de monade de réduction et leur signatures, puis les généralisons pour aboutir à la notion de monade opérationnelle.

---

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

**Keywords:** monads, syntax, semantic, signature

**Abstract:** This thesis deals with the specification and construction of syntax and operational semantics of a programming language. We work with a general notion of "signature" for specifying objects of a given category as initial objects 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 monads on the category of sets specified 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.

Finally, we tackle the specification of the operational semantics of a programming language such as lambda calculus with  $\beta$ -reduction. To this end, we introduce the notion of reduction monad and their signatures, then we generalize them to get the notion of operational monad.