Modules over monads and operational semantics

André Hirschowitz, Tom Hirschowitz, Ambroise Lafont

▶ **To cite this version:**

André Hirschowitz, Tom Hirschowitz, Ambroise Lafont. Modules over monads and operational semantics. 5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020), 2020, Paris, France. pp.12:1–12:23, 10.4230/LIPIcs.FSCD.2020.12 . hal-02338144v2

HAL Id: hal-02338144

https://hal.science/hal-02338144v2

Submitted on 3 Mar 2020 (v2), last revised 1 Sep 2020 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Modules over monads and operational semantics

André Hirschowitz
Université Côte d'Azur, CNRS, Nice, France

Tom Hirschowitz
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000, Chambéry, France

Ambroise Lafont
University of New South Wales, Australia

Abstract

This paper is a contribution to the search for efficient and high-level mathematical tools to specify and reason about (abstract) programming languages or calculi. Generalising the *reduction monads* of Ahrens et al., we introduce *transition monads*, thus covering new applications such as $\overline{\lambda}\mu$-calculus, π-calculus, Positive GSOS specifications, differential λ-calculus, and the big-step, simply-typed, call-by-value λ-calculus. Finally, we design a notion of signature for transition monads that generates all our examples.

2012 ACM Subject Classification Theory of computation → Operational semantics

Keywords and phrases Operational semantics, Category theory

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The search for a mathematical notion of programming language goes back at least to Turi and Plotkin [24], who coined the name "Mathematical Operational Semantics", and explained how known classes of well-behaved rules for structural operational semantics, such as GSOS [6], can be categorically understood and specified via distributive laws and bialgebras. Their initial framework did not cover variable binding, and several authors have proposed variants which do [11, 10, 23], treating examples like the π-calculus. However, none of these approaches covers higher-order languages like the λ-calculus.

In recent work, following previous work on modules over monads for syntax with binding [15, 2], Ahrens et al. [3] introduce *reduction* monads, and show how they cover several standard variants of the λ-calculus. Furthermore, as expected in similar contexts, they propose a mechanism for specifying reduction monads by suitable signatures.

Our starting point is the fact that already the call-by-value λ-calculus does not form a reduction monad. Indeed, in this calculus, variables are placeholders for values but not for λ-terms; in other words, reduction, although it involves general terms, is stable under substitution by values only.

In the present work, we generalise reduction monads to what we call *transition* monads. The main new ingredients of our generalisation are as follows.

- We now have two kinds of terms, called *placetakers* and *states*: variables are placeholders for our placetakers, while reductions relate states. Typically, in call-by-value, small-step λ-calculus, placetakers are values, while states are general terms.
- We also have a set of types for placetakers, and a possibly different set of types for states. Typically, in call-by-value, simply-typed λ-calculus, both sets of types coincide and are given by simple types, while in $\overline{\lambda}\mu$-calculus, we have two placetaker types, one for terms and one for stacks, and one state type, for processes.
- We in fact have two possibly different kinds of states, *source* states and *target* states, so that a transition now relates a source state to a target state. Typically, in call-by-value, big-step λ-calculus, source states are general terms, while target states are values.

© Author: Please provide a copyright holder;
licensed under Creative Commons License CC-BY

- The relationship between placetakers and states is governed by two functors S_1 and S_2, as follows: given an object X (for variables), we have an object $T(X)$ of placetakers ('with free variables in X'), and the corresponding objects of source and target states are respectively $S_1(T(X))$ and $S_2(T(X))$ (see §2.2).

Reduction monads correspond to the untyped case with $S_1 = S_2 = \mathrm{Id}_{\mathbf{Set}}$.

In §2.1, after giving a 'monadic' definition of transition monads in terms of *relative monads* [4], we provide a 'modular' definition (in terms of modules over monads), which we prove equivalent in Proposition 7. From the modular point of view, a transition monad consists of a *placetaker* monad T, two *state* functors S_1, S_2, a *transition* T-module R, and two T-module morphisms $s\colon R \to S_1 \circ T$ and $t\colon R \to S_2 \circ T$. Such a triple (R, s, t) is thus an object of the slice category of T-modules over $(S_1 \circ T) \times (S_2 \circ T)$.

In §2.2, we present a series of examples of transition monads: $\overline{\lambda}\mu$-calculus, simply-typed λ-calculus (in its call-by-value, big-step variant), π-calculus (as an unlabelled reduction system), and differential λ-calculus.

Finally, in §2.3, we organise transition monads into categories. For the category of transition monads over a fixed triple (T, S_1, S_2), we take the slice category of T-modules alluded to above, and we wrap together all these 'little' slice categories into what we call a *record* category of transition monads.

We then proceed to the main concern of this work: the specification of transition monads via suitable signatures.

For this, we start in §3 by proposing a new, abstract notion of *semantic signature* over a category $\mathbf{C}$. A semantic signature $S = (\mathbf{E}, U)$ over $\mathbf{C}$ consists of a category $\mathbf{E}$ of *algebras*, together with a *forgetful* functor $U\colon \mathbf{E} \to \mathbf{C}$, such that $\mathbf{E}$ has an initial object $S^{\circledast}$: we think of such a semantic signature as *specifying* the object $S^* := U(S^{\circledast})$ underlying the initial algebra. For instance, if $\mathbf{C}$ is cocomplete, each finitary endofunctor on $\mathbf{C}$ generates a semantic signature via its algebras. Abstracting over this generating procedure, we introduce *registers* of signatures in §3. A register R for the category $\mathbf{C}$ consists of a class $\mathbf{Sig}_R$ of *signatures*, together with a map associating to each signature S a semantic signature $[\![S]\!]_R$, say $\mathbf{U}_S\colon S\text{-alg} \to \mathbf{C}$. Just as for semantic signatures, ommitting $[\![-]\!]_R$ for readability, we think of a signature S as specifying the object $S^* = \mathbf{U}_S(S^{\circledast})$.

We may now state our goal properly: construct a register for transition monads, containing signatures specifying the desired examples.

Towards this goal, we start in §4 by exploiting Fiore and Hur's *equational systems* [8] to design registers for monads and functors. This will allow us to efficiently specify the base components (T, S_1, S_2) of the desired example transition monads, separately.

We continue in §5 by presenting some general constructions on registers, whose combination will yield a register for transition monads. First, the product construction allows us to group the signatures of T, S_1, and S_2 into a single signature for the triple (T, S_1, S_2).

Then, we introduce in §4.2 a register for a slice category of modules over a monad. This yields a register for transition monads over a fixed triple (T, S_1, S_2), since these form such a slice category. Finally, in §5 we address the task of grouping into a single signature the signatures for the triple (T, S_1, S_2) and for the transition module (R, s, t) over it. For this, we propose a *record* construction for registers, which binds together registers on the base and on the fibers of a record category. Applying this to the previously constructed registers for our base product of three categories and our fibre slice categories of modules, we give in Example 53 our final register for the category of transition monads (with fixed sets of types). This register covers all examples of transition monads from §2.2. We emphasize in particular in §6 the subtle case of differential λ-calculus.

Related work

Beyond the already evoked related work [3, 24, 8], there is a solid body of work on categorical approaches to rewriting with variable binding [13, 16, 1], which only covers transition relations that are stable under arbitrary contexts. Furthermore, Hirschowitz [18] proposes an alternative categorical approach to operational semantics, which is however only equipped with an insufficiently expressive specification technique [17], and has not yet been shown to apply to higher-order languages.

Regarding signatures, some authors [9, 5, 12] use notions of signatures involving some form of type dependency, which may be amenable to describing the dependency of transitions on terms and states. However, to our knowledge, these notions have never been applied to general operational semantics.

Finally, most of the material presented here is extracted from the third author's PhD thesis [19].

Notations

In the following, **Set** denotes the category of sets, $[\mathbf{Set}^P, \mathbf{Set}^Q]$ denotes the locally small category of finitary functors $\mathbf{Set}^P \to \mathbf{Set}^Q$ for any sets P and Q.

The category of finitary monads on $\mathbf{C}$ is denoted by $\mathbf{Mnd}(\mathbf{C})$. Given a monad T on $\mathbf{C}$, the category of $\mathbf{D}$-valued T-modules is denoted by $T\text{-}\mathbf{Mod}(\mathbf{D})$, where we recall [15] that such a T-module consists of a functor $M\colon \mathbf{C} \to \mathbf{D}$ equipped with a right T-action $M \circ T \to M$ satisfying some coherence conditions. If F is a functor $\mathbf{C} \to \mathbf{D}$, we denote by $\overline{F}$ the 'free' $\mathbf{D}$-valued T-module defined by $\overline{F}(c) = F(T(c))$.

For any sequence $p_1, \dots, p_n$ in a set $\mathbb{P}$, for any monad T on $\mathbf{Set}^{\mathbb{P}}$ and $\mathbf{D}$-valued T-module M, we denote by $M^{(p_1,\dots,p_n)}$ the D-valued T-module defined by $M^{(p_1,\dots,p_n)}(X) = M(X + \mathbf{y}_{p_1} + \dots + \mathbf{y}_{p_n})$, where $\mathbf{y}\colon \mathbb{P} \to \mathbf{Set}^{\mathbb{P}}$ is the Yoneda embedding, i.e., $\mathbf{y}_p(q) = 1$ if $p = q$ and $\varnothing$ otherwise. If $\mathbb{P}$ is a singleton, we abbreviate this to $M^{(n)}$.

2 Transition monads

2.1 Definition of transition monads

Definition of transition monads In this section, we introduce the main new mathematical notion of the paper, transition monads, which was already motivated by the case of call-by-value, simply-typed λ-calculus in §1. The notion of transition monad is quite elaborate. We first describe the various components of a transition monad. Then we give the monadic definition. And finally we give a modular description, which is better suited for later use.

Placetakers and states In standard λ-calculus, we have terms, variables are placeholders for terms, and reductions relate a source term to a target term. In a general transition monad we still have variables and reductions, but placetakers for variables and endpoints of reductions can be of a different nature, which we phrase as follows: variables are placeholders for *placetakers*, while reductions relate a *source state* with a *target state*.

The categories for placetakers and for states In standard λ-calculi, we have a set $\mathbb{T}$ of types for terms (and variables); for instance in the untyped version, $\mathbb{T}$ is a singleton. Accordingly, terms form a *monad* on the category $\mathbf{Set}^{\mathbb{T}}$.

Similarly, in a general transition monad we have a set $\mathbb{P}$ of placetaker types, and a set $\mathbb{S}$ of state types. And at least placetakers form a *monad* on the category $\mathbf{Set}^{\mathbb{P}}$.

▸ Notation 1. In the following, $\mathbb{P}$ denotes a (fixed) set of placetaker types. Similarly, $\mathbb{S}$ denotes a (fixed) set of state types.

The object of variables In our (monadic) view of the untyped λ-calculus, there is a (variable!) set of variables and everything is parametric in this 'variable set'. Similarly, in a general transition monad, there is a 'variable object' V in $\mathbf{Set}^{\mathbb{P}}$ and everything is functorial in this variable object. In particular, we have a placetaker object $T(V)$ in $\mathbf{Set}^{\mathbb{P}}$ and a source (resp. target) state object in $\mathbf{Set}^{\mathbb{S}}$, both depending upon the variable object.

The state functors S_1 and S_2 While in the λ-calculus, states are just the same as placetakers, in a general transition monad, they may differ, and more precisely the two state objects are derived from the placetaker object by applying the *state functors* $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$.

The reductions In standard λ-calculi, there is a (typed!) set of reductions, which yields a graph on the set of terms. That is to say, if V is the variable object, and $LC(V)$ the term object, there is a reduction object $Red(V)$ equipped with two morphisms $src, trg \colon Red(V) \to LC(V)$. Note that we consider 'proof-relevant' reductions here, in the sense that two different reductions have the same source and target.

In a general transition monad R, we still have the variable object V in $\mathbf{Set}^{\mathbb{P}}$ and the corresponding object of placetakers $T_R(V)$ also in $\mathbf{Set}^{\mathbb{P}}$, while the reduction object $Red_R(V)$ and the two state objects $S_1(T_R(V))$ and $S_2(T_R(V))$ live in $\mathbf{Set}^{\mathbb{S}}$ so that src and trg form a span $S_1(T_R(V)) \leftarrow Red_R(V) \to S_2(T_R(V))$.

The S-graph of reductions Now we rephrase the previous status of reductions in terms of a graph-like notion which we call S-graph: here $S \coloneqq (S_1, S_2)$ is the pair of state functors. In the untyped λ-calculus, $Red(V)$ and the maps src and trg turn the term object $LC(V)$ into a graph (which depends functorially on the variable object V).

For an analogous statement in a general transition monad, we will use the following notion:

▸ **Definition 2.** *For any pair $S = (S_1, S_2)$ of functors $\mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, an* S-graph *over an object $V \in \mathbf{Set}^{\mathbb{P}}$ consists of*

- *an object E (of* edges*) in $\mathbf{Set}^{\mathbb{S}}$, and*
- *a span $S_1(V) \leftarrow E \to S_2(V)$, which we alternatively view as a morphism $\partial \colon E \to S_1(V) \times S_2(V)$.*

Now we can say that in a general transition monad, reductions form an S-graph over the placetaker object (the whole thing depending upon the variable object…).

The category of S-graphs A reduction monad [3] (in particular the untyped λ-calculus) is just a monad relative to the 'discrete graph' functor from sets to graphs. In order to have a similar definition for transition monads, the last missing piece is the category of S-graphs, which we now describe. A morphism $G \to G'$ of S-graphs consists of a morphism for vertices $f \colon V_G \to V_{G'}$ together with a morphism for edges $f \colon E_G \to E_{G'}$ making the following diagram commute.

$$\begin{array}{ccc} E_G & \xrightarrow{\quad g \quad} & E_{G'} \\ {\scriptstyle \partial_G}\downarrow & & \downarrow{\scriptstyle \partial_{G'}} \\ S_1(V_G) \times S_2(V_G) & \xrightarrow[S_1(f) \times S_2(f)]{} & S_1(V_{G'}) \times S_2(V_{G'}) \end{array}$$

▸ **Proposition 3.** *For any pair $S = (S_1, S_2)$ of functors $\mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, S-graphs form a category S-**Gph**.*

Monadic definition of transition monad Now we are ready to deliver a first, monadic definition of transition monad.

▸ **Definition 4.** *A transition monad consists of*
- *two finitary functors* $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$*, and*
- *a finitary monad relative to the functor* J_S *for* $S = (S_1, S_2)$*, mapping an object* V *to the* S*-graph* $J_S(V)$ *on* V *with no edges.*

Let us recall briefly that a relative monad consists of
- an object mapping $T \colon \mathbf{ob}(\mathbf{Set}^{\mathbb{P}}) \to \mathbf{ob}(S\text{-}\mathbf{Gph})$, together with
- morphisms $J_S(X) \to T(X)$, saying that variables in X are vertices of $T(X)$, and
- for each morphism $f \colon J_S(X) \to T(Y)$, morally mapping variables in X to vertices in $T(Y)$, a lifting $f^\star \colon T(X) \to T(Y)$, which provides substitution for vertices and transitions at the same time.

▸ Remark 5. There is a full, reflective subcategory category $S\text{-}\mathbf{Rel} \hookrightarrow S\text{-}\mathbf{Gph}$ consisting of subobjects $E \hookrightarrow S_1(V) \times S_2(V)$. So because relative monads are stable under composition with left adjoints, transition monads map to a proof-irrelevant variant, which is perhaps closer to most of the literature. We stick to the proof-relevant definition for simplicity.

Modular definition of transition monad The monadic definition just given does not mention explicitly one crucial feature we had mentioned earlier: the monad of placetakers. In order to clarify this point, we give an alternative 'modular' definition.

▸ **Definition 6.** *A* modular transition monad *over* $(\mathbb{P}, \mathbb{S})$ *consists of*
- *two finitary functors* $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$
- *a finitary monad* T *on* $\mathbf{Set}^{\mathbb{P}}$*, called the* placetaker *monad,*
- *a finitary* T*-module* R*, called the* transition *module,*
- *a* source T*-module morphism* $s_1 \colon R \to S_1 \circ T$*,*
- *a* target T*-module morphism* $s_2 \colon R \to S_2 \circ T$*.*

This is the definition that we use in the following.

▸ **Proposition 7.** *Modular and monadic transition monads are in one-to-one correspondence.*

Proof. The proof consists merely in unfolding and comparing the definitions, considering T-modules as functors from the Kleisli category of T. ◂

2.2 Examples of transition monads

2.2.1 $\overline{\lambda}\mu$-calculus

Let us start with an example with several placetaker types. Consider the $\overline{\lambda}\mu$-calculus of [14]. Its grammar is given by

Processes	Programs	Stacks
$c ::= \langle e \vert \pi \rangle$	$e ::= x \mid \mu\alpha.c \mid \lambda x.e$	$\pi ::= \alpha \mid e \cdot \pi,$

where α and x range over two disjoint sets of variables, called *stack* and *program* variables respectively. Both constructions μ and λ bind their variable in the body. There are two reduction rules: $\langle \mu\alpha.c \vert \pi \rangle \to c[\alpha \mapsto \pi]$ $\langle \lambda x.e \vert e' \cdot \pi \rangle \to \langle e[x \mapsto e'] \vert \pi \rangle$.

Let us show how this calculus gives rise to a transition monad. First of all, there are two placetaker types, for programs and stacks, so $\mathbb{P} = 2 = \{\mathbf{p}, \mathbf{s}\}$. A variable object is an element of $\mathbf{Set}^{\mathbb{P}}$, that is, a pair of sets: one gives the available free program variables, and the other the available free stack variables. The syntax may be viewed as a monad $T \colon \mathbf{Set}^2 \to \mathbf{Set}^2$:

given a variable object $X = (X_{\mathbf{p}}, X_{\mathbf{s}}) \in \mathbf{Set}^2$, the placetaker object $(T(X)_{\mathbf{p}}, T(X)_{\mathbf{s}}) \in \mathbf{Set}^2$ consists of the sets of program and stack terms with free variables in X. As usual, monad multiplication is given by substitution.

For transitions, source and target states are processes, so there is only one state type: $\mathbb{S} = 1$. Furthermore, processes are pairs of a program and a stack, so that, setting $S_1(A) = S_2(A) = A_{\mathbf{p}} \times A_{\mathbf{s}}$, we get $S_1(T(X)) = T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}$ as desired. Finally, transitions with free variables in X form a graph with vertices in $T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}$, which we model as a pair of functions $\partial_X \colon R(X) \to (T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}})^2$. This family is natural in X and commutes with substitution, hence forms a T-module morphism. We thus have a transition monad.

2.2.2 The π-calculus

For an example involving equations on placetakers, let us recall the following simple variant of π-calculus [22]. The syntax for *processes* is given by $P, Q ::= 0 \mid (P|Q) \mid \nu a.P \mid \overline{a}\langle b\rangle.P \mid a(b).P$, where a and b range over *channel names*, and b is bound in $a(b).P$. Processes are considered equivalent up to *structural congruence*, the smallest equivalence relation $\equiv$ stable under context and satisfying $0|P \equiv P \quad P|Q \equiv Q|P \quad P|(Q|R) \equiv (P|Q)|R \quad (\nu a.P)|Q \equiv \nu a.(P|Q)$, where in the last equation a should not occur free in Q. Reduction is then given by the inductive rules $\dfrac{}{\overline{a}\langle b\rangle.P|a(c).Q \longrightarrow P|(Q[c \mapsto b])} \qquad \dfrac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \qquad \dfrac{P \longrightarrow Q}{\nu a.P \longrightarrow \nu a.Q}$.

The π-calculus gives rise to a transition monad as follows. Again, we consider two placetaker types, one for channels and one for processes. Hence, $\mathbb{P} = 2 = \{\mathbf{c}, \mathbf{p}\}$. Then, the syntax may be viewed as a monad $T \colon \mathbf{Set}^2 \to \mathbf{Set}^2$: given a variable object $X = (X_{\mathbf{c}}, X_{\mathbf{p}}) \in \mathbf{Set}^2$, the placetaker object $T(X) = (X_{\mathbf{c}}, T(X)_{\mathbf{s}}) \in \mathbf{Set}^2$ consists of the sets of channels and processes with free variables in X. Note that $T(X)_{\mathbf{c}} = X_{\mathbf{c}}$ as there is no operation on channels.

Reductions relate processes, so we take $\mathbb{S} = 1$ and $S_1(X) = S_2(X) = X_{\mathbf{p}}$. Transitions are stable under substitution, hence form a transition monad.

2.2.3 Positive GSOS rules

An example involving labelled transitions (and $S_1 \neq S_2$) is given by Positive GSOS rules [6]. They specify labelled transitions $e \xrightarrow{a} f$. For any set O of operations with arities in $\mathbb{N}$, Positive GSOS rules have the shape $\dfrac{x_i \xrightarrow{a_{i,j}} y_{i,j}}{op(x_1, \ldots, x_n) \xrightarrow{c} e}$, where the variables x_i and $y_{i,j}$ are all distinct, $op \in O$ is an operation with arity n, and e is an expression potentially depending on all the variables.

This yields a transition monad with $\mathbb{P} = 1$, because we are in an untyped setting, and $\mathbb{S} = 1$ because states are terms. The syntax is given by the term monad T on $\mathbf{Set}$. For transitions, in order to take labels into account, we take $S_1(X) = X$ and $S_2(X) = \mathbb{A} \times X$, where $\mathbb{A}$ denotes the set of labels. Transitions thus form a subset of $X \times (\mathbb{A} \times X)$ as desired.

2.2.4 Differential λ-calculus

The differential λ-calculus [7] provides a further example with $S_1 \neq S_2$. Its syntax may [25, §6] be defined by

$$\begin{array}{rcll} e, f & ::= & x \mid \lambda x.e \mid e\ U \mid De \cdot f & \text{(terms)} \\ U, V & ::= & \langle e_1, \ldots, e_n \rangle & \text{(multiterms)}, \end{array}$$

where $\langle e_1, \ldots, e_n \rangle$ denotes a (possibly empty) multiset, i.e., the ordering is irrelevant.

Reductions relate terms to multiterms, and is based on two intermediate notions:

1. *Unary multiterm substitution* $e[x \mapsto U]$ in a term e, where a term variable x is replaced with a multiterm U, and which returns a multiterm (not to be confused with the monadic substitution, which handles the particular case where U is a mere term).
2. *Partial derivation* $\frac{\partial e}{\partial x} \cdot U$ of a term e w.r.t. a term variable x along a multiterm U. This again returns a multiterm.

Both are defined by induction on e and induce T-module morphisms $T^{(1)} \times ! \circ T \to ! \circ T$, for the term monad T on **Set** underlying the transition monad, where $!$ is the functor sending a set X to the set of (finite) multisets over X.

Unary multiterm substitution and partial derivation are used to define the reduction relation as the smallest relation closed under context and satisfying the rules below where capture-avoiding substitution is defined by induction as usual.

$$(\lambda x.e)\ U \to e[x \mapsto U] \qquad\qquad D(\lambda x.e) \cdot f \to \lambda x.\left(\frac{\partial e}{\partial x} \cdot f\right)$$

The second rule relies on the abbreviation $\lambda x.\langle e_1, \ldots, e_n\rangle \coloneqq \langle \lambda x.e_1, \ldots, \lambda x.e_n\rangle$. We work with $\mathbb{P} = \mathbb{S} = 1$, i.e., only one placetaker and state type; S_1 is the identity; and $S_2 = !$.

Reduction is stable under substitution by terms, hence we again have a transition monad.

2.2.5 Call-by-value, simply-typed λ-calculus, big-step style

Let us finally organise simply-typed, call-by-value, big-step λ-calculus into a transition monad. The subtlety lies in the fact that variables are only placeholders for values.

Because variables and values are indexed by simple types, we take $\mathbb{P} = \mathbb{S}$ to be the set of simple types (generated from some fixed set of type constants). The monad T over $\mathbf{Set}^{\mathbb{P}}$ is then given by values: given a variable object $X \in \mathbf{Set}^{\mathbb{P}}$, the placetaker object $T(X) \in \mathbf{Set}^{\mathbb{P}}$ assigns to each simple type τ the set $T(X)_\tau$ of values of type τ taking free (typed) variables in X.

In big-step semantics, reduction relates terms to values. Hence, we are seeking state functors $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$ such that $S_1(T(X))_\tau$ is the set of λ-terms of type τ with free variables in X, and $S_2(T(X))_\tau$ is the set of values. For S_2, we should clearly take the identity functor. For S_1, we first observe that λ-terms can be described as *application* binary trees whose leaves are values (internal nodes being typed applications). Thus, we define $S_1(X)_\tau$ to be the set of application binary trees of type τ with leaves in X.

Finally, reduction is stable under value substitution, so we obtain a transition monad.

2.3 Categories of transition monads

Our goal in the sequel is to generate the example transition monads of the previous section from more basic data. For this, we follow the recipe of initial semantics; this requires as input a category of *models* and outputs the carrier of the initial model (of course, the existence of an initial model is also required). In order to do this for transition monads, we need to organise them into a category. We start with a particular case.

▸ **Definition 8.** *For any sets* $\mathbb{P}$ *and* $\mathbb{S}$*, monad* T *over* $\mathbf{Set}^{\mathbb{P}}$*, and functors* $S_1, S_2 \colon \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$*, let* $\mathbf{OMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ *denote the slice category* T*-***Mod**$/(S_1 \circ T) \times (S_2 \circ T)$.

This gives a first family of categories of transition monads, that we will integrate through a simple construction[1]:

[1] There is a more comprehensive construction, obtained by observing that the assignment $(T, S_1, S_2) \mapsto \mathbf{OMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ forms a pseudofunctor and applying the so-called Grothendieck construction.

▸ **Definition 9.** *A* record category K *is a category of the form* $\sum_b \mathbf{P}_b$ *where* b *ranges over the objects of a* base category $\mathbf{B}_K$*, and each* $\mathbf{P}_b$*, called the* fibre over b*, is a category. In other words, it is given by a (base) category* $\mathbf{B}_K$ *equipped with a map* $\mathbf{P}\colon \mathbf{ob}(\mathbf{B}_K) \to \mathbf{CAT}$.

The relevant example for the present work is the following.

▸ **Definition 10.** *Given two sets* $\mathbb{P}$ *and* $\mathbb{S}$*, let* $\mathbf{OMnd}_{\mathbb{P},\mathbb{S}}$ *denote the record category* $\mathbf{OMnd}_{\mathbb{P},\mathbb{S}}$ *of transition monads with* $\mathbb{P}$ *and* $\mathbb{S}$ *as sets of types for placetakers and states:*

- *it has as its base category the product* $\mathbf{Mnd}(\mathbf{Set}^{\mathbb{P}})\times[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]^2$ *of the category of monads on* $\mathbf{Set}^{\mathbb{P}}$ *with two copies of the functor category* $[\mathbf{Set}^{\mathbb{P}},\mathbf{Set}^{\mathbb{S}}]$*;*
- *the fibre over a triple* (T, S_1, S_2) *is the category* $\mathbf{OMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ *of Definition 8.*

3 Semantic signatures and registers

The rest of the paper is devoted to the specification of transition monads via suitable signatures. More concretely, each of our example transition monads may be characterised as underlying the initial object in some suitable category of models.

We start in §3.1-3.2 by introducing a general notion of semantic signature over a category. In §3.3, we define registers of signatures as families of distinguished semantic signatures. Our main goal (achieved in Example 58) consists in proposing a register for the category of transition monads.

3.1 Semantic signatures

Our notion of semantic signature is an abstract counterpart of usual signatures.

▸ **Definition 11.** *A* semantic signature S *over a given category* $\mathbf{C}$ *consists of*

- *a category* S-alg of models of S (or algebras)*, which admits an initial object, denoted by* $S^{\circledast}$*, and*
- *a* forgetful *functor* $\mathbf{U}_S\colon S\text{-alg} \to \mathbf{C}$.

▸ **Terminology 12.** *Given a semantic signature* S *over a category* $\mathbf{C}$*, we say that* S *is a* signature for $S^* \coloneqq \mathbf{U}_S(S^{\circledast})$*, or alternatively that* S specifies S^*.

▸ Notation 13. When convenient, we introduce a semantic signature over $\mathbf{C}$ as $u\colon \mathbf{E} \to \mathbf{C}$, to be understood as the semantic signature S with S-alg $\coloneqq \mathbf{E}$ and $\mathbf{U}_S \coloneqq u$.

▸ **Example 14.** For a given category $\mathbf{C}$, an object $c \in \mathbf{C}$ is always specified by the following signatures:

- the functor $1 \to \mathbf{C}$ mapping the only object of the final category (with one object and one morphism) to c;
- the codomain functor $c/\mathbf{C} \to \mathbf{C}$ from the coslice category.

▸ **Example 15.** Consider the standard endofunctor $F\colon \mathbf{Set} \to \mathbf{Set}$ with $F(X) = X + 1$. We define a semantic signature for which the category of models is the category of F-algebras, and the forgetful functor sends any F-algebra to its carrier. In order to complete the definition of this example, we should prove that the category of F-algebras has an initial object. This is well-known and the carrier of the initial model is $\mathbb{N}$.

▸ Notation 16. We denote by $UR_{\mathbf{C}}$ the class of semantic signatures over the category $\mathbf{C}$ (UR stands for 'universal register', as later justified by Definition 20, Section 3.3).

3.2 The external product of semantic signatures

A first basic construction of semantic signatures is for a product of categories. The application we have in mind is the product category $\mathbf{Mnd}(\mathbf{Set}^{\mathbb{P}}) \times [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]^2$ (Definition 10), which is the base category of our record category of transition monads.

▸ **Lemma 17.** *Given a set I and functors $U_i \colon \mathbf{E}_i \to \mathbf{C}_i$ for $i \in I$, if each $\mathbf{E}_i$ has an initial object, then so does the product $\prod_i \mathbf{E}_i$.*

▸ **Definition 18.** *Given a family $(\mathbf{C}_i)_{i\in I}$ of categories, and a corresponding family of semantic signatures $u_i \colon E_i \to \mathbf{C}_i$, the product $\prod_i u_i \colon \prod_i E_i \to \prod_i \mathbf{C}_i$ is a semantic signature. This defines our external product of signatures $\prod_I \colon \prod_i UR_{\mathbf{C}_i} \to UR_{\prod_i \mathbf{C}_i}$.*

3.3 Registers of signatures

In this section, we introduce *registers* of signatures for a category $\mathbf{C}$, which are (possibly large) families of semantic signatures over $\mathbf{C}$. Roughly speaking, each register allows to write down specific signatures, gives the recipe for the corresponding semantic signature, hence, last but not least, ensures (once and for all) that there is an initial model.

▸ **Definition 19.** *A* register *R for a given category $\mathbf{C}$ consists of*
- *a class $\mathbf{Sig}_R$ (*of signatures*), and*
- *a map $[\![-]\!]_R \colon \mathbf{Sig}_R \to UR_{\mathbf{C}}$.*

We can now motivate the notation UR_C above:

▸ **Definition 20.** *For a given category $\mathbf{C}$, the* universal register *$UR_{\mathbf{C}}$ is defined as follows:*
- *its signatures are semantic signatures for $\mathbf{C}$, and*
- *the map $[\![-]\!]_{UR_{\mathbf{C}}}$ is the identity (on $UR_{\mathbf{C}}$).*

▸ Notation 21. When convenient, we introduce a register as $u \colon S \to UR_{\mathbf{C}}$ to be understood as the register R with $\mathbf{Sig}_R \coloneqq S$ and $[\![-]\!]_R \coloneqq u$. Moreover, we sometimes omit $[\![-]\!]_{UR_{\mathbf{C}}}$, thus identifying any $s \in \mathbf{Sig}_R$ with its associated semantic signature $[\![s]\!]_R$.

We can now translate the slogan *Endofunctors are signatures* with a register, using a well-known initiality result [21, §2, Theorem]

▸ **Definition 22.** *For a given cocomplete category $\mathbf{C}$, the* universal endofunctorial register *$UEF_{\mathbf{C}}$ is defined as the map $[\mathbf{C}, \mathbf{C}] \to UR_{\mathbf{C}}$ sending any finitary endofunctor F to the forgetful functor F-alg $\to \mathbf{C}$ from its category of algebras.*

Let us now define simple constructions on registers.

▸ **Proposition 23.** *For any register R for $\mathbf{C}$ and functor $F \colon \mathbf{C} \to \mathbf{D}$, postcomposition of semantic signatures with F induces a register $\sum_F(R)$ for $\mathbf{D}$.*

▸ **Example 24.** As an easy application, by Remark 5, any register R for transition monads induces one for the proof-irrelevant variant.

▸ **Proposition 25.** *For any register R for $\mathbf{C}$ and map $f \colon \mathbf{S} \to \mathbf{Sig}_R$, precomposition with f induces a register $\Delta_f(R)$ for $\mathbf{C}$ whose signatures are elements of $\mathbf{S}$.*

Here is an important application.

▸ **Definition 26.** *We deem* endofunctorial *all registers of the form $\Delta_f(UEF_{\mathbf{C}})$, for some map $f \colon \mathbf{S} \to \mathbf{Sig}_{UEF_{\mathbf{C}}}$.*

A useful fact is that endofunctorial registers are closed under the family construction, as follows.

▸ **Definition 27.** *For any endofunctorial register $R = \Delta_f(UEF_{\mathbf{C}})$, let R^* denote the endofunctorial register whose signatures are families of signatures in $\mathbf{Sig}_R$, and whose semantics maps any family to the coproduct of associated endofunctors.*

A final basic construction on registers is the external product.

▸ **Definition 28.** *The* product *of a family $(u_i \colon S_i \to UR_{\mathbf{C}_i})_{i\in I}$ of registers is obtained by post-composing $\prod_i u_i$ with the product of semantic signatures: $\prod_i S_i \xrightarrow{\prod_i u_i} \prod_i UR_{\mathbf{C}_i} \xrightarrow{\Pi_I} UR_{\prod_i \mathbf{C}_x}$.*

▸ **Example 29.** The product register $OMnd_b \coloneqq \mathbf{MonReg}(\mathbf{Set}^{\mathbb{P}}) \times EqSys[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]^2$ for monads and state functors allows us to specify the base components of our transition monads.

3.4 Equational registers

In this section, we show that equational systems [8] on any category $\mathbf{C}$ define a register $EqSys(\mathbf{C})$ for $\mathbf{C}$, which refines endofunctorial register $UEF_{\mathbf{C}}$ (Definition 22) with equations. In order to explain them, the starting point is the observation that for any (nice) endofunctor Σ, any term $e \in \Sigma^\star(n)$ for the monad generated by Σ generates a functor Σ-alg $\to (-)^n$-alg preserving the carrier, mapping any $\rho \colon \Sigma(X) \to X$ to the composite

$$X^n \xrightarrow{\langle e^\sharp, \Sigma^\star_{n,X}\rangle} \Sigma^\star(n) \times [\Sigma^\star(n), \Sigma^\star(X)] \xrightarrow{ev} \Sigma^\star(X) \to X,$$

viewed as a $(-)^n$-algebra. Generalizing equations as pairs of terms in $\Sigma^\star(n)$, an equation may be modelled as a pair of such functors.

Furthermore, this technique generalises to families $(t_i = u_i)_i$ of equations, with each $t_i, u_i \in \Sigma^\star(n_i)$, by replacing $(-)^n$ with the coproduct functor $\sum_i (-)^{n_i}$. This works even for the empty family of course.

Equational systems are obtained by abstracting over this situation.

▸ **Definition 30** ([8, Definition 3.3])**.** *For any endofunctors Σ, Γ on a category $\mathbf{C}$, a* metaterm *of type Γ is a functor $L \colon \Sigma$-alg $\to \Gamma$-alg preserving carriers, i.e., such that $\mathbf{U}_\Gamma \circ L = \mathbf{U}_\Sigma$, where $\mathbf{U}_\Sigma$ and $\mathbf{U}_\Gamma$ are the forgetful functors.*

An equational system *$\mathbb{E} = (\Sigma \rhd \Gamma \vdash L = R)$ over $\mathbf{C}$ consists of an endofunctor Γ, together with two metaterms L and R of type Γ.*

▸ **Definition 31.** *Given an equational system $\mathbb{E} = (\mathbf{C} : \Sigma \rhd \Gamma \vdash L = R)$, a model for $\mathbb{E}$, or an* $\mathbb{E}$-algebra*, is a Σ-algebra $\rho \colon \Sigma(X) \to X$ for which $L(X, \rho) = R(X, \rho)$.*

▸ **Lemma 32** (cf. [8, Theorem 5.1])**.** *Let $\mathbb{E} = (\mathbf{C} : \Sigma \rhd \Gamma \vdash L = R)$ be a* well-behaved *equational system, in the sense that $\mathbf{C}$ is locally presentable, and Σ and Γ preserve epimorphisms and colimits of ω-chains. Then the forgetful functor $\mathbf{U}_{\mathbb{E}} \colon \mathbb{E}$-alg $\to \mathbf{C}$ has a left adjoint.*

▸ **Proposition 33.** *For any category $\mathbf{C}$, there is a register $EqSys(\mathbf{C})$ whose signatures are well-behaved equational systems, and which maps any $\mathbb{E}$ to the forgetful functor $\mathbf{U}_{\mathbb{E}} \colon \mathbb{E}$-alg $\to \mathbf{C}$.*

Let us now present equational systems for some state functors from our examples.

▸ **Example 34.** The identity functor on $\mathbf{Set}$ is specified by the equational system on $[\mathbf{Set}, \mathbf{Set}]$ defined by $\Sigma(F)(X) = X$ and no equation. Indeed, algebras are functors F equipped with a natural transformation $X \to F(X)$, i.e., pointed endofunctors. The initial one is thus clearly the identity.

▸ **Example 35.** The state functor $S(X) = X_{\mathbf{p}} \times X_{\mathbf{s}}$ from §2.2.1 is specified by the equational system defined on $[\mathbf{Set}^2, \mathbf{Set}]$ by $\Sigma(F)(X) = X_{\mathbf{p}} \times X_{\mathbf{s}}$ and no equation. Algebras are functors F equipped with a natural transformation $X_{\mathbf{p}} \times X_{\mathbf{s}} \to F(X)$, so the initial algebra is clearly $X_{\mathbf{p}} \times X_{\mathbf{s}}$ itself.

▸ **Example 36.** The first state functor of call-by-value, simply-typed λ-calculus could be specified by taking $\Sigma(F)(X) = S_1(X)$, with S_1 as in §2.2.5, and no equation. However, let us observe that it may also be specified by the simpler endofunctor $\Sigma(F)(X)_t = X_t + \sum_{t'} F(X)_{t' \to t} \times F(X)_{t'}$ with no equation.

4 Registers for monads and slice module categories

In this section, we recast results from the literature as registers for functors and monads.

4.1 A register for monads

In order to specify monads through equational systems, we first specify them as endofunctors, and then refine the result into a register for monads.

▸ **Proposition 37.** *For any locally presentable category* $\mathbf{C}$*, finitary monads on* $\mathbf{C}$ *are the algebras of an equational system* $\mathbb{E}_{Mnd}(\mathbf{C}) = (\Sigma_{Mnd} \rhd \Gamma_{Mnd} \vdash L_{Mnd} = R_{Mnd})$ *over* $[\mathbf{C}, \mathbf{C}]$.

Proof. This is a particular case of [8, §3.3(4)]. Briefly, we take $\Sigma_{Mnd}(F) = \mathrm{Id}_{\mathbf{C}} + F \circ F$ to specify the unit and multiplication, and then encode the monad equations. ◂

In order to apply this for specifying our examples, we augment the endofunctor Σ with arities for the relevant operations.

▸ **Example 38.** For pure λ-calculus, the relevant endofunctor on $[\mathbf{Set}, \mathbf{Set}]$ is $\Sigma(F)(X) = X + F(F(X)) + F(X)^2 + F(X+1)$.

We furthermore encode the substitution rules for each operation as an equation.

▸ **Example 39.** We enforce the usual equation $(M\ N)[\sigma] = M[\sigma]\ N[\sigma]$ through the equation $L = R \colon \Sigma\text{-alg} \to \Gamma\text{-alg}$, where $\Gamma(F)(X) = F(F(X))^2$, and the structure maps of $L(\Sigma(F) \xrightarrow{\rho} F)$ and $R(\Sigma(F) \xrightarrow{\rho} F)$ at X are respectively: $F(F(X))^2 \xrightarrow{@_{F(X)}} F(F(X)) \xrightarrow{\mu_X} F(X)$ and $F(F(X))^2 \xrightarrow{\mu^2} F(X)^2 \xrightarrow{@_X} F(X)$. The maps $@$ and μ follow from the Σ-algebra structure ρ on F. E.g., $@_X$ is defined as the composite $F(X)^2 \xhookrightarrow{in_3} \Sigma(F)(X) \xrightarrow{\rho_X} F(X)$.

Finally, if needed, we further encode the remaining equations, such as $P|(Q|R) \equiv (P|Q)|R$ in π-calculus, as abstract equations.

Let us now describe the general pattern, by defining a register for monads. The idea is that a signature should be an equational system on endofunctors of the considered category $\mathbf{C}$ whose operation endofunctor Σ contains Σ_{Mnd}, and whose equations include the monad equations. One way of enforcing this consists in asking the endofunctors to have the shape $\Sigma_{Mnd} + \Sigma$ and $\Gamma_{Mnd} + \Gamma$. For equations, we rely on the following well-known fact.

▸ **Proposition 40.** *For all endofunctors* F, G *on* $\mathbf{C}$*,* $(F+G)$*-alg is a pullback of* F*-alg and* G*-alg over* $\mathbf{C}$*.*

Thus, given functors $L_1 \colon \mathbf{D} \to F\text{-alg}$ and $L_2 \colon \mathbf{D} \to G\text{-alg}$ mapping objects and morphisms of $\mathbf{D}$ to the same underlying objects and morphisms of $\mathbf{C}$, we may form their pairing $\langle L_1, L_2 \rangle \colon \mathbf{D} \to (F+G)\text{-alg}$. Denoting by $\downarrow_F$ the forgetful functor $(F+G)\text{-alg} \to F\text{-alg}$, we state:

▸ **Definition 41.** *A* monadic signature on $\mathbf{C}$ *is an equational system on* $[\mathbf{C},\mathbf{C}]$ *extending* $\mathbb{E}_{Mnd}$*, i.e., that has the shape* $(\Sigma_{Mnd} + \Sigma) \rhd (\Gamma_{Mnd} + \Gamma) \vdash \langle L_{Mnd} \circ \downarrow_{\Sigma_{Mnd}}, L \rangle = \langle R_{Mnd} \circ \downarrow_{\Sigma_{Mnd}}, R \rangle$.

Because monadic signatures extend $\mathbb{E}_{Mnd}$, their models are, in particular, monads:

▸ **Proposition 42.** *Given a monadic signature* $\mathbb{E}$ *on* $\mathbf{C}$*, the forgetful functor* $\mathbb{E}\text{-alg} \to [\mathbf{C},\mathbf{C}]$ *factors through* $\mathbf{Mnd}(\mathbf{C}) \to [\mathbf{C},\mathbf{C}]$*. Thus, any monadic signature defines a semantic signature on* $\mathbf{Mnd}(\mathbf{C})$*.*

▸ **Definition 43.** *We define the monadic register* $\mathbf{MonReg}(\mathbf{C})$ *for* $\mathbf{Mnd}(\mathbf{C})$ *consisting of monadic signatures.*

4.2 Registers for slice module categories

In this section, we fix two sets $\mathbb{P}$ and $\mathbb{S}$, a monad T on $\mathbf{Set}^{\mathbb{P}}$, and a $\mathbf{Set}^{\mathbb{S}}$-valued T-module M. We then define an endofunctorial register $\mathbf{Rule}^{*}_{T,M}$ for the category $T\text{-}\mathbf{Mod}/M$. Later on, we will use this register with $M \coloneqq (S_1 \circ T) \times (S_2 \circ T)$, i.e., for the category of transition monads over (T, S_1, S_2).

The naive register We start by defining a much simpler endofunctioral sub-register, $NRule_{T,M}$. A signature in $NRule_{T,M}$ consists of

- a *metavariable* module V,
- a *conclusion* module morphism $t \colon V \to M_\tau$ for some *conclusion* state type $\tau \in \mathbb{S}$, and
- a list of *premise* module morphisms of the form $s \colon V \to M_\sigma$, for some *premise* state types $\sigma \in \mathbb{S}$.

▸ **Example 44.** For the left application congruence rule of pure λ-calculus $\dfrac{e \to e'}{e\ f \to e'\ f}$, there are three metavariables e, e', and f, so the metavariable module V is T^3. The conclusion and premise are respectively defined as the module morphisms

$$\begin{array}{rcl} T^3 & \to & T^2 \\ (e,e',f) & \mapsto & (e\ f, e'\ f) \end{array} \quad \text{and} \quad \begin{array}{rcl} T^3 & \to & T^2 \\ (e,e',f) & \mapsto & (e,e'). \end{array}$$

The endofunctor Σ_S associated to any signature $S \coloneqq (\tau, V, t, (\sigma_i, s_i)_{i \in n})$ is a composite

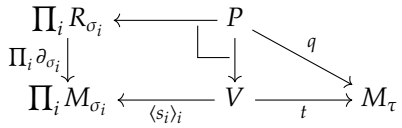
$$\begin{array}{ccccc} T\text{-}\mathbf{Mod}(\mathbf{Set})/\prod_i M_{\sigma_i} & \xrightarrow{\Delta_{\langle s_i \rangle_i}} & T\text{-}\mathbf{Mod}(\mathbf{Set})/V & \xrightarrow{\Sigma_t} & T\text{-}\mathbf{Mod}(\mathbf{Set})/M_\tau \\ \Pi_i(-)_{\sigma_i} \uparrow & & & & \downarrow \\ T\text{-}\mathbf{Mod}(\mathbf{Set}^{\mathbb{S}})/M & & & & T\text{-}\mathbf{Mod}(\mathbf{Set}^{\mathbb{S}})/M, \end{array}$$

of four functors, where

- $\prod_i (\partial \colon R \to M)_{\sigma_i}$ denotes $\prod_i \partial_{\sigma_i} \colon \prod_i R_{\sigma_i} \to \prod_i M_{\sigma_i}$,
- $\Delta_{\langle s_i \rangle_i}$ is defined by pullback along the tupling $\langle s_i \rangle_i \colon V \to \prod_i M_{\sigma_i}$ of all premises,
- $\sum_i$ is defined by postcomposition with the conclusion $t \colon V \to M_\tau$.

The last functor is the canonical embedding, which maps any $R \to M_\tau$ to $R \cdot \mathbf{y}_\tau \to M$, where $R \cdot \mathbf{y}_\tau$ is defined for all X by $(R \cdot \mathbf{y}_\tau)(X)_\tau = R(X)$ and $(R \cdot \mathbf{y}_\tau)(X)_\sigma = \varnothing$ for $\sigma \neq \tau$.

▸ Remark 45. The embedding $(-)\cdot \mathbf{y}_\tau$ is left adjoint to evaluation at τ: $(-)\cdot \mathbf{y}_\tau \dashv (-)_\tau$. Thus Σ_S maps any $\partial\colon R \to M$ to the transpose of the right-hand composite q below.



(1)

▸ **Proposition 46.** *The assignment $S \mapsto \Sigma_S$ defines a register $NRule_{T,M}$ for T-$\mathbf{Mod}(Set^{\mathbb{S}})/M$.*

▸ **Example 47.** Consider the endofunctor associated to the left application rule of Example 44. Because $\mathbb{S} = 1$, the functor $(-)\cdot \mathbf{y}_\tau$ is trivial, so the endofunctor maps any $\partial\colon R \to T^2$:

- to the pullback P, where $P(X)$ is the set of 4-tuples $(r, e, e', f) \in R(X) \times T(X)^3$ such that r is a transition $e \to e'$,
- with projection to T^2 mapping any (r, e, e', f) to $(e\ f, e'\ f)$.

An algebra is thus such a $\partial\colon R \to T^2$ which, to each such tuple (r, e, e', f) associates a reduction over $(e\ f, e'\ f)$, as desired.

Binding rule registers Let us now refine the naive rules of the previous section. The motivation lies in rules whose premises have additional free variables.

▸ **Example 48.** Consider the ξ rule of pure λ-calculus: $\dfrac{e \to f}{\lambda x.e \to \lambda x.f}$,

The metavariable and conclusion may remain the same; the problem is with the premise, which cannot be a morphism $V \to T^2$, but should rather have type $V \to T^{(1)} \times T^{(1)}$. We thus generalise $NRule_{T,M}$ to let them have premises of this shape:

▸ **Definition 49.** *The register $Rule_{T,M}$ is defined by:*

- *signatures are just as in $NRule_{T,M}$, except that the premises now have the shape $s\colon V \to M_\sigma^{(\vec{p})}$, for $\sigma \in \mathbb{S}$ and $\vec{p}$ a list of placetaker types; and*
- *the semantics is defined exactly as for naive rules, replacing $\prod_i R_i$ with $\prod_i R_i^{(\vec{p^i})}$.*

Registers from families of binding rules Recalling Definition 27, we obtain:

▸ **Proposition 50.** *Families of binding (T, M)-rules (over potentially different types) are the signatures of a register $Rule^*_{T,M}$.*

▸ **Example 51.** The ξ rule is specified by the binding rule with metavariable module given by $(T^{(1)})^2$, whose conclusion is $\lambda^2\colon (T^{(1)})^2 \to T^2$, and whose premise is the identity.

5 Record registers

The construction on registers introduced in the previous sections allow us to design registers for the various components of our transition monad, separately: we may specify the underlying monad T and state functors S_1 and S_2 using signatures from the registers for functors and monads previously defined. We may even assemble these signatures into a single signature Σ for the product register of Definition 28. Then, we may specify the desired transition monad as an object of the fibre $\mathbf{OMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$, using a family $\mathbf{R}$ of binding (T, M)-rules from Proposition 50, with $M = (S_1 \circ T) \times (S_2 \circ T)$.

In this section, we now want to assemble Σ and $\mathbf{R}$ into a single signature for some compound register for the record category $\mathbf{OMnd}_{\mathbb{P},\mathbb{S}}$. This can be done in general for an arbitrary record category. The input for the construction is the following indexed variant of registers.

▸ **Definition 52.** *An* indexed register (R_b, R_f) *for a record category* $\sum_b \mathbf{P}(b)$*, with* $\mathbf{P}\colon \mathbf{ob}(\mathbf{B}) \to$ **CAT***, consists of*

- *a* base *register* R_b *for* $\mathbf{B}$*, together with,*
- *for each signature* B *in* $\mathbf{Sig}_{R_b}$*, a* fibre *register* $R_f(B)$ *for the fibre* $\mathbf{P}_{B^*}$ *over the initial* B*-algebra.*

▸ **Example 53.** Consider the product register $OMnd_b$ of Example 29 for monads and state functors, and define, for all signatures $\Sigma \in \mathbf{Sig}_{OMnd_b}$, the register $OMnd_f(\Sigma) \coloneqq Rule_{T,\overline{S_1}\times\overline{S_2}}$, where $\Sigma^* = (T, S_1, S_2)$. The pair $OMnd \coloneqq (OMnd_b, OMnd_f)$ forms an indexed register for the record category of transition monads.

From any fixed indexed register $R = (R_b, R_f)$ for K, let us now construct a proper register $\sum R$, which we call the *record* register of R. First of all, let us define the signatures of $\sum R$.

▸ **Definition 54.** *A* signature record for R *is a pair* (B, F) *with* $B \in \mathbf{Sig}_{R_b}$ *and* $F \in \mathbf{Sig}_{R_f(B)}$.

▸ **Example 55.** A signature record for the indexed register $OMnd$ from Example 53 consists of a triple $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2)$ of signatures, specifying a monad $T = \Sigma_0^*$ and state functors $S_i = \Sigma_i^*$, $i = 1, 2$, together with a family $\mathbf{R}$ of binding $(T, \overline{S_1} \times \overline{S_2})$-rules.

Finally, we construct the record register $\sum R$ by defining the semantics of signature records.

▸ **Definition 56.** *Given a signature record* (B, F) *for some indexed register for a record category* $K = \sum_b \mathbf{P}_b$*, the semantic signature* $\sum(B, F)$ associated *to* (B, F) *is* F*-*alg $\xrightarrow{\mathbf{U}_F}$ $\mathbf{P}_{B^*} \hookrightarrow K$.

▸ **Proposition 57.** *Given an indexed register* $R = (R_b, R_f)$*, signature records* (B, F) *form the signatures of the* record *register* $\sum R$*, whose models are given by Definition 56.*

We can now achieve our goal and propose a register for transition monads.

▸ **Example 58.** The indexed register $OMnd$ defined in Example 53 induces a register $\sum OMnd$ for the category of transition monads.

6 Applications

All examples from §2.2 may be specified by signatures from the record register $\sum OMnd$ of Example 58. By Example 24, this also holds for the proof-irrelevant variant. For the case of Positive GSOS, we can even define a specific register, whose signatures are Positive GSOS specifications, the semantics being given by interpreting them as signatures for $\sum OMnd$.

In this section, we present in some detail the signature for differential λ-calculus, as a transition monad with $\mathbb{P} = \mathbb{S} = 1$, introduced in §2.2.4. A signature in the register of transition monads consists of two components: a (product) signature for the state functors and monad, given in §6.1, and a signature for the β and ∂-reduction rules. Both are straightforwardly modelled by a signature over as explained in §4.2, but they first require us to construct some intermediate operations $-[x \mapsto -]$ and $\frac{\partial -}{\partial x} \cdot -$. We tackle this task in §6.2.

6.1 State functors and monad of differential λ-calculus

The first state functor is the identity functor $\mathrm{Id}\colon \mathbf{Set} \to \mathbf{Set}$, and thus is specified by the signature of Example 34. The second state functor is !, the multiset functor, and is specified by an equational system $(\Sigma_2 \rhd \Gamma_2 \vdash L_2 = R_2)$ on $[\mathbf{Set}, \mathbf{Set}]$, where $\Sigma_2(F)(X) = X + F(X) \times F(X) + 1$, so that an algebra of Σ_2 is an endofunctor equipped with a binary operation and

a constant. Then Γ_2, L_2, and R_2 are defined so as to enforce commutativity, associativity, and unitality of the constant with respect to the binary operation.

Next, the monad of differential λ-calculus is specified by a monadic equational system $((\Sigma_{Mnd} + \Sigma) \rhd (\Gamma_{Mnd} + \Gamma) \vdash \langle L_{Mnd} \circ \downarrow_{\Sigma_{Mnd}}, L\rangle = \langle R_{Mnd} \circ \downarrow_{\Sigma_{Mnd}}, R\rangle)$ on **[Set, Set]**, which we now define.

We take $\Sigma(T) = T^{(1)} + T \times !T + T \times T$, modelling the operations $\lambda x.-$, $- -$, and $D - \cdot -$. Then, we choose Γ, L, and R so as to enforce that these operations are compatible with monadic substitution, in the sense that they are module morphisms.

The resulting signature specifies a monad (T, η, μ) with a module morphism $\sigma\colon \Sigma(T) \to T$.

6.2 Intermediate constructions for differential λ-calculus

Specifying the reduction rules requires two intermediate constructions: *unary multiterm substitution* $-[x \mapsto -]$, and *partial derivation* $\frac{\partial -}{\partial x} \cdot -$, which we both model as T-module morphisms from $T^{(1)} \times !T \to !T$, or equivalently from $T^{(1)} \to (!T)^{!T}$. [2]

In [25, §6], the underlying maps are defined by induction. Let us briefly upgrade these constructions into T-module morphisms. If the domain was T, then we could exploit the bijection between module morphisms $T \to M$ and $M(1)$, for any module M. More precisely, any element $m \in M(\{*\})$ yields a module morphism $\underline{m}\colon T \to M$, mapping any term $t \in T(X)$ to $m[* \mapsto t] \in M(X)$.

In our case, the domain of the desired morphisms is $T^{(1)}$. We thus propose the following general recipe for building a T-module morphism $T^{(1)} \to M$:

1. provide an element of $M(0)$;
2. equip M with $\Sigma^{\uparrow}$-algebra structure, where $\Sigma^{\uparrow}$ denotes Σ (canonically) viewed as an endofunctor on T-modules;
3. provide an element $m \in M(1)$ such that $\underline{m}\colon T \to M$ is compatible with the Σ-algebra structures of T and M, that is, $\underline{m}$ upgrades into a Σ-algebra morphism.

This recipe relies on the following lemma:

▸ **Lemma 59.** *Let U denote the forgetful functor $(\Sigma+1)^{\uparrow}$-alg $\to \Sigma^{\uparrow}$-alg. Then, $T^{(1)}$, equipped with its canonical structure, is initial in the comma category $T \downarrow U$.*

7 Conclusion and perspectives

We have introduced transition monads as a generalisation of reduction monads, and demonstrated that they cover relevant new examples. We have introduced a register of signatures for specifying them. In future work, we plan on investigating other forms of state modules. E.g., using an arbitrary module covers the subtle labelled transition system for π-calculus. We also consider refining our signatures so as to enlarge the category of models and allow the monad and state functors to vary. Finally, it would be relevant to prove general theorems about the transition monads that we now know how to generate, typically about sufficient conditions for bisimilarity to be a congruence [20].

References

1 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *MSCS*, 26:3–37, 2016.

[2] As a presheaf category, the category of finitary **Set**-valued T-modules has exponentials.

2 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular specification of monads through higher-order presentations. In Herman Geuvers, editor, *Proc. 4th Int. Conf. on Formal Structures for Comp. and Deduction*, LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.

3 Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Reduction monads and their signatures. *PACMPL*, 4(POPL):31:1–31:29, 2020. doi:10.1145/3371099.

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

5 Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek Klin, and Corina Cîrstea, editors, *Proc. 4th Alg. and Coalgebra in Comp. Sci.*, volume 6859 of *LNCS*, pages 70–84. Springer, 2011. doi:10.1007/978-3-642-22944-2_6.

6 Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can't be traced. *J. ACM*, 42(1):232–268, 1995. doi:10.1145/200836.200876.

7 Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *TCS*, 309:1–41, 2003.

8 Marcelo Fiore and Chung-Kil Hur. On the construction of free algebras for equational systems. *TCS*, 410:1704–1729, 2009.

9 Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proc. 23rd Logic in Comp. Sci.*, pages 57–68. IEEE, 2008. doi:10.1109/LICS.2008.38.

10 Marcelo P. Fiore and Sam Staton. A congruence rule format for name-passing process calculi from mathematical structural operational semantics. In *Proc. 21st Logic in Comp. Sci.*, pages 49–58. IEEE, 2006. doi:10.1109/LICS.2006.7.

11 Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *Proc. 16th Logic in Comp. Sci.*, pages 93–104. IEEE, 2001. doi:10.1109/LICS.2001.932486.

12 Richard Garner. Combinatorial structure of type dependency. *CoRR*, abs/1402.6799, 2014. URL: http://arxiv.org/abs/1402.6799, arXiv:1402.6799.

13 Makoto Hamana. Term rewriting with variable binding: An initial algebra approach. In *Proc. 5th Princ. and Practice of Decl. Prog.* ACM, 2003. doi:10.1145/888251.888266.

14 Hugo Herbelin. *Séquents qu'on calcule: de l'interprétation du calcul des séquents comme calcul de lambda-termes et comme calcul de stratégies gagnantes.* PhD thesis, Paris Diderot University, France, 1995. URL: https://tel.archives-ouvertes.fr/tel-00382528.

15 André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In *WoLLIC*, volume 4576 of *LNCS*, pages 218–237. Springer, 2007. doi:10.1007/3-540-44802-0_3.

16 Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-order rewriting. *LMCS*, 9(3), 2013. URL: http://hal.archives-ouvertes.fr/hal-00540205, doi:10.2168/LMCS-9(3:10)2013.

17 Tom Hirschowitz. Cellular monads from positive GSOS specifications. In Jorge A. Pérez and Jurriaan Rot, editors, *Proc. 26th International Workshop on Expressiveness in Concurrency and 16th Workshop on Structural Operational Semantics, EXPRESS/SOS*, volume 300 of *EPTCS*, pages 1–18, 2019. doi:10.4204/EPTCS.300.1.

18 Tom Hirschowitz. Familial monads and structural operational semantics. *PACMPL*, 3(POPL):21:1–21:28, 2019. URL: https://dl.acm.org/citation.cfm?id=3290334.

19 Ambroise Lafont. *Signatures and models for syntax and operational semantics in the presence of variable binding.* PhD thesis, École Nationale Superieure Mines – Telecom

Atlantique Bretagne Pays de la Loire – IMT Atlantique, 2019. URL: https://arxiv.org/abs/1910.09162v2.

20 Andrew M. Pitts. *Howe's Method for Higher-Order Languages*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*, chapter 5. Cambridge University Press, 2011.

21 Jan Reiterman. A left adjoint construction related to free triples. *JPAA*, 10:57–71, 1977.

22 Davide Sangiorgi and David Walker. *The π-calculus - a theory of mobile processes.* Cambridge University Press, 2001.

23 Sam Staton. General structural operational semantics through categorical logic. In *Proc. 23rd Logic in Comp. Sci.*, pages 166–177. IEEE, 2008. doi:10.1109/LICS.2008.43.

24 Daniele Turi and Gordon D. Plotkin. Towards a mathematical operational semantics. In *Proc. 12th Logic in Comp. Sci.*, pages 280–291, 1997. doi:10.1109/LICS.1997.614955.

25 Lionel Vaux. *λ-calcul différentiel et logique classique : interactions calculatoires.* PhD thesis, Université Aix-Marseille 2, 2007.

659 ## Contents

A Proof of Proposition 50

In this section, we show that given any family $(\rho_i)_i$ of binding (T,M)-rules, the coproduct endofunctor $\coprod_i \Sigma_{\rho_i}$ on T-**Mod**$/M$ is finitary, where T is a fixed monad and M a fixed T-module.

As coproducts of finitary functors are finitary, it is enough to show that given one (T,M)-rule ρ, the endofunctor Σ_ρ is finitary. Such a rule comes with a conclusion module morphism $V \to M_\tau$, and a list of premise module morphisms $(V \to M_{\sigma_i}^{(\vec{p}_i)})_{i\in n}$.

Note that $\Sigma_\rho = F \cdot \mathbf{y}_\tau$, where $F(R \xrightarrow{\partial} M)$ is the composite $P \to V \to M_\tau$ in (1). More precisely, let $\mathscr{D}^\rho \colon T_0\text{-}\mathbf{Mod}(\mathbf{Set}^{\mathbf{S}}) \to T_0\text{-}\mathbf{Mod}(\mathbf{Set})$ map any T_0-module W to $\prod_{i\in n} W_{\sigma_i}^{(\vec{p}_i)}$. The functor F is the composite

$$\begin{array}{ccc} T_0\text{-}\mathbf{Mod}(\mathbf{Set})/\mathscr{D}^\rho(M) & \xrightarrow{\Delta_{(\langle p_i\rangle_i)_E}} & T_0\text{-}\mathbf{Mod}(\mathbf{Set})/V \\ {\scriptstyle \mathscr{D}^\rho/M}\uparrow & & \downarrow{\scriptstyle \Sigma_{C_E}} \\ T_0\text{-}\mathbf{Mod}(\mathbf{Set}^{\mathbf{S}})/M & & T_0\text{-}\mathbf{Mod}(\mathbf{Set})/M_s. \end{array}$$

where $\mathscr{D}^\rho/M$ maps any $\partial \colon R \to M$ to $\mathscr{D}^\rho(\partial) \colon \mathscr{D}^\rho(R) \to \mathscr{D}^\rho(M)$, and Δ and $\sum$ respectively denote pullback and postcomposition functors.

Now, Σ_ρ is a composite of four functors, three of which are left adjoints (because we restrict to finitary), hence readily finitary. It remains to show that the fourth factor, $\mathscr{D}^\rho/M$, is finitary. Because the domain functors $T_0\text{-}\mathbf{Mod}(\mathbf{Set}^{\mathbf{S}})/M \to T_0\text{-}\mathbf{Mod}(\mathbf{Set}^{\mathbf{S}})$ and $T_0\text{-}\mathbf{Mod}(\mathbf{Set})/\mathscr{D}^\rho(M) \to T_0\text{-}\mathbf{Mod}(\mathbf{Set})$ create colimits, this reduces to $\mathscr{D}^\rho$ being finitary. But finitary functors are closed under finite products, so, because colimits are pointwise in presheaf categories, this in turn reduces to each $(-)^{(p)}$ being finitary, which follows from their being left adjoints. (They may be viewed as precomposition with an endofunctor of $\mathbf{Kl}(T_0)$, hence admit a right adjoint given by right Kan extension.)

705 # Contents