Reduction Monads and Their Signatures

BENEDIKT AHRENS, University of Birmingham, United Kingdom
ANDRÉ HIRSCHOWITZ, Université Côte d'Azur, CNRS, France
AMBROISE LAFONT, IMT Atlantique, France
MARCO MAGGESI, Università degli Studi di Firenze, Italy

In this work, we study *reduction monads*, which are essentially the same as monads relative to the free functor from sets into multigraphs. Reduction monads account for two aspects of the lambda calculus: on the one hand, in the monadic viewpoint, the lambda calculus is an object equipped with a well-behaved substitution; on the other hand, in the graphical viewpoint, it is an oriented multigraph whose vertices are terms and whose edges witness the reductions between two terms.

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

Our main result identifies a class of reduction signatures which specify a reduction monad in the above sense. We show in the examples that our approach covers several standard variants of the lambda calculus.

CCS Concepts: • **Theory of computation → Equational logic and rewriting**; **Categorical semantics**.

Additional Key Words and Phrases: Initial semantics, Higher-order languages, Reduction systems, Lambda calculus, Monads

ACM Reference Format:
Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. 2020. Reduction Monads and Their Signatures. *Proc. ACM Program. Lang.* 4, POPL, Article 31 (January 2020), 29 pages. https://doi.org/10.1145/3371099

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 multigraph (or a relation, or a preorder, or even a category), while others view it as a monad (on the category of sets). The first account incorporates the β-reduction, while the second addresses substitution but incorporates only the β-equality. Merging these two perspectives led Lüth and Ghani [Lüth and Ghani 1997] to consider monads on the category of preordered sets, and Ahrens [Ahrens 2016] to consider monads relative to the free functor from sets into preorders. In the present work, we propose a variant of their approaches. Here we call

Authors' addresses: Benedikt Ahrens, School of Computer Science, University of Birmingham, United Kingdom, B.Ahrens@cs.bham.ac.uk; André Hirschowitz, LJAD, Université Côte d'Azur, CNRS, France, ah@unice.fr; Ambroise Lafont, Gallinette, Inria, IMT Atlantique, France, ambroise.lafont@inria.fr; Marco Maggesi, DiMaI, Università degli Studi di Firenze, Italy, marco.maggesi@unifi.it.

2475-1421/2020/1-ART31
https://doi.org/10.1145/3371099

reduction monad a monad relative to the discrete injection of sets in multigraphs, 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*. Our main result (Theorem 5.10) provides a natural criterion for a reduction signature to be effective. Our main examples are variants of the lambda calculus (Section 3.2 and Example 5.13).

In summary, our notions of reduction monad and (effective) reduction signature

- allow "dynamic" reduction systems (as opposed to "static" ones, where reduction is understood as an equivalence relation);
- cover (some) higher-order languages;
- allow reduction systems that are not fully congruent, e.g., weak head reduction in lambda calculus; and
- allow reduction systems that are proof-relevant.

While the first three features listed above are definitely positive, we feel the need to discuss the advantages of the fourth one. Basically, one can model reductions either via preorders or relations, or via multigraphs, and we choose multigraphs. On the one hand, multigraphs are appropriate to avoid problems with the well-know phenomena of *syntactic accidents* (see [Bezem et al. 2003, Example 2.2.9 and Section 8.2] for a classic reference). On the other hand, the approaches via preorders or relations can be easily recovered as special cases. Indeed, note that the category of preordered sets (as considered, e.g., in [Ahrens 2016]) is a reflective subcategory of the category of relations (as considered, e.g., in [Plotkin 2004]), which is again a reflective subcategory of the category of multigraphs. Then, the corresponding adjunctions make it easy to customize our formalism to deal with preordered sets or relations instead of multigraphs.

1.1 Related Work

The search for a mathematical notion of programming language goes back at least to Turi and Plotkin [Turi and Plotkin 1997] who coined the name "Mathematical Operational Semantics" and explained how known classes of well-behaved rules for structural operational semantics (SOS) [Plotkin 2004], such as GSOS [Bloom et al. 1988], can be categorically understood via distributive laws and bialgebras (see also [Klin 2011]). Their initial framework did not cover variable binding, and several authors have proposed variants which do [Fiore and Staton 2006; Fiore and Turi 2001; Staton 2008], treating examples like the π-calculus. More recently T. Hirschowitz [Hirschowitz 2019] proposed an alternative categorical approach to SOS allowing variable binding. However, none of these approaches covers higher-order languages like the lambda calculus.

Meanwhile, similar research has been done from the point of view of "Rewriting Systems" or "Equational Systems". (A rewriting system is intended to specify a fully congruent relation, while an equational system is intended to specify a fully congruent equivalence relation.) In particular:

- Lüth and Ghani [Lüth and Ghani 1997] interpreted a class of rewriting systems (without considering bindings) in monads in the category of preordered sets.
- Hamana [Hamana 2003] studied his "binding term rewriting system (BTRS)" via preorder-valued functors.
- Building upon [Fiore et al. 1999], Fiore and Hur [Fiore and Hur 2007] have studied a large class of "term equational systems", covering in particular the lambda calculus (see also [Fiore

and Mahmoud 2010]). In our recent work [Ahrens et al. 2019], we have proposed a variant of their work in terms of signatures for monads; this is reviewed below in Section 2.3 since it is the starting building block for the present work.

- T. Hirschowitz [Hirschowitz 2013] approached higher-order rewriting systems via Cartesian closed 2-categories.
- Ahrens [Ahrens 2016] provided a signature for the lambda calculus viewed as a monad relative to the inclusion of sets into preorders mapping a set to its discrete preorder.

This last work was a starting point of the present one and we stress two main differences. The first difference is technical but important: our signatures are built from modules with values in Set (not in the category of multigraphs) while in [Ahrens 2016] signatures involve modules with values in Preorder (not in Set). This latter choice is responsible for the fact that full congruence of reductions is hard-coded in [Ahrens 2016]. A second difference is that the format for reduction rules considered in [Ahrens 2016] does not allow rules with hypotheses, as present for instance in the "non-full" congruence rules which specify the head-β-reduction (see the table at the end of Section 4.6).

1.2 Plan of the Paper

In Section 2, we review those notions from our previous work [Ahrens et al. 2018, 2019] that we build on in the present work. In Section 3, we define the category of reduction monads. In Section 4, we give our notion of reduction rules. Building on this notion, we define signatures for reduction monads—*reduction signatures*—in Section 5. There, we also state our main result (Theorem 5.10), which provides a simple criterion for such a signature to be effective. Section 6 is devoted to the proof of this result. Then, in Section 7, we give a detailed example of a reduction signature specifying Kesner's lambda calculus with explicit substitutions [Kesner 2009]. Finally, in Section 8, we explain the recursion principle which, as usual, can be derived from initiality in our categories of models. We then use this recursion principle to specify a translation from Kesner's lambda calculus with explicit substitutions to the pure lambda calculus.

2 C-MODULES AND SIGNATURES FOR MONADS

The present work is devoted to reduction monads and their signatures. The first building block of a reduction monad is a monad, and accordingly, the first building block of one of our signatures for reduction monads will be a signature for monads. In the present section, we review monads and their signatures as introduced in [Ahrens et al. 2019]. In that paper, one can also find a review of related work on the generation of monads.

A signature prescribes a monad by specifying:

(1) a family of *constructions*;
(2) a family of *equations* among these constructions.

As an example, in the case of the lambda calculus, the constructions are application and abstraction (in the following denoted app and abs). Equations that can be considered over those constructions are the β- and η-equations. Notice, however, that later in this paper we will opt for considering β and η as **reductions** (as opposed to equivalences, or equations).

In Section 2.1 we review the notions of monads and modules (on sets). In Section 2.2 we introduce the notion of C-module for a category C. This notion is used in our definition of signature. In our formalism, constructions are specified by *1-signatures* (reviewed in Section 2.3), and equalities between constructions specified by 1-signatures are specified by *equations* (reviewed in Section 2.4). A 2-signature is a pair consisting of a 1-signature and a family of equations over that signature. We refer to both 1- and 2-signatures simply as **signatures for monads** when the distinction between the two notions is not relevant.

2.1 Monads and Modules

In this section we review the notions of monad and module to the extent required in this work. We write $g \circ f$ for the composition of morphisms $f : A \to B$ and $g : B \to C$ in any category. Similarly, functor composition is written $G \cdot F$. For the purpose of this work, we restrict ourselves to the category Mon of monads over the category Set of sets.

A **monad** consists of a function $R : \mathsf{Set} \to \mathsf{Set}$, a family of functions $\eta_X : X \to R(X)$, and a family of functions $\_\{\_\} : RX \times (X \to RY) \to RY$ called *substitution*, satisfying the equations $t\{\eta_X\} = t$, $\eta_X(x)\{f\} = f(x)$, $t\{f\}\{g\} = t\{x \mapsto f(x)\{g\}\}$ for $t \in RX$, $f : X \to RY$ and $g : Y \to RZ$. Sometimes, we identify an element $x \in X$ with the corresponding term $\eta_X(x) \in R(X)$ when no confusion can arise. A **module** M **over a monad** R is a function $\mathsf{Set} \to \mathsf{Set}$ with a family of functions $\_\{\_\} : MX \times (X \to RY) \to MY$ called *substitution*, satisfying the equations $t\{\eta_X\} = t$ and $t\{f\}\{g\} = t\{x \mapsto f(x)\{g\}\}$ for $t \in MX$, $f : X \to RY$ and $g : Y \to RZ$. Note that we use the same notation for the substitution of monads and of modules; in the last equation, the inner substitution on the right hand side is the one from the monad R. A monad R (resp. a module M over a monad R) gives rise to a functorial action $RX \times (X \to Y) \to RY$ (resp. $MX \times (X \to Y) \to MY$) by $(t, g) \mapsto t\{\eta_Y \circ g\}$. A **morphism** $R \to S$ **of monads** is a family of functions $\alpha_X : RX \to SX$ commuting with η and $\_\{\_\}$ of R and S, in the sense that $\alpha(\eta^R(x)) = \eta^S(x)$ and $\alpha(t\{f\}) = \alpha(t)\{\alpha \circ f\}$ for $x \in X$, $t \in RX$ and $f : X \to RY$. With the obvious composition operation, this yields the **category** Mon **of monads on sets**. A **morphism of modules** $M \to N$ **over a monad** R is a family of functions $\beta_X : MX \to NX$ commuting with $\_\{\_\}$ of M and N in the sense that $\beta(t\{f\}) = \beta(t)\{f\}$ for $t \in M(X)$ and $f : X \to RY$. Monad and module morphisms are (in particular) natural transformations.

If M is a module over S and $\alpha : R \to S$ is a monad morphism, then we denote $\alpha^* M$ the R-module with the underlying function $X \mapsto M(X)$ on sets, and with substitution induced by that of M by $(t, f) \mapsto t\{\alpha_Y \circ f\}$; this is called the **reindexing of** M **along** α.

We denote by $\mathsf{Mod}(R)$ the **category of modules over a fixed monad** R, and by $\int_R \mathsf{Mod}(R)$ the **total category of modules**, fibered over the category Mon of monads [Ahrens et al. 2018, Proposition 7]: objects of $\int_R \mathsf{Mod}(R)$ are pairs of a monad and a module over it, and a morphism from (R, M) to (S, N) is a monad morphism $\alpha : R \to S$ together with a R-module morphism between M and $\alpha^* N$.

A prominent example of monad, used heavily in the rest of this paper, is the (untyped, syntactic) lambda calculus: given a set X (to be understood as a collection of free variables), $\mathsf{LC}(X)$ is the set of lambda terms with free variables in X modulo α-equivalence, and the usual parallel, capture-avoiding substitution endows LC with a monadic structure (see [Altenkirch and Reus 1999]).

Modules and morphisms of modules are used to specify constructions and equations within a signature. To this end, we recall some basic module constructions that are used compositionally.

First of all, a simple observation is that every monad R is (trivially) a module over itself. We denote $\Theta(R)$ (sometimes just R) the monad R regarded as an R-module. The assignment $R \mapsto \Theta(R)$ is the object map of the **tautological** section $\Theta : \mathsf{Mon} \to \int_R \mathsf{Mod}(R)$.

Next, in the category $\mathsf{Mod}(R)$ of modules over a fixed monad R, we have arbitrary limits and colimits, which are computed pointwise. In particular, we have the initial and final modules and, for two R-modules M and N, the binary product $M \times N$ and the binary coproduct $M + N$.

Given an R-module M, we have the **derived** R-module M' defined as follows. We denote by $X + \{*\}$ the disjoint union of a set X with a one-element set. Then, we set $M'(X) := M(X + \{*\})$.

The R-module structure on M induces naturally an R-module structure on M'. This derivation yields an endofunctor on the category of modules $\mathsf{Mod}(R)$ for any monad R. It can be iterated; we call $M^{(k)}$ the k-th derivative of M.

2.2 C-Modules

In this section, C is a category equipped with a functor $U : \mathsf{C} \to \mathsf{Mon}$. We introduce the notion of C-module, generalizing Σ-modules of [Ahrens et al. 2019].

Definition 2.1. A C-**module** is a functor T from the category C to the category $\int_R \mathsf{Mod}(R)$ commuting with the functors to the category Mon of monads,

$$\begin{tikzcd} \mathsf{C} \arrow[rr, "T"] \arrow[dr, "U"'] & & \int_R \mathsf{Mod}(R) \arrow[dl] \\ & \mathsf{Mon} & \end{tikzcd}$$

More concretely, a C-module T maps an object c of C to a module $T(c)$ over $U(c)$ and a morphism $f : c \to c'$ to a $U(c)$-module morphism $T(f) : T(c) \to U(f)^*T(c')$. In this paper, U will arise as a forgetful functor to the category Mon of monads and will be tacitly omitted.

Definition 2.2. The **tautological** C-**module** Θ maps an object c of C to the module $\Theta(U(c))$.

Definition 2.3. Let A be a C-module. The **derivative of** A is the C-module A' mapping an object c of C to the $U(c)$-module $A(c)'$.

We typically apply (iteratively) the derivation construction to the tautological C-module Θ: The C-module Θ' maps an object c of C to the $U(c)$-module $U(c)'$, and the C-module $\Theta^{(n)}$ maps c to the $U(c)$-module $U(c)^{(n)}$.

We introduce the *category of* C*-modules*. Importantly, as we will see later, our *term-pairs* are built from pairs of parallel morphisms of C-modules for a suitable category C.

Definition 2.4. Let S and T be C-modules. A **morphism of** C-**modules** from S to T is a natural transformation from S to T which becomes the identity of U when postcomposed with the forgetful functor from the category of modules $\int_R \mathsf{Mod}(R)$ to the category of monads.

Proposition 2.5. *As defined above,* C*-modules and their morphisms, with the obvious composition and identity, form a category.*

Definition 2.6. Let A and B be C-modules, and let $f : A \to B$ be a morphism of C-modules. The **derivative of** f is the C-module morphism from A' to B' mapping an object c of C to the $U(c)$-module morphism f_c'.

2.3 1-Signatures

In this section we review the 1-signatures introduced in [Ahrens et al. 2018].

Definition 2.7 (1-Signatures and their models, [Ahrens et al. 2018, Defs. 14 and 27]). A **1-signature** is a Mon-module, hence it is just a section to the forgetful functor $\pi : \int_R \mathsf{Mod}(R) \to \mathsf{Mon}$.

Let $\Sigma : \mathsf{Mon} \to \int_R \mathsf{Mod}(R)$ be a 1-signature. A **model of** Σ is a pair of a monad R and a module morphism $\Sigma(R) \to R$, called an **action of** Σ **in** R. A **morphism from** (R, r) **to** (S, s) is a morphism of monads $m : R \longrightarrow S$ making the following diagram of R-modules commutes:

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

Recall that the module m^*S is the reindexing of S along m in the fibration $\int_R \mathsf{Mod}(R) \longrightarrow \mathsf{Mon}$. We call Mon^{Σ} the induced category of models of Σ.

The desired property for a signature is "effectivity":

Definition 2.8. Given a 1-signature Σ, the initial object in Mon^{Σ}, if it exists, is denoted by $\hat{\Sigma}$. In this case, the 1-signature Σ is called **effective**.

Example 2.9. Consider the 1-signature Σ_{LC} given on objects by $R \mapsto R \times R + R'$. A model of this signature is a triple $(R, \mathsf{app}, \mathsf{abs})$ with $\mathsf{app} : R \times R \longrightarrow R$ and $\mathsf{abs} : R' \longrightarrow R$. The initial model is the triple $(\mathsf{LC}, \mathsf{app}, \mathsf{abs})$ of the monad LC of lambda terms, with application and abstraction as module morphisms. Note that α-equivalence corresponds to syntactic equality of lambda terms.

Example 2.10 (Non-effective 1-signature). The 1-signature mapping a monad R to the R-module $\mathcal{P} \cdot R$, where $\mathcal{P}$ is the powerset functor, is not effective, for cardinality reasons.

Remark 2.11. Let C be a category equipped with a functor $U : \mathsf{C} \longrightarrow \mathsf{Mon}$. Any 1-signature Ψ (i.e. any functor from Mon to $\int_R \mathsf{Mod}(R)$ commuting with the forgetful functors to Mon) induces a C-module still denoted Ψ, by precomposition with U.

2.4 Equations and 2-Signatures

In this section we review the 2-signatures introduced in [Ahrens et al. 2019].

Let Σ be a 1-signature. In the following, we use the term "Σ-module" for Mon^{Σ}-modules.

Definition 2.12 (Σ-equation, [Ahrens et al. 2019, Def. 8]). We define a **Σ-equation** to be a pair of parallel morphisms of Σ-modules.

Definition 2.13 (2-signature, [Ahrens et al. 2019, Def. 12]). A **2-signature** is a pair (Σ, E) of a 1-signature Σ and a family E of Σ-equations.

Definition 2.14 (Model of a 2-signature, [Ahrens et al. 2019, Def. 17]). We say that a model M of a 1-signature Σ **satisfies the Σ-equation** (e_1, e_2) if $e_1(M) = e_2(M)$. If E is a family of Σ-equations, we say that a model M of Σ **satisfies** E if M satisfies each Σ-equation in E.

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

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

As for 1-signatures, we define a notion of *effectivity*:

Definition 2.15. A 2-signature Υ is said to be **effective** if its category of models Mon^{Υ} has an initial object, denoted $\hat{\Upsilon}$.

As an example, consider the signature Σ_{Λ} obtained from Σ_{LC} by imposing the β- and η-equalities. Then, the monad Λ of lambda calculus modulo $\beta\eta$-equality is the initial object of the category of models $\mathsf{Mon}^{\Sigma_{\Lambda}}$ [Hirschowitz and Maggesi 2010, Theorem 3].

As 1-signatures are particular 2-signatures with an empty set of equations, Example 2.10 yields a non-effective 2-signature. Another example is a 2-signature containing the equation $\mathsf{inl}, \mathsf{inr} : \Theta \rightrightarrows \Theta + \Theta$ given by the left and right inclusions. An important class of effective signatures for monads is the one of *algebraic* (2-)signatures [Ahrens et al. 2019, Theorem 32], which covers both Σ_{LC} and Σ_{Λ} as particular cases.

3 REDUCTION MONADS

Here below, we define *the category of reduction monads* in Section 3.1. We also consider some examples of reduction monads, in Section 3.2.

3.1 The Category of Reduction Monads

Definition 3.1. A **reduction monad** R is given by:

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

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

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

Remark 3.2. 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 fiber $s \blacktriangleright t$ is not necessarily a subsingleton.

Remark 3.3. Let R be a reduction monad, X and Y two sets, $f : X \longrightarrow R(Y)$ a map, and u and v two elements of $R(X)$ related by $m : u \blacktriangleright v$. The module structure on $\mathsf{Red}(R)$ yields a reduction denoted $m\{f\}$ between $u\{f\}$ and $v\{f\}$.

However, if we are given two maps 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 3.2.

Definition 3.4. A **morphism of reduction monads** from R to S is given by a pair (f, α) of

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

satisfying the following two conditions:

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

$$\begin{tikzcd}
\mathsf{Red}(R) \arrow[r, "\alpha"] \arrow[d, "\mathsf{red}_R"'] & \mathsf{Red}(S) \arrow[d, "\mathsf{red}_S"] \\
R \times R \arrow[r, "f \times f"'] & S \times S
\end{tikzcd}$$

commutes in the category of functors and natural transformations.

In Section 8 we specify morphisms of reduction monads via a recursion principle.

Intuitively, a morphism (f, α) as above maps terms of R to terms of S via f, and reductions of R to reductions of S via α. Condition 3 states compatibility of the map of reductions with substitution: $\alpha(m\{g\}) = \alpha(m)\{f_Y \circ g\}$ for any reduction $m : u \blacktriangleright v$ and any map $g : X \longrightarrow R(Y)$. Condition 4 states preservation of source and target by the map of reductions: a reduction $m : u \blacktriangleright v$ between elements of $R(X)$ is mapped by α to a reduction $\alpha(m) : f_X(u) \blacktriangleright f_X(v)$.

Proposition 3.5 (Category of reduction monads). *Reduction monads and their morphisms, with the obvious composition and identity, form a category* RedMon*, equipped with a forgetful functor to the category of monads.*

It turns out that reduction monads are the same as monads relative to the free functor from sets to multigraphs (for the definition of relative monads, see [Altenkirch et al. 2015, Definition 2.1]):

Theorem 3.6. *The category of reduction monads is isomorphic to the category of monads relative to the functor mapping a set to its discrete multigraph.*

Proof. This is obvious after unfolding the definitions. □

3.2 Examples of Reduction Monads

We are interested in reduction monads with underlying monad LC, the monad of syntactic lambda terms specified in Example 2.9. We start with a detailed simple example. The other ones (Examples 3.8, 3.9, and 3.10) are more informal. Reduction signatures specifying them will be given later in Example 5.13.

*Example 3.7 (Lambda calculus with **top-β-reduction**).* Consider the reduction monad $\mathsf{LC}_{\text{top-}\beta}$ given as follows:

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

*Example 3.8 (Lambda calculus with **weak head β-reduction**).* We introduce the reduction monad $\mathsf{LC}_{wh\beta}$. A reduction $m \in \mathsf{Red}(\mathsf{LC}_{wh\beta})(X)$ in the reduction monad $\mathsf{LC}_{wh\beta}$ is a leftmost β-reduction in a chain of applications. Using the standard syntax of lambda calculus, $(\lambda x.t)\, u_1\, u_2\, \ldots\, u_n$ reduces to $t\{x := u_1\}\, u_2\, \ldots\, u_n$.

*Example 3.9 (Lambda calculus with **congruent β-reduction**).* We introduce the reduction monad LC_β. A reduction $m \in \mathsf{Red}(\mathsf{LC}_\beta)(X)$ in the reduction monad LC_β is "one step" of β-reduction, anywhere in the source term.

*Example 3.10 (Lambda calculus with **parallel β-reduction**).* A reduction $m \in \mathsf{Red}(\mathsf{LC}_{\beta\|})(X)$ in the reduction monad $\mathsf{LC}_{\beta\|}$ is the simultaneous β-reduction of a "parallel" set of redexes in the source term. "Parallel" here means that the subtrees of the contracted redexes are disjoint.

Next, we consider the *closure under identity and composition of reductions.*

Definition 3.11. Given a reduction monad R, we define the reduction monad R^* as follows:

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

$$\begin{tikzcd}
& \mathsf{Red}(R) \arrow[ld, "\mathsf{source}_R"'] \arrow[rd, "\mathsf{target}_R"] & & \mathsf{Red}(R) \arrow[ld, "\mathsf{source}_R"'] \arrow[rd, "\mathsf{target}_R"] & & \cdots \arrow[ld, "\mathsf{source}_R"'] \\
R & & R & & R &
\end{tikzcd}$$

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

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

Example 3.12 (The reduction monad of the lambda calculus). The **reduction monad of the lambda calculus** is defined to be the reduction monad LC^*_β.

In Section 5 we introduce signatures that allow for the specification of reduction monads.

4 REDUCTION RULES

In this section, we define an abstract notion of reduction rule over a signature for monads Σ (Section 4.2). We first focus, in Section 4.1, on the example of the congruence rule for the application construction in the signature Σ_{LC} (cf. Example 2.9) for the monad of the lambda calculus, in order to motivate the definitions. The purpose of a reduction rule over Σ is to be "modeled" in a reduction monad equipped with an action of Σ (this is what we will call a *reduction* Σ*-model* in Section 4.3). We make this notion of model precise in Section 4.4, as an action of the reduction rule in the reduction Σ-model. Finally, we give a protocol for specifying reduction rules in Section 4.5 that we apply in Section 4.6 to some examples.

Notation 4.1. In the following, we use the shorter terminology of Σ-modules for Mon^Σ-modules, when Σ is any signature for monads (either a 1-signature or a 2-signature).

4.1 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 [Selinger 2008], as follows:

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

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 $\mathrm{app} : R \times R \longrightarrow R$, 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 $c_R : \mathcal{V}(R) \longrightarrow R$ maps a set X and a quadruple (T, T', U, U') to the pair $(\mathrm{app}(T, U), \mathrm{app}(T', U'))$. These assignments are actually functorial in R, and abstracting over R yields our notion of *term-pair* over the Σ-module $\mathcal{V}$, as morphisms from $\mathcal{V}$ to $\Theta \times \Theta$, where Σ is any signature including a single first-order binary operation app (for example, Σ_{LC}). The three term-pairs, one for each hypothesis and one for the conclusion, define the desired reduction rule.

Now, we explain in which sense such a rule can be modeled 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 $\mathrm{app\text{-}cong}(s, t) : \mathrm{app}(T, U) \blacktriangleright \mathrm{app}(T', U')$. Of course, this only makes sense if the monad $\underline{R}$ underlying the reduction monad is equipped with an application operation, that is, with an operation of Σ_{app}. We will call such a structure a *reduction* Σ_{app}*-model* (see Section 4.3).

4.2 Definition of Reduction Rules

In this subsection, Σ is a signature for monads. We present our notion of *reduction rule over* Σ, from which we build *reduction signatures* in Section 5.

We begin with the definition of *term-pair*, alluded to already in Section 4.1:

Definition 4.2. Given a Σ-module $\mathcal{V}$, a **term-pair from $\mathcal{V}$** is a pair (n, p) of a natural number n and a morphism of Σ-modules $p : \mathcal{V} \longrightarrow \Theta^{(n)} \times \Theta^{(n)}$.

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

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

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

Example 4.4 (term-pair of the conclusion of the congruence for application). The term-pair corresponding to the conclusion $\mathrm{app}(T, U) \rightsquigarrow \mathrm{app}(T', U')$ of congruence for application (Section 4.1) is given by $(0, c)$, on the Σ_{LC}-module Θ^4. Here, we have

$$c : \mathcal{V} \longrightarrow \Theta \times \Theta$$
$$c_{R,X}(T, T', U, U') := \Big(\mathrm{app}_{R,X}(T, U), \mathrm{app}_{R,X}(T', U')\Big)$$

More schematically:

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

We now give our definition of *reduction rule*, making precise the intuition developed in Section 4.1.

Definition 4.5. A **reduction rule** $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$ **over** Σ is given by:

- Metavariables: a Σ-module $\mathcal{V}$ of metavariables, that we sometimes denote by $\mathrm{MVar}_{\mathcal{A}}$;
- 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 4.6 (Reduction rule for congruence of application). The reduction rule $\mathcal{A}_{\text{app-cong}}$ for congruence of application (Section 4.1) is defined as follows:

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

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

- Conclusion: Given by the term-pair $(0, c)$ of Example 4.4.

More examples of reduction rules are given in Section 4.6.

4.3 Reduction Σ-Models

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

Definition 4.7. Let Σ be a signature for monads. The **category** RedMon^{Σ} **of reduction** Σ**-models** is defined as the following pullback:

$$\begin{tikzcd}
\mathsf{RedMon}^{\Sigma} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathsf{RedMon} \arrow[d] \\
\mathsf{Mon}^{\Sigma} \arrow[r] & \mathsf{Mon}
\end{tikzcd}$$

More concretely,

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

4.4 Action of a Reduction Rule

Let Σ be a signature for monads. In this section, we introduce the notion of *action of a reduction rule over* Σ *in a reduction* Σ*-model*. 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 *fibered* product.

Definition 4.8. Let (n, p) be a term-pair from a Σ-module $\mathcal{V}$, and R be a reduction Σ-model. We denote by $p^*(\mathsf{Red}(R)^{(n)})$ the pullback of $\mathsf{red}_R^{(n)} : \mathsf{Red}(R)^{(n)} \to R^{(n)} \times R^{(n)}$ along $p_R : \mathcal{V}(R) \to R^{(n)} \times R^{(n)}$:

$$\begin{tikzcd}
p^*(\mathsf{Red}(R)^{(n)}) \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & \mathsf{Red}(R)^{(n)} \arrow[d, "\mathsf{red}_R^{(n)}"] \\
\mathcal{V}(R) \arrow[r, "p_R"'] & R^{(n)} \times R^{(n)}
\end{tikzcd}$$

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

Definition 4.9. Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (n, c))$ be a reduction rule, and R be a reduction Σ-model. The R**-module** $\mathsf{Hyp}_{\mathcal{A}}(R)$ **of hypotheses of** $\mathcal{A}$ is $\prod_{i \in I}^{\mathcal{V}(R)} h_i^* \mathsf{Red}(R)^{(n_i)}$, i.e., the fiber product of all the R-modules $h_i^* \mathsf{Red}(R)^{(n_i)}$ along their projection to $\mathcal{V}(R)$. It thus comes with a projection $\mathsf{hyp}_{\mathcal{A}}(R) : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathcal{V}(R)$

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

Example 4.10. Let R be a reduction Σ_{LC}-model. The R-module of conclusion of the congruence reduction rule for application (Example 4.6, see also Section 4.1) maps a set X to the set of quintuples (T, T', U, U', m) where $(T, T', U, U') \in R^4(X)$ and m is a reduction $m : \mathsf{app}(T, U) \blacktriangleright \mathsf{app}(T', U')$. The R-module of hypotheses of this reduction rule maps a set X to the set of sextuples (T, T', U, U', m, n) where $(T, T', U, U') \in R^4(X)$, $m : T \blacktriangleright T'$, and $n : U \blacktriangleright U'$.

Definition 4.11. Let $\mathcal{A}$ be a reduction rule over Σ. An **action of** $\mathcal{A}$ **in a reduction** Σ**-model** R is a morphism between $\mathsf{hyp}_{\mathcal{A}}(R)$ and $\mathsf{con}_{\mathcal{A}}(R)$ in the slice category $\mathsf{Mod}(R)/\mathsf{MVar}_{\mathcal{A}}(R)$, that is, a morphism of R-modules

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

making the following diagram commute:

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \arrow[rr, "\tau"] \arrow[dr] & & \mathsf{Con}_{\mathcal{A}}(R) \arrow[dl] \\
& \mathsf{MVar}_{\mathcal{A}}(R) &
\end{tikzcd} \tag{1}$$

Example 4.12 (Action of the congruence rule for application). Consider the reduction rule of the congruence for application of Example 4.6. Let R be a reduction Σ_{LC}-model R. An action τ in R is an R-module morphism which, for each set X, maps a sextuple (T, T', U, U', r, s) with $T, T', U, U' \in R(X)$, $r : T \blacktriangleright T'$, and $s : U \blacktriangleright U'$ to a quintuple (T, T', U, U', m) with $m : \mathsf{app}(T, U) \blacktriangleright \mathsf{app}(T', U')$. The fact that τ is fully determined by its last component allows us to present it as in triangle (1).

Alternatively (as justified formally by Lemma 6.3), an action is a morphism mapping the same sextuple to a reduction $m : \mathsf{app}(T, U) \blacktriangleright \mathsf{app}(T', U')$.

4.5 Protocol for Specifying Reduction Rules

In Section 4.6, we adopt the following schematic presentation of a reduction rule over a signature Σ:

$$\frac{s_1 \rightsquigarrow t_1 \qquad \dots \qquad s_n \rightsquigarrow t_n}{s_0 \rightsquigarrow t_0}$$

where s_i and t_i are expressions depending on "metavariables" $X_1, \dots, X_q$. Each pair (s_i, t_i) defines a term-pair as follows:

$$p_i : M_1 \times \cdots \times M_q \longrightarrow \Theta^{(m_i)} \times \Theta^{(m_i)}$$
$$p_{i,R,X}(T_1, \dots, T_q) := (s_i[\vec{X}/\vec{T}], t_i[\vec{X}/\vec{T}]) \tag{2}$$

where $s_i[\vec{X}/\vec{T}]$ is s_i where each metavariable X_i has been replaced with T_i. The Σ-modules $M_1, \dots, M_q$, and the natural numbers $m_0, \dots, m_n$ are inferred for Equation (2) to be well defined for all $i \in \{0, \dots, n\}$.

The induced reduction rule is:

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

4.6 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 Σ. Figure 1 shows some notable examples of reduction rules. In order, they are: reflexivity, transitivity, congruence for abs, β-reduction, η-expansion, and expansion of the fixpoint operator.

For the example of the fixpoint operator (rule fix-Exp), we consider the signature Σ_{fix}, as described in [Ahrens et al. 2019, Section 6.4] (but without enforcing the fixpoint equation, which is replaced here by the reduction rule under consideration). A model of Σ_{fix} is a monad R equipped with an R-module morphism $\mathsf{fix} : R' \longrightarrow R$.

$$\frac{}{T \leadsto T}\,\text{Refl} \qquad \frac{T \leadsto U \qquad U \leadsto W}{T \leadsto W}\,\text{Trans}$$

$$\frac{T \leadsto U}{\mathsf{abs}(T) \leadsto \mathsf{abs}(U)}\,\text{abs-Cong} \qquad \frac{T \leadsto T' \qquad U \leadsto U'}{\mathsf{app}(T, U) \leadsto \mathsf{app}(T', U')}\,\text{applr-Cong}$$

$$\frac{T \leadsto T'}{\mathsf{app}(T, U) \leadsto \mathsf{app}(T', U)}\,\text{appl-Cong} \qquad \frac{U \leadsto U'}{\mathsf{app}(T, U) \leadsto \mathsf{app}(T, U')}\,\text{appr-Cong}$$

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

$$\frac{}{T \leadsto \mathsf{abs}(\mathsf{app}(\iota(T), *))}\,\eta\text{-Exp} \qquad \frac{}{\mathsf{abs}(\mathsf{app}(\iota(T), *)) \leadsto T}\,\eta\text{-Contr}$$

Here, $\iota : \Theta \longrightarrow \Theta'$ denotes the canonical morphism $\iota_{R,X} : R(X) \longrightarrow R(X + 1)$.

Fig. 1. Examples of reduction rules.

Rule	Signature	Metavariables	Hypotheses	Conclusion
Refl	any	Θ		$(0, \langle \mathsf{id}, \mathsf{id} \rangle)$
Trans	any	Θ^3 for (T, U, W)	$(0, \pi_{1,2}), (0, \pi_{2,3})$	$(0, \pi_{1,3})$
abs-Cong	Σ_{LC}	$\Theta' \times \Theta'$ for (T, U)	$(1, \mathsf{id})$	$(0, \mathsf{abs} \times \mathsf{abs})$
applr-Cong	Σ_{LC}	Θ^4 for (T, T', U, U')	$(0, \pi_{1,2}), (0, \pi_{3,4})$	$(0, \mathsf{app} \times \mathsf{app})$
appl-Cong	Σ_{LC}	Θ^3 for (T, T', U)	$(0, \pi_{1,2})$	$(0, \langle \mathsf{app} \circ \pi_{1,3}, \mathsf{app} \circ \pi_{2,3} \rangle)$
appr-Cong	Σ_{LC}	Θ^3 for (T, U, U')	$(0, \pi_{2,3})$	$(0, \langle \mathsf{app} \circ \pi_{1,2}, \mathsf{app} \circ \pi_{1,3} \rangle)$
β-Red	Σ_{LC}	$\Theta' \times \Theta$ for (T, U)		$(0, c_{\beta\text{-Red}})$
fix-Exp	Σ_{fix}	Θ'		$(0, c_{\text{fix-Exp}})$
η-Exp	Σ_{LC}	Θ		$(0, \langle \mathsf{id}, b_\eta \rangle)$
η-Contr	Σ_{LC}	Θ		$(0, \langle b_\eta, \mathsf{id} \rangle)$

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

Fig. 2. Modules and term pairs relative to the reduction rules of Figure 1.

Figure 2 lists the modules and term pairs for hypotheses and conclusion of each of these reduction rules. There, $\pi_{i,j}$ designates the pair projection described in Definition 4.3. Below we present different sets of reduction rules over the signature Σ_{LC} for different variants of lambda calculus.

Variant of lambda calculus	Associated reduction rules
weak head β (Example 3.8)	β-Red, appl-Cong
congruent β (Example 3.9)	β-Red, abs-Cong, appl-Cong, appr-Cong
parallel β (Example 3.10)	β-Red, abs-Cong, applr-Cong

5 SIGNATURES FOR REDUCTION MONADS AND INITIALITY

In this section, we define the notion of *reduction signature*, consisting of a signature for monads Σ and a family of reduction rules over Σ (see Section 5.1). As usual, we assign to each such signature a *category of models*. We call a reduction signature *effective* if the associated category of models has an initial object. Our main result, Theorem 5.10 (see Section 5.3), states that a reduction signature is effective as soon as its underlying signature for monads is effective.

5.1 Signatures and Their Models

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

Definition 5.1. A **reduction signature** is a pair $(\Sigma, \mathfrak{R})$ of a signature Σ for monads and a family $\mathfrak{R}$ of reduction rules over Σ.

Definition 5.2. Given a reduction monad R and a reduction signature $\mathcal{S} = (\Sigma, \mathfrak{R})$, an **action of $\mathcal{S}$ in** R consists of an action of Σ in its underlying monad $\underline{R}$ and an action of each reduction rule of $\mathfrak{R}$ in R.

Definition 5.3. Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a reduction signature. A **model of $\mathcal{S}$** is a reduction monad equipped with an action of $\mathcal{S}$, or equivalently, a reduction Σ-model equipped with an action of each reduction rule of $\mathfrak{R}$.

5.2 The Functors $\mathsf{Hyp}_{\mathcal{A}}$ and $\mathsf{Con}_{\mathcal{A}}$

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

Definition 5.4. Let Σ be a signature for monads, and $\mathcal{A}$ be a reduction rule over Σ. Definition 4.9 assigns to each model R of Σ the R-modules $\mathsf{Hyp}_{\mathcal{A}}(R)$ and $\mathsf{Con}_{\mathcal{A}}(R)$. These assignments extend to functors $\mathsf{Hyp}_{\mathcal{A}}, \mathsf{Con}_{\mathcal{A}} : \mathsf{RedMon}^{\Sigma} \to \int_R \mathsf{Mod}(R)$.

Proposition 5.5. *Given the same data, the functors* $\mathsf{Hyp}_{\mathcal{A}}$ *and* $\mathsf{Con}_{\mathcal{A}}$ *are* RedMon^{Σ}*-modules, i.e., they commute with the forgetful functors to* Mon*:*

$$\begin{tikzcd}
\mathsf{RedMon}^{\Sigma} \arrow[rr, shift left, "\mathsf{Hyp}_{\mathcal{A}}"] \arrow[rr, shift right, "\mathsf{Con}_{\mathcal{A}}"'] \arrow[dr] & & \int_R \mathsf{Mod}(R) \arrow[dl] \\
& \mathsf{Mon} &
\end{tikzcd}$$

5.3 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 5.10, which gives a sufficient condition for $\mathcal{S}$ to admit an initial model.

Definition 5.6. Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a reduction signature. A morphism between models R and T of $\mathcal{S}$ is a morphism f of reduction Σ-models commuting with the action of any reduction rule $\mathcal{A} \in \mathfrak{R}$, i.e., such that, for any such $\mathcal{A}$, the following diagram of natural transformations commutes:

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \arrow[r] \arrow[d, "\mathsf{Hyp}_{\mathcal{A}}(f)"'] & \mathsf{Con}_{\mathcal{A}}(R) \arrow[d, "\mathsf{Con}_{\mathcal{A}}(f)"] \\
\mathsf{Hyp}_{\mathcal{A}}(T) \arrow[r] & \mathsf{Con}_{\mathcal{A}}(T)
\end{tikzcd}$$

Example 5.7 (Example 4.12 continued). Consider the reduction signature consisting of the signature Σ_{app} of a binary operation app and the reduction rule of congruence (Section 4.1) for application (Example 4.6).

Let R and T be models for this signature: they are reduction Σ_{app}-models equipped with an action ρ and τ, in the alternative sense of Example 4.12. A reduction Σ_{app}-model morphism (f, α) between R and T is a model morphism if, for any set X, any sextuple (A, A', B, B', m, n) with $(A, A', B, B') \in R^4(X)$, $m : A \blacktriangleright A'$, and $n : B \blacktriangleright B'$, the reduction $\rho(A, A', B, B') : \mathrm{app}(A, B) \blacktriangleright \mathrm{app}(A', B')$ is mapped to the reduction $\tau(f(A), f(A'), f(B), f(B'))$ by $\alpha : \mathrm{Red}(R) \longrightarrow \mathrm{Red}(T)$.

Proposition 5.8. *Let $\mathcal{S} = (\Sigma, \mathfrak{R})$ be a reduction signature. Models of $\mathcal{S}$ and their morphisms, with the obvious composition and identity, define a category that we denote by $\mathsf{RedMon}^{\mathcal{S}}$, equipped with a forgetful functor to RedMon^{Σ}.*

Definition 5.9. A reduction signature $\mathcal{S}$ is said to be **effective** if its category of models $\mathsf{RedMon}^{\mathcal{S}}$ has an initial object, denoted $\widehat{\mathcal{S}}$. In this case, we say that $\widehat{\mathcal{S}}$ (or more precisely the underlying reduction monad) is **generated (or specified) by $\mathcal{S}$**.

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

Theorem 5.10. *Let $(\Sigma, \mathfrak{R})$ be a reduction signature. If Σ is effective, then so is $(\Sigma, \mathfrak{R})$.*

The proof of this theorem is given in Section 6.

Definition 5.11. A reduction signature $(\Sigma, \mathfrak{R})$ is called **algebraic** if Σ is (in the sense of [Ahrens et al. 2019]).

Theorem 32 of [Ahrens et al. 2019] entails the following corollary:

Corollary 5.12. *Any algebraic reduction signature is effective.*

All the examples of reduction signatures considered here satisfy the condition of Corollary 5.12.

Example 5.13 (Variants of lambda calculus). In Section 4.6, we considered different sets of reduction rules for different variants of lambda calculus. Each such set $\mathfrak{R}$ defines an algebraic reduction signature $(\Sigma_{\mathsf{LC}}, \mathfrak{R})$.

Variant of lambda calculus	Signature	Initial model
weak head β (Example 3.8)	$\mathcal{S}_{\mathsf{LC}_{wh\beta}}$	$\mathsf{LC}_{wh\beta}$
congruent β (Example 3.9)	$\mathcal{S}_{\mathsf{LC}_{\beta}}$	LC_{β}
parallel β (Example 3.10)	$\mathcal{S}_{\mathsf{LC}_{\beta\parallel}}$	$\mathsf{LC}_{\beta\parallel}$

Remark 5.14 (Continuation of Remark 3.2). Just as our reduction monads are “proof-relevant” (cf. Remark 3.2), our notion of reduction signature allows for the specification of multiple reductions between terms. As a trivial example , duplicating the β-rule in the signature $\mathcal{S}_{\mathsf{LC}_{\beta}}$ yields two distinct β-reductions in the initial model.

Example 5.15 (Reduction signature of lambda calculus with a fixpoint operator). The signature $\mathcal{S}_{\mathsf{LC}_{\mathrm{fix}}}$ specifying the reduction monad $\mathsf{LC}_{\mathrm{fix}}$ of the lambda calculus with a fixpoint operator extends the signature $\mathcal{S}_{\mathsf{LC}_{\beta}}$ of Example 5.13 with:

- a new operation $\mathsf{fix} : \Theta' \longrightarrow \Theta$ (thus extending the signature for monads Σ_{LC});
- the reduction rule for the fixpoint reduction (cf. Section 4.6);
- a congruence rule for fix:

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

6 PROOF OF THEOREM 5.10

This section details the proof of Theorem 5.10.

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

In Section 6.1, 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 6.2, we give an alternative definition of the category of models that we make use of in the proof of effectivity, in Section 6.3.

6.1 Normalizing Reduction Rules

Definition 6.1. A reduction rule $(\mathcal{V}, (n_j, h_j)_{j\in J}, (n, c))$ is said to be **normalized** if $n = 0$.

Lemma 6.2. *Let $\mathcal{A} = (\mathcal{V}, (n_j, h_j)_{j\in J}, (n, c))$ be a reduction rule over Σ. Then there exists a normalized reduction rule $\mathcal{A}'$ over Σ such that the induced notion of action is equivalent, in the sense that:*

- *for a reduction Σ-model R, there is a bijection between actions of $\mathcal{A}$ in R and actions of $\mathcal{A}'$ in R;*
- *a morphism between reduction Σ-models equipped with an action of $\mathcal{A}$ preserves the action (in the sense of Definition 5.6) 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 6.3. *Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i\in I}, (n, c))$ be a reduction rule over Σ. By universal property of the pullback $\mathsf{Con}_{\mathcal{A}}(R) = c^*\mathsf{Red}(R)^{(n)}$, an action can be alternatively be defined as an R-module morphism $\sigma : \mathsf{Hyp}_{\mathcal{A}}(R) \to \mathsf{Red}(R)^{(n)}$ making the following diagram commute*

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \arrow[r, "\sigma"] \arrow[d] & \mathsf{Red}(R)^{(n)} \arrow[d, "\mathsf{red}_R{}^{(n)}"] \\
\mathcal{V}(R) \arrow[r, "c"'] & R^{(n)} \times R^{(n)}
\end{tikzcd} \tag{3}$$

Lemma 6.4. *Using this alternative definition of action, a morphism between models R and T of a reduction signature $\mathcal{S} = (\Sigma, \mathfrak{R})$ is a morphism f of reduction Σ-models making the following diagram commute, for any reduction rule $\mathcal{A} = (\mathcal{V}, (n_i, l_i, r_i)_{i\in I}, (n, l, r))$ of $\mathfrak{R}$:*

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \arrow[r] \arrow[d, "\mathsf{Hyp}_{\mathcal{A}}(f)"'] & \mathsf{Red}(R)^{(n)} \arrow[d, "\mathsf{Red}(f)^{(n)}"] \\
\mathsf{Hyp}_{\mathcal{A}}(T) \arrow[r] & \mathsf{Red}(T)^{(n)}
\end{tikzcd}$$

We now prove Lemma 6.2 using these alternative definitions:

Proof of Lemma 6.2. The reduction rule $\mathcal{A}' = (\mathcal{V}', (n_j, h'_j)_{j\in J}, (0, c'))$ is defined as follows:

- Metavariables: $\mathcal{V}' = \mathcal{V} \times \Theta^n$
- Hypotheses: For each $j \in J$, $h'_j : \mathcal{V}' \to \Theta^{(n_j)} \times \Theta^{(n_j)}$ is defined as the composition of $\pi_1 : \mathcal{V} \times \Theta^n \to \mathcal{V}$ with $h_j : \mathcal{V} \to \Theta^{(n_j)} \times \Theta^{(n_j)}$.
- Conclusion: The morphism $c' : \mathcal{V} \times \Theta^n \to \Theta \times \Theta$ maps a model R of Σ to the n^{th} transpose of $c : \mathcal{V}(R) \to R^{(n)} \times R^{(n)}$ with respect to the adjunction $\_ \times R \dashv \_'$ in $\mathsf{Mod}(R)$ described in [Ahrens et al. 2018, Proposition 13].

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

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \arrow[r, "\tau"] \arrow[d] & \mathsf{Red}(R)^{(n)} \arrow[d, "\mathsf{red}_R^{(n)}"] \\
\mathcal{V}(R) \arrow[r, "c"'] & R^{(n)} \times R^{(n)}
\end{tikzcd}$$

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

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \times R^n \arrow[r, "\tau^*"] \arrow[d] & \mathsf{Red}(R) \arrow[d, "\mathsf{red}_R"] \\
\mathcal{V}(R) \times R^n \arrow[r, "c^*"'] & R \times R
\end{tikzcd}$$

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

Corollary 6.5. *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 6.2. □

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

6.2 Models as Vertical Algebras

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

First we rephrase the notion of action of a reduction rule as an algebra structure for a suitably chosen endofunctor. Indeed, an action of a normalized reduction rule $\mathcal{A} = (\mathcal{V}, (n_j, h_j)_{j \in J}, (0, c))$ in a reduction Σ-model R is given by a R-module morphism $\tau : \mathsf{Hyp}_{\mathcal{A}}(R) \longrightarrow \mathsf{Red}(R)$ such that the following square commutes:

$$\begin{tikzcd}
\mathsf{Hyp}_{\mathcal{A}}(R) \arrow[r, "\tau"] \arrow[d] & \mathsf{Red}(R) \arrow[d, "\mathsf{red}_R"] \\
\mathcal{V}(R) \arrow[r, "p"'] & R \times R
\end{tikzcd}$$

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

Lemma 6.6. *Given any model R of Σ, the assignment $(M, p : M \longrightarrow R \times R) \mapsto F_{\mathcal{A}|R}(M, c)$ yields an endofunctor $F_{\mathcal{A}|R}$ on $\mathsf{Mod}(R)/R^2$. An action of $\mathcal{A}$ in a reduction Σ-model R is exactly the same as an algebra structure for this endofunctor on $(\mathsf{Red}(R), \mathsf{red}_R) \in \mathsf{Mod}(R)/R^2$.*

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

Proof. This is a consequence of the functoriality of $\mathrm{Hyp}_{\mathcal{A}}$, as noticed in Section 5.2. □

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

Proposition 6.7. *Let $F_{\mathcal{S}} : \mathrm{RedMon}^{\Sigma} \to \mathrm{RedMon}^{\Sigma}$ be the coproduct $\coprod_i F_{\mathcal{A}_i}$. Then, the category of models of $\mathcal{S}$ is isomorphic to the* ***category of vertical algebras*** *of $F_{\mathcal{S}}$ defined as follows:*

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

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

Lemma 6.8. *The forgetful functor from the category of models of $\mathcal{S}$ to the category of models of Σ 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 $\mathcal{U}^{\Sigma} : \mathrm{RedMon}^{\Sigma} \to \mathrm{Mon}^{\Sigma}$ is a fibration) and $F = F_{\mathcal{S}}$:

Lemma 6.9. *Let $p : E \to B$ be a fibration and F and endofunctor on E satisfying $p \cdot F = p$. Then the category of vertical algebras of F is fibered over B.*

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

$$\begin{tikzcd}
F(a^*R) \arrow[r, "F(\overline{a})"] \arrow[d, "\rho"'] & F(R) \arrow[d, "r"] \\
a^*R \arrow[r, "\overline{a}"'] & R
\end{tikzcd}$$

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

$$\begin{tikzcd}
F(S) \arrow[r, "F(w)"] \arrow[d, "s"'] & F(a^*R) \arrow[d, "\rho"] \\
S \arrow[r, "w"'] & a^*R
\end{tikzcd}$$

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

We want to apply this lemma for proving Lemma 6.8. We thus need to show that $\mathcal{U}^{\Sigma} : \mathrm{RedMon}^{\Sigma} \to \mathrm{Mon}^{\Sigma}$ is a fibration:

Lemma 6.10. *The forgetful functors* $\mathrm{RedMon} \to \mathrm{Mon}$ *and* $\mathcal{U}^{\Sigma} : \mathrm{RedMon}^{\Sigma} \to \mathrm{Mon}^{\Sigma}$ *are fibrations.*

Proof. We have the two following pullbacks:

$$\begin{tikzcd}
\mathsf{RedMon}^{\Sigma} \arrow[r] \arrow[d, "\mathcal{U}^{\Sigma}"'] \arrow[dr, phantom, "\lrcorner", very near start] & \mathsf{RedMon} \arrow[r] \arrow[d] \arrow[dr, phantom, "\lrcorner", very near start] & V(\int_R \mathsf{Mod}(R)) \arrow[d, "\text{codom}"] \\
\mathsf{Mon}^{\Sigma} \arrow[r] & \mathsf{Mon} \arrow[r, "\Theta\times\Theta"'] & \int_R \mathsf{Mod}(R)
\end{tikzcd}$$

where $V(\int_R \mathsf{Mod}(R))$ is the full subcategory of arrows of $\int_R \mathsf{Mod}(R)$ which are vertical (that is, they are mapped to the identity monad morphism by the functor from $\int_R \mathsf{Mod}(R)$ to Mon), and codom maps such an arrow to its codomain. By [Ahrens et al. 2018, Propositions 4 and 8], the category $\int_R \mathsf{Mod}(R)$ has fibered finite limits, so that codom is a fibration ([Jacobs 1999, Exercise 9.4.2 (i)]).
Now, Proposition 8.1.15 of [Borceux 1994] states that a pullback of a fibration is a fibration. Thus, the middle functor $\mathsf{RedMon} \to \mathsf{Mon}$ is a fibration, and then, $\mathcal{U}^{\Sigma} : \mathsf{RedMon}^{\Sigma} \to \mathsf{Mon}^{\Sigma}$ also is. □

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

Proof of Lemma 6.8. Apply Lemma 6.9 with the fibration $p = \mathcal{U}^{\Sigma}$ (Lemma 6.10) and $F = F_{\mathcal{S}}$. □

6.3 Effectivity

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

Lemma 6.11. *Let $p : E \to B$ be a fibration, b_0 be an initial object in B and e_0 be an object over b_0 that is initial in the fiber category over b_0. Then e_0 is initial in E.*

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

Lemma 6.12. *The fiber category over a given model R of Σ through the fibration from models of $\mathcal{S}$ (Lemma 6.8) is the category of algebras of the endofunctor $F_{\mathcal{S}|R} = \coprod_i F_{\mathcal{A}_i|R}$ on the slice category $\mathsf{Mod}(R)/R^2$.*

Thus, our task is to construct the initial algebra of some specific endofunctor. Adámek's theorem [Adámek 1974] provides a sufficient condition for the existence of an initial algebra:

Lemma 6.13 (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 6.14. *The category $\mathsf{Mod}(R)/R^2$ is cocomplete for any monad R.*

Proof. The category of modules $\mathsf{Mod}(R)$ over a given monad R is cocomplete [Ahrens et al. 2018, Proposition 4], so any of its slice categories is, by the dual of [Mac Lane 1998, Exercise V.1.1], in particular $\mathsf{Mod}(R)/R^2$. □

Let us show that the finitary requirement of Lemma 6.13 is also satisfied for the case of a signature with a single reduction rule:

Lemma 6.15. *Let $\mathcal{A} = (\mathcal{V}, (n_i, h_i)_{i \in I}, (0, c))$ be a normalized reduction rule over Σ, and R be a model of Σ. Then, $F_{\mathcal{A}|R}$ is finitary.*

Proof. In this proof, we denote by F the endofunctor $F_{\mathcal{A}|R}$ on $\mathrm{Mod}(R)/R^2$, by $\pi : D/d \longrightarrow D$ the projection for a general slice category, and by $\alpha : \pi \longrightarrow d$ the natural transformation from π to the functor constant at d induced by the underlying morphism of a slice object: $\alpha_p : \pi(p) \longrightarrow d$. Note that π creates colimits, by the dual of [Mac Lane 1998, Exercise V.1.1].

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

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

$$\begin{tikzcd}
 & \mathcal{V}(R) \arrow[dl, "h_{i,R}"'] \arrow[dr, "h_{i',R}"] \arrow[drr] & & \\
R^{(n_i)} \times R^{(n_i)} & & R^{(n_{i'})} \times R^{(n_{i'})} & \dots \\
\pi(q)^{(n_i)} \arrow[u, "\alpha_q^{(n_i)}"] & & \pi(q)^{(n_{i'})} \arrow[u, "\alpha_q^{(n_{i'})}"] &
\end{tikzcd}$$

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

$$\begin{tikzcd}
 & \mathcal{V}(R) \arrow[dl, "h_{i,R}"'] \arrow[dr, "h_{i',R}"] \arrow[drr] & & \\
R^{(n_i)} \times R^{(n_i)} & & R^{(n_{i'})} \times R^{(n_{i'})} & \dots \\
\operatorname{colim} \pi(J)^{(n_i)} \arrow[u, "\alpha_{\operatorname{colim} J}^{(n_i)}"] & & \operatorname{colim} \pi(J)^{(n_{i'})} \arrow[u, "\alpha_{\operatorname{colim} J}^{(n_{i'})}"] &
\end{tikzcd}$$

Now, as limits and colimits are computed pointwise in the category of modules, and as finite limits commute with filtered colimits in Set ([Mac Lane 1998, Section IX.2, Theorem 1]), we have that $\pi(F(\operatorname{colim} J))$, as the limit of such a diagram, is canonically isomorphic to the colimit of $\pi \cdot F \cdot J$. □

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

Lemma 6.16. *For any model R of Σ, 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 6.15), and so is finitary as colimits commute with colimits, by [Mac Lane 1998, Equation V.2.2]. □

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

Proof of Theorem 5.10. We assume that Σ is effective; let R be the initial model of Σ. We want to show that $\mathcal{S}$ has an initial model. We apply Lemma 6.11 with p the fibration from models of $\mathcal{S}$ to models of Σ (Lemma 6.8): we are left with providing an initial object in the fiber category over R. By Lemma 6.12, this boils down to constructing an initial algebra for the endofunctor $F_{\mathcal{S}|R}$ on the category $\mathrm{Mod}(R)/R^2$. We apply Lemma 6.13: $\mathrm{Mod}(R)/R^2$ is indeed cocomplete by Lemma 6.14, and $F_{\mathcal{S}|R}$ is finitary by Lemma 6.16). □

7 EXAMPLE: LAMBDA CALCULUS WITH EXPLICIT SUBSTITUTIONS

Here, we give a signature specifying the reduction monad of the lambda calculus with explicit substitutions as described in [Kesner 2009]. One feature of this example is that it involves operations subject to some equations, and on top of this syntax with equations, a "multigraph of reductions".

In Section 7.1, we present the underlying signature for monads, and in Section 7.2, we list the reduction rules of the signature.

7.1 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.

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

Definition 7.1. The signature $\Upsilon_{\mathsf{LC}_{\mathsf{ex}}}$ for the monad of the lambda calculus with explicit substitutions without equations is the coproduct of $\Theta' \times \Theta$ and Σ_{LC}.

7.1.2 Equation. The syntax of lambda calculus with explicit substitutions of [Kesner 2009] is subject to the equation (see [Kesner 2009, Figure 1, "Equations"])

$$t[x/u][y/v] = t[y/v][x/u] \quad \text{if } y \notin \mathsf{fv}(u) \text{ and } x \notin \mathsf{fv}(v) \ . \tag{4}$$

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

$$\begin{tikzcd}
\Theta'' \times \Theta \times \Theta \arrow[r, "\Theta'' \times \iota \times \Theta"] & \Theta'' \times \Theta' \times \Theta \arrow[r, "\mathsf{esubst}' \times \Theta"] & \Theta' \times \Theta \arrow[r, "\mathsf{esubst}"] & \Theta \\
\Theta'' \times \Theta \times \Theta \arrow[r, "\Theta'' \times \Theta \times \iota"'] & \Theta'' \times \Theta \times \Theta' \arrow[r, "\langle \mathsf{esubst}^{\vee} \circ \pi_{1,3}, \pi_2 \rangle"'] & \Theta' \times \Theta \arrow[r, "\mathsf{esubst}"'] & \Theta
\end{tikzcd} \tag{5}$$

Here, ι denotes the canonical morphism $\Theta \to \Theta'$ as before, and $\mathsf{esubst}^{\vee}$ is the composition of esubst' with $\mathsf{swap} : \Theta'' \to \Theta''$ swapping the two fresh variables.

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

Definition 7.2. The signature $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$ of the lambda calculus monad with explicit substitutions consists of $\Upsilon_{\mathsf{LC}_{\mathsf{ex}}}$ and the single $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$-equation stating the equality between the two morphisms of Equation 5.

Lemma 7.3. *The signature $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$ for monads is effective*

Proof. This is a direct consequence of Corollary 5.12. □

7.2 Reduction Rules for Lambda Calculus with Explicit Substitutions

The reduction signature for the lambda calculus with explicit substitutions consists of two components: the first one is the signature for monads $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$ of Definition 7.2; the second one is the list of reduction rules that we enumerate here, taken from [Kesner 2009, Figure 1, "Rules"]. Except for congruence, none of them involve hypotheses.

First, let us state the congruence rules (that are implicit in [Kesner 2009]):

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

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

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

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

Fig. 3. Reduction rules of lambda calculus with explicit substitutions.

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

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

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

$$\frac{}{\mathsf{esubst}(\mathsf{esubst}'(T, \kappa(U)), V) \rightsquigarrow \mathsf{esubst}(\mathsf{esubst}^{\vee}(T, \iota(V)), \mathsf{esubst}(\kappa(U), V))}\ [][]$$

$$\kappa : \Theta_* \longrightarrow \Theta'$$
$$\mathsf{esubst}^{\vee} : \Theta'' \times \Theta' \longrightarrow \Theta'$$

where Θ_* is the 1-hole context $\Sigma_{\mathsf{LC}_{\mathrm{ex}}}$-submodule of Θ' (Definition 7.4), and $\mathsf{esubst}^{\vee}$ is defined as the composition

$$\Theta'' \times \Theta' \xrightarrow{\mathsf{swap} \times \Theta'} \Theta'' \times \Theta' \xrightarrow{\mathsf{esubst}} \Theta'$$

Here, swap exchanges the fresh variables:

$$\mathsf{swap}_{X,R} : R((X + \{*_1\}) + \{*_2\}) \longrightarrow R((X + \{*_1\}) + \{*_2\})(t)$$
$$\mathsf{swap}_{X,R} : t \mapsto t\{*_1 := *_2; *_2 := *_1\}$$

Fig. 4. Reduction rules of Figure 3 reformulated in our setting.

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

They are translated into reduction rules through the protocol described in Section 4.5.

Figure 3 gives Kesner's rules. Five out of six of Kesner's rules translate straightforwardly, see Figure 4. Note how the explicit weakening $\iota : \Theta \longrightarrow \Theta'$ accounts for the side condition $x \notin \mathsf{fv}(t)$ of the Gc-rule in Figure 3.

Expressing the side condition $y \in \mathsf{fv}(u)$ of the [][]-rule of Figure 3 requires the definition of the $\Sigma_{\mathsf{LC}_{\mathrm{ex}}}$-module Θ_* such that $\mathsf{LC}_{\mathrm{ex}*}$ is the submodule of $\mathsf{LC}_{\mathrm{ex}}'$ 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 elements in R' having at least one occurrence of the fresh variable $*$ as follows, for a given model R of $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$:

- $\eta(*) \in R_*(X)$, for any set X;
- (application)
  - if $t \in R(X)$ and $u \in R_*(X)$, then $\mathrm{app}(\iota(t), u) \in R_*(X)$
  - if $t \in R_*(X)$ and $u \in R(X)$, then $\mathrm{app}(t, \iota(u)) \in R_*(X)$
  - if $t \in R_*(X)$ and $u \in R_*(X)$, then $\mathrm{app}(t, u) \in R_*(X)$
- 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 $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$-module Θ_* equipped with a morphism $\kappa : \Theta_* \to \Theta'$.

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

- the terminal $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$-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 6.13). Unfortunately, the resulting $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$-module does not yield the module that we are expecting in the case of the monad $\mathsf{LC}_{\mathsf{ex}}$: it does not satisfy Equation 5, and thus contains more terms than necessary. To obtain the desired $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$-module, we equip Θ' with its canonical algebra structure, inducing a morphism from the initial algebra, and we define Θ_* as the image of this morphism, thus equipped with an inclusion $\kappa : \Theta_* \to \Theta'$.

Definition 7.4. We define the $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$**-module of "one-hole contexts"** to be Θ_*, equipped with an inclusion $\kappa : \Theta_* \to \Theta'$.

Remark 7.5. 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 7.6. The reduction signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{ex}}}$ of the lambda calculus reduction monad with explicit substitutions consists of the signature $\Sigma_{\mathsf{LC}_{\mathsf{ex}}}$ of Definition 7.2 and all the reduction rules specified in this section.

LEMMA 7.7. *The reduction signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{ex}}}$ is effective.*

PROOF. Apply Theorem 5.10. The underlying signature for monads is effective by Lemma 7.3. □

8 RECURSION

In this section, we derive, for any effective reduction signature $\mathcal{S}$, a recursion principle from initiality. In Section 8.1, we state this recursion principle, then we give an example of application in Section 8.2, by translating lambda calculus with a fixpoint operator to lambda calculus. In Section 8.2, we apply this principle to translate lambda calculus with explicit substitutions into lambda calculus with *unary congruent substitution*. Then, in Section 8.4, we translate this latter variant of lambda calculus into lambda calculus closed under identity and composition of reductions.

8.1 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 8.1 (Recursion principle). *Let $\mathcal{S}$ be an effective reduction signature, and R be the reduction monad underlying the initial model. Let T be a reduction monad. Any action τ of $\mathcal{S}$ in T induces a reduction monad morphism $\hat{\tau} : R \longrightarrow T$.*

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

In the next sections, we illustrate this principle.

8.2 Translation of Lambda Calculus with Fixpoint to Lambda Calculus

In this section, we consider the signature $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ of Example 5.15 for the lambda calculus with an explicit fixpoint operator.

We build, by recursion, a reduction monad morphism from the initial model $\mathsf{LC}_{\mathsf{fix}}$ of this signature to LC^*_β, the "closure under identity and composition of reductions" (Definition 3.11) of the initial model LC_β of the signature $\mathcal{S}_{\mathsf{LC}_\beta}$ (Example 5.13).

As explained in Section 8.1, we need to define an action of $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ in LC^*_β. Note that $\mathcal{S}_{\mathsf{LC}_{\mathsf{fix}}}$ is an extension of $\mathcal{S}_{\mathsf{LC}_\beta}$ (Example 5.15). First, we focus on the core $\mathcal{S}_{\mathsf{LC}_\beta}$ part: we show that the reduction monad LC^*_β inherits the canonical action of $\mathcal{S}_{\mathsf{LC}_\beta}$ in LC_β.

Lemma 8.2. *There is an action of $\mathcal{S}_{\mathsf{LC}_\beta}$ in LC^*_β.*

We have formalized a proof of this statement in Agda.[1]

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 $r_1 \ldots r_n$ going from t_0 to t_n. We want to provide a reduction between $\mathsf{abs}(t_0)$ and $\mathsf{abs}(t_n)$. For each i, we have a reduction between $\mathsf{abs}(t_{i-1})$ and $\mathsf{abs}(t_i)$. By composing the corresponding sequence, we obtain the desired reduction. □

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

- An operation $\mathsf{fix} : \mathsf{LC}^{*\prime}_\beta \longrightarrow \mathsf{LC}^*_\beta$. A fixpoint combinator Y is a closed term with the property that for any other term t, the term $\mathsf{app}(Y, t)$ β-reduces in some steps to $\mathsf{app}(t, \mathsf{app}(Y, t))$. Here, we choose a fixpoint combinator Y (for example, the one of Curry), and set $\mathsf{fix}_X(t) = \mathsf{app}(Y, \mathsf{abs}(t))$, in accordance with [Ahrens et al. 2018, Section 8.4].
- An action of the reduction rule

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

We denote by $r \in \mathsf{Red}(\mathsf{LC}^*_\beta)(\{*\})$ a reduction between $\mathsf{app}(Y, *)$ and $\mathsf{app}(*, \mathsf{app}(Y, *))$. Then, r induces an LC-module morphism $\hat{r} : \mathsf{LC} \longrightarrow \mathsf{Red}(\mathsf{LC}^*_\beta)$ by mapping an element $t \in \mathsf{LC}(X)$ to $r\{* := t\}$. We define the action of this reduction rule as the composition of the following reductions:

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

[1] The source code can be downloaded from https://github.com/amblafont/unary-subst-LCstar/blob/master/fiberlambda.agda.

- an action of the congruence rule

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

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

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

8.3 Translation of Lambda Calculus with Explicit Substitutions into Lambda Calculus with Congruent Unary Substitution

Here, we consider the reduction signature $\mathcal{S}_{\mathsf{LC}_{\mathrm{ex}}} = (\Sigma_{\mathsf{LC}_{\mathrm{ex}}}, \mathfrak{R}_{\mathsf{LC}_{\mathrm{ex}}})$ introduced in Definition 7.6. The underlying monad of the initial model $\mathsf{LC}_{\mathrm{ex}}$ is the monad of lambda calculus with an application and abstraction operation, and an explicit substitution operator $\mathsf{LC}_{\mathrm{ex}}{}' \times \mathsf{LC}_{\mathrm{ex}} \longrightarrow \mathsf{LC}_{\mathrm{ex}}$ satisfying Equation 5, for $R = \mathsf{LC}_{\mathrm{ex}}$. The associated reduction monad has all the rules specified in Section 7.

We build, by recursion, a reduction monad morphism from the initial model $\mathsf{LC}_{\mathrm{ex}}$ of this signature to $\mathsf{LC}_{\text{1-cong}}$, a variant of the lambda calculus specified by the signature $\mathcal{S}_{\mathsf{LC}_\beta}$ (Example 5.13) extended with the *congruence for unary substitution*:

$$\frac{T \rightsquigarrow T'}{U\{* := T\} \rightsquigarrow U\{* := T'\}}$$

Note that this reduction rule accounts for the reflexivity rule, and makes congruences for application (but not congruence for abstraction) redundant:

Reflexivity Any $U \in \mathsf{LC}(X)$ can be weakened into $\iota(U) \in \mathsf{LC}'(X)$. Then, consider any reduction $m : T \blacktriangleright T'$. The action of the reduction rule above yields a reduction between $\iota(U)\{* := T\} = U$ and $\iota(U)\{* := T'\} = U$. By choosing m adequately (for example, take the β-reduction between $\mathsf{app}(\mathsf{abs}(*), \mathsf{abs}(*))$ and $\mathsf{abs}(*)$), this yields an action of the reflexivity reduction rule.

Congruence Consider the left congruence rule (the cases of the right one and the congruence for abstraction are similar): from any reduction $m : T \blacktriangleright T'$, we want a reduction between $\mathsf{app}(T, U)$ and $\mathsf{app}(T', U)$, for $T, T', U \in \mathsf{LC}(X)$. We obtain it by applying the action of unary congruent substitution to m for the term $\mathsf{app}(*, U)$. One checks that this indeed defines an action for the left congruence reduction rule of application.

As explained in Section 8.1, we need to define an action of $\mathcal{S}_{\mathsf{LC}_{\mathrm{ex}}}$ in $\mathsf{LC}_{\text{1-cong}}$:

- the operations of application and abstraction are those of $\mathsf{LC}_{\text{1-cong}}$ as the initial model of Σ_{LC} (recall that the underlying monad of $\mathsf{LC}_{\text{1-cong}}$ is just LC);
- the explicit substitution operation $\mathsf{LC}_{\text{1-cong}}{}' \times \mathsf{LC}_{\text{1-cong}} \longrightarrow \mathsf{LC}_{\text{1-cong}}$ is defined using the monadic substitution, mapping a pair $(t, u) \in \mathsf{LC}_{\text{1-cong}}(X + \{*\}) \times \mathsf{LC}_{\text{1-cong}}(X)$ to the monadic substitution $t\{* := u\}$;
- Equation 5 for the underlying monad is satisfied thanks to the usual monadic equations;
- the action of the congruence rules for application and abstraction are induced by the action of the congruence rule for unary substitution, as explained above;
- $\mathsf{LC}_{\text{1-cong}}$ has already an action for β-reduction;
- all the actions for the remaining reduction rules involving explicit substitution (except the congruences for explicit substitution that are discussed below) are given by an action of the reflexivity reduction rule;

- the non-obvious actions are the ones of the congruence rules for explicit substitution:
$$\frac{T \rightsquigarrow T'}{T[x/U] \rightsquigarrow T'[x/U]} \qquad \frac{U \rightsquigarrow U'}{T[x/U] \rightsquigarrow T[x/U']}$$
The left one is obtained from the substitution of the module of reductions (see Remark 3.3). The right one is exactly given by the action of the congruence rule for unary substitution.

Finally, by the recursion principle, we get a reduction monad morphism from LC_{ex} to $\mathsf{LC}_{\text{1-cong}}$. 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 β-reduction and congruences.

8.4 Translation of Lambda Calculus with Congruent Unary Substitution into Lambda Calculus

In the previous section, we translated lambda calculus with explicit substitution into lambda calculus with congruent unary substitution. In this section, we translate this variant of lambda calculus into LC^*_β (introduced in Section 8.2), the closure under identity and composition of reductions the initial model LC_β of the signature $\mathcal{S}_{\mathsf{LC}_\beta}$ (Example 5.13).

As per Section 8.1, we need to define an action in LC^*_β of the signature $\mathcal{S}_{\mathsf{LC}_\beta}$ extended with the reduction rule:
$$\frac{T \rightsquigarrow T'}{U\{* := T\} \rightsquigarrow U\{* := T'\}} \tag{6}$$
Thanks to Lemma 8.2, we have an action of $\mathcal{S}_{\mathsf{LC}_\beta}$ in LC^*_β. Thus, the main challenge consists in equipping LC^*_β with an action of the rule (6).

Proposition 8.3. *The reduction monad* LC^*_β *can be equipped with an action of the rule* (6).

We have formalized a proof of this statement in Agda.[2]

Proof. We will write $r : R \longrightarrow \mathsf{LC} \times \mathsf{LC}$ in place of $\mathsf{red}_{\mathsf{LC}^*_\beta} : \mathsf{Red}(\mathsf{LC}^*_\beta) \longrightarrow \mathsf{LC} \times \mathsf{LC}$.

Such an action is equivalently given (see Lemma 6.6) by a morphism $\alpha : \mathsf{LC}' \times R \longrightarrow R$ such that the following diagram commutes, where $q_X(t, m) = \big(t\{* := \mathsf{source}(m)\}, t\{* := \mathsf{target}(m)\}\big)$.
$$\begin{tikzcd} \mathsf{LC}' \times R \arrow[rr, dashed, "\alpha"] \arrow[dr, "q"'] & & R \arrow[dl, "r"] \\ & \mathsf{LC} \times \mathsf{LC} & \end{tikzcd} \tag{7}$$

We first construct the collection of functions $(\alpha_X)_X$ with $\alpha_X : \mathsf{LC}'(X) \times R(X) \longrightarrow R(X)$ and then show the two required properties, i.e., that it commutes with substitution (thus inducing a LC-module morphism), and that it satisfies Equation 7.

The construction of the collection of functions (without naturality conditions) is worked out in the functor category $[\mathsf{Set}_0, \mathsf{Set}]$, where Set_0 is the discretized category of sets (this base category allows us to get rid of naturality conditions). This is done by recursion on the first argument. More formally, we exploit some initiality property of $\mathsf{LC}' \cdot j$, where $j : \mathsf{Set}_0 \longrightarrow \mathsf{Set}$ is the inclusion of the discretized category of sets into sets. Indeed, $\mathsf{LC}' \cdot j$ is the initial algebra of the endofunctor
$$\Psi : [\mathsf{Set}_0, \mathsf{Set}] \longrightarrow [\mathsf{Set}_0, \mathsf{Set}] \qquad F \mapsto j + 1 + F \times F + F'$$
where F' is the functor mapping a set X to $F(X + 1)$.

[2]The source code can be downloaded from https://github.com/amblafont/unary-subst-LCstar/blob/master/fiberlambda.agda.

The two properties that we want to show about the collection of functions $(\alpha_X)_X$ are then proved by *induction* on the first argument, again exploiting initiality of $\mathsf{LC}' \cdot j$, as we explain below. The proof goes as follows:

(1) construct (by initiality) a morphism from $\mathsf{LC}' \cdot j$ to the exponential of $R \cdot j$ with itself, that is, to the functor $R^R : \mathrm{Set}_0 \longrightarrow \mathrm{Set}$ defined on objects by $X \mapsto R(X)^{R(X)}$;
(2) show that the induced morphism from $\mathsf{LC}' \cdot j \times R \cdot j \longrightarrow R \cdot j$ yields a LC-module morphism $\alpha : \mathsf{LC}' \times R \longrightarrow R$;
(3) show the commutation required by Equation 7.

Note how working in the functor category $[\mathrm{Set}_0, \mathrm{Set}]$ allows us to define the functor R^R as above, without worrying about the functorial action on morphisms. Below we sometimes omit the explicit precomposition with j in order to simplify the notation. Now we perform the steps listed above.

(1) As we argued before, $\mathsf{LC}' \cdot j$ is the initial algebra of Ψ, so our task consists in equipping R^R with an algebra structure for Ψ, that we split into the following four components, using the universal properties of the coproduct and the exponential in the category $[\mathrm{Set}_0, \mathrm{Set}]$:
  (a) The morphism $j \times R \longrightarrow R$ corresponds to the case of variables. We expect that the resulting module morphism $\alpha : \mathsf{LC}' \times R \longrightarrow R$ satisfies $\alpha_X(\eta(x), m) = \mathsf{refl}_X(x)$ for any $x \in X$, where $\mathsf{refl} : \mathsf{LC} \longrightarrow R$ maps a term to the reflexive reduction on itself and $\eta : \mathsf{Id} \longrightarrow \mathsf{LC}'$ is the unit of the monad LC. Accordingly, we define the morphism $j \times R \longrightarrow R$ as mapping a pair $(x, m) \in X \times R(X)$ to $\mathsf{refl}_X(\eta(x))$.
  (b) The morphism $R \longrightarrow R$ corresponds to the case of the fresh variable $*$. We expect that $\alpha_X(*, m) = m$. Accordingly, the required morphism is taken as the identity on R.
  (c) The morphism $(R \times R)^R \longrightarrow R^R$ corresponds to the case of an application. We expect that $\alpha_X(\mathsf{app}(t, u), m) = \text{app-cong}(\alpha_X(t, m), \alpha_X(u, m))$, where $\text{app-cong} : R \times R \longrightarrow R$ is the action of the reduction rule of congruence for application defined as $\text{app-cong}(m_1, m_2) = \mathsf{trans}(\text{app-cong}_1(m_1), \text{app-cong}_2(m_2))$, where trans denotes an action of the transitivity reduction rule with which we can equip LC^*_β (by concatenating sequences of reductions). Accordingly, the morphism is defined as app-cong^R.
  (d) The morphism $R'^{R'} \longrightarrow R^R$ corresponds to the case of an abstraction. We expect that $\alpha_X(\mathsf{abs}(t), m) = \text{abs-cong}_X(\alpha_{X+1}(t, R\iota_X(m)))$, where $\iota : \mathsf{Id} \longrightarrow \mathsf{Id}'$ is the canonical inclusion. Accordingly, we take $\text{abs-cong}^{R\iota}$ as the the required morphism.

  By initiality, we get an algebra morphism from $\mathsf{LC} \cdot j$ to R^R, which by uncurrying yields a morphism $\alpha : \mathsf{LC}' \cdot j \times R \longrightarrow R$.
(2) Upgrading α into a module morphism from $\mathsf{LC}' \times R$ to R consists in showing compatibility with substitution in the following sense: for any map $f : X \longrightarrow \mathsf{LC}(Y)$, for any pair $(t, m) \in \mathsf{LC}'(X) \times R(X)$, the equality $\alpha_X(t, m)\{f\} = \alpha_Y(t\{f\}, m\{f\})$ is satisfied. This is shown by induction on $t \in \mathsf{LC}'(X)$. The case of variables requires a preliminary step: for $t = \eta(x)$, the equation amounts to $\mathsf{refl}(f(x)) = \alpha(\mathsf{LC}i(f(x)), m)$, which is not straightforward. We hence first prove by induction on $t \in \mathsf{LC}(X)$ that $\alpha(\mathsf{LC}i(t), m) = \mathsf{refl}(t)$. We do not detail these straightforward inductions, but rather explain the general methodology to perform induction on LC' (the case of LC is similar). Suppose given, for each $t \in \mathsf{LC}'(X)$, a predicate $P_X(t)$. Then, one can form the functor $\mathsf{LC}'_{|P} : \mathrm{Set}_0 \longrightarrow \mathrm{Set}$ mapping a set X to the subset of $\mathsf{LC}'(X)$ satisfying the predicate P_X. It follows that $\mathsf{LC}'_{|P}$ embeds into LC, and if $\mathsf{LC}'_{|P}$ inherits the algebra structure for Ψ through this embedding, then by initiality we get a section of the embedding, which exactly translates the fact that any term $t \in \mathsf{LC}'(X)$ satisfies the property.
(3) It remains to show the commutation of Diagram 7. Again, an induction on the first argument (thus exploiting initiality of $\mathsf{LC}' \cdot j$) is enough to conclude. □

9 CONCLUSION AND FUTURE WORK

We introduced the notions of reduction monad and reduction signature. For each such signature, we defined 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 category of models has an initial object; then, we say that the reduction monad underlying the initial object is generated by the signature. Our main result identifies a simple sufficient condition for a reduction signature to be effective.

This work is the first step towards a theory for the algebraic specification of programming languages and their semantics. Future work could include

- generalizing our notion of signature to encompass richer languages;
- extending our work to simply-typed languages;
- proving modularity results for our signatures and their models, analogous to that of [Ahrens et al. 2019, Theorem 27].

Part of this is already under way [Hirschowitz et al. 2019; Lafont 2019].

ACKNOWLEDGMENTS

We thank the referees for their careful reading and thoughtful and constructive criticism. Furthermore, we thank Tom Hirschowitz for many discussions about this work and, in particular, for sharing his knowledge about related work.

Ahrens acknowledges the support of the Centre for Advanced Study (CAS) in Oslo, Norway, which funded and hosted the research project *Homotopy Type Theory and Univalent Foundations* during the 2018/19 academic year.

Lafont has been supported by the European Research Council under Grant No. 637339 (http://coqhott.gforge.inria.fr/).

Maggesi has been supported by Gruppo Nazionale per le Strutture Algebriche, Geometriche e le loro Applicazioni (GNSAGA-INdAM) and Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR).

REFERENCES

Jiří Adámek. 1974. Free algebras and automata realizations in the language of categories. *Commentationes Mathematicae Universitatis Carolinae* 015, 4 (1974), 589–602. http://eudml.org/doc/16649

Benedikt Ahrens. 2016. Modules over relative monads for syntax and semantics. *Mathematical Structures in Computer Science* 26 (2016), 3–37. Issue 1. https://doi.org/10.1017/S0960129514000103

Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. 2018. High-Level Signatures and Initial Semantics. In *27th EACSL Annual Conference on Computer Science Logic (CSL 2018) (Leibniz International Proceedings in Informatics (LIPIcs))*, Dan Ghica and Achim Jung (Eds.), Vol. 119. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 4:1–4:22. https://doi.org/10.4230/LIPIcs.CSL.2018.4

Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. 2019. Modular Specification of Monads Through Higher-Order Presentations. In *4th International Conference on Formal Structures for Computation and Deduction (FSCD 2019) (Leibniz International Proceedings in Informatics (LIPIcs))*, Herman Geuvers (Ed.), Vol. 131. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, 6:1–6:19. https://doi.org/10.4230/LIPIcs.FSCD.2019.6

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

Thorsten Altenkirch and Bernhard Reus. 1999. 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 (Lecture Notes in Computer Science)*, Jörg Flum and Mario Rodríguez-Artalejo (Eds.), Vol. 1683. Springer, 453–468. https://doi.org/10.1007/3-540-48168-0_32

Marc Bezem, Jan Willem Klop, and Roel de Vrijer (Eds.). 2003. *Term Rewriting Systems*. Cambridge Tracts in Theoretical Computer Science, Vol. 55. Cambridge University Press, New York, NY, USA.

Bard Bloom, Sorin Istrail, and Albert R. Meyer. 1988. Bisimulation Can't Be Traced. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages, San Diego, California, USA, January 10-13, 1988*, Jeanne Ferrante and P. Mager (Eds.). ACM Press, 229–239. https://doi.org/10.1145/73560.73580

Francis Borceux. 1994. *Handbook of Categorical Algebra*. Encyclopedia of Mathematics and its Applications, Vol. 2. Cambridge University Press. https://doi.org/10.1017/CBO9780511525865

Marcelo P. Fiore and Chung-Kil Hur. 2007. Equational Systems and Free Constructions. In *ICALP (Lecture Notes in Computer Science)*, Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki (Eds.), Vol. 4596. Springer, 607–618. https://doi.org/10.1007/978-3-540-73420-8_53

Marcelo P. Fiore and Ola Mahmoud. 2010. Second-Order Algebraic Theories (Extended Abstract). In *MFCS (Lecture Notes in Computer Science)*, Petr Hlinený and Antonín Kucera (Eds.), Vol. 6281. Springer, 368–380. https://doi.org/10.1007/978-3-642-15155-2_33

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

Marcelo P. Fiore and Sam Staton. 2006. A Congruence Rule Format for Name-Passing Process Calculi from Mathematical Structural Operational Semantics. In *21th IEEE Symposium on Logic in Computer Science (LICS 2006), 12-15 August 2006, Seattle, WA, USA, Proceedings*. IEEE Computer Society, 49–58. https://doi.org/10.1109/LICS.2006.7

Marcelo P. Fiore and Daniele Turi. 2001. Semantics of Name and Value Passing. In *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*. IEEE Computer Society, 93–104. https://doi.org/10.1109/LICS.2001.932486

Makoto Hamana. 2003. Term Rewriting with Variable Binding: An Initial Algebra Approach. In *Proceedings of the 5th ACM SIGPLAN International Conference on Principles and Practice of Declaritive Programming (PPDP '03)*. ACM, New York, NY, USA, 148–159. https://doi.org/10.1145/888251.888266

André Hirschowitz, Tom Hirschowitz, and Ambroise Lafont. 2019. Modules over monads and operational semantics. (Oct. 2019). https://hal.archives-ouvertes.fr/hal-02338144 working paper or preprint.

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

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

Tom Hirschowitz. 2019. Familial monads and structural operational semantics. *PACMPL* 3, POPL (2019), 21:1–21:28. https://doi.org/10.1145/3290334

Bart Jacobs. 1999. *Categorical Logic and Type Theory*. Number 141 in Studies in Logic and the Foundations of Mathematics. North Holland, Amsterdam.

Delia Kesner. 2009. A Theory of Explicit Substitutions with Safe and Full Composition. *Logical Methods in Computer Science* 5, 3 (2009). https://doi.org/10.2168/LMCS-5(3:1)2009

Bartek Klin. 2011. Bialgebras for structural operational semantics: An introduction. *Theor. Comput. Sci.* 412, 38 (2011), 5043–5069. https://doi.org/10.1016/j.tcs.2011.03.023

Ambroise Lafont. 2019. *Signatures and models for syntax and operational semantics in the presence of variable binding*. Ph.D. Dissertation. École Nationale Superieure Mines – Telecom Atlantique Bretagne Pays de la Loire – IMT Atlantique. https://arxiv.org/abs/1910.09162v2 forthcoming.

Christoph Lüth and Neil Ghani. 1997. Monads and Modular Term Rewriting. In *Category Theory and Computer Science, 7th International Conference, CTCS '97 (Lecture Notes in Computer Science)*, Eugenio Moggi and Giuseppe Rosolini (Eds.), Vol. 1290. Springer, 69–86. https://doi.org/10.1007/BFb0026982

Saunders Mac Lane. 1998. *Categories for the working mathematician* (second ed.). Graduate Texts in Mathematics, Vol. 5. Springer-Verlag, New York. xii+314 pages.

Gordon D. Plotkin. 2004. A structural approach to operational semantics. *J. Log. Algebr. Program.* 60-61 (2004), 17–139.

Peter Selinger. 2008. Lecture notes on the lambda calculus. *CoRR* abs/0804.3434 (2008). arXiv:0804.3434 http://arxiv.org/abs/0804.3434

Sam Staton. 2008. General Structural Operational Semantics through Categorical Logic. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*. IEEE Computer Society, 166–177. https://doi.org/10.1109/LICS.2008.43

Daniele Turi and Gordon D. Plotkin. 1997. Towards a Mathematical Operational Semantics. In *Proceedings, 12th Annual IEEE Symposium on Logic in Computer Science, Warsaw, Poland, June 29 - July 2, 1997*. IEEE Computer Society, 280–291. https://doi.org/10.1109/LICS.1997.614955