Modules over monads and operational semantics

André Hirschowitz, Tom Hirschowitz, Ambroise Lafont

Modules over Monads and Operational Semantics

André Hirschowitz
Université Côte d'Azur, CNRS, Nice, France
https://math.unice.fr/~ah

Tom Hirschowitz
Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, 73000 Chambéry, France
https://www.lama.univ-savoie.fr/pagesmembres/hirschowitz

Ambroise Lafont
University of New South Wales, Sydney, Australia
https://amblafont.github.io

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 suitable notion of signature for transition monads.

2012 ACM Subject Classification Theory of computation → Semantics and reasoning; Theory of computation → Categorical semantics; Theory of computation → Operational semantics

Keywords and phrases Operational semantics, Category theory

Digital Object Identifier 10.4230/LIPIcs.FSCD.2020.12

Acknowledgements We thank the anonymous referees for their constructive comments.

1 Introduction

The search for a mathematical notion of programming language goes back at least to Turi and Plotkin [25], who coined the name "Mathematical Operational Semantics", and explained how known classes of well-behaved rules for structural operational semantics, such as GSOS [7], 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 [14, 13, 24], 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 [18, 2] (see also [1]), 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 transitions 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.
- 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 6. 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 $src : R \to S_1T$ and $tgt : R \to S_2T$. Such a triple (R, src, tgt) is thus an object of the slice category of T-modules over $S_1T \times S_2T$.

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 transition 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. Then, we wrap together 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**[1] functor $U : \mathbf{E} \to \mathbf{C}$, 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. Abstracting over this generating procedure, we introduce **registers** 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 $\mathbf{U}_S : S\text{-alg} \to \mathbf{C}$. Just as for semantic signatures, omitting $[\![-]\!]_R$ for readability, we think of a signature S as specifying the object $S^* = \mathbf{U}_S(S^{\circledast})$.

We may now state our achievement properly: we construct a register for transition monads, containing signatures specifying the desired examples. Towards this goal, we start in §4 by designing registers for monads and functors, relying on Ahrens et al. [2] and Fiore and Hur [11]. 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 of 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 §5.2 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 §5.3 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 fibres of a record category. Applying this

[1] Here "algebra" and "forgetful" have no technical meaning and are chosen by analogy.

to the previously constructed registers for our base product of three categories and our fibre slice categories of modules, we give in Definition 63 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, as we demonstrate in the appendix.

Related work

Beyond the already evoked related work [3, 25, 11], there is a solid body of work on categorical approaches to rewriting with variable binding, which only covers transition relations that are stable under arbitrary contexts, e.g., Hamana [16], T. Hirschowitz [19], and Ahrens [1]. Regarding signatures, Fiore [12], Altenkirch et al. [5], and Garner [15] use notions of signatures for languages with dependent types, which may provide an alternative approach to the specification of operational semantics systems. Finally, let us mention that a preliminary account of the present work appears in the third author's PhD thesis [20, Chapter 6].

Notations

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

The category of finitary monads on $\mathbf{C}$ is denoted by $\mathbf{Mnd}_f(\mathbf{C})$, or sometimes just $\mathbf{Mnd}_f$ when $\mathbf{C}$ is clear from context. Given a monad T on $\mathbf{C}$, the category of $\mathbf{D}$-valued (finitary) T-modules is denoted by $T\text{-}\mathbf{Mod}_f(\mathbf{D})$, where we recall [18] that such a T-module consists of a finitary functor $M : \mathbf{C} \to \mathbf{D}$ equipped with a right T-action $M \circ T \to M$ satisfying some coherence conditions.

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 $M^{(p_1,\dots,p_n)}(X) = M(X + \mathbf{y}_{p_1} + \cdots + \mathbf{y}_{p_n})$, where $\mathbf{y} : \mathbb{P} \to \mathbf{Set}^{\mathbb{P}}$ is the embedding defined by $\mathbf{y}_p(q) = 1$ if $p = q$ and $\emptyset$ otherwise. If $\mathbb{P}$ is a singleton, we abbreviate this to $M^{(n)}$.

2 Transition monads

2.1 Definition of transition monads

In this section, we introduce the main new mathematical notion of the paper which was already motivated by the case of the call-by-value, simply-typed, big-step λ-calculus in §1: transition monads. 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 transitions relate a source term to a target term. In a general transition monad we still have variables and transitions, but placetakers for variables and endpoints of transitions can be of a different nature, which we phrase as follows: variables are placeholders for **placetakers**, while transitions 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. For example, for simply-typed λ-calculus, $\mathbb{P} = \mathbb{S}$ is the set of simple types.

Placetakers form a **monad** on the category $\mathbf{Set}^{\mathbb{P}}$.

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 R, 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 $T_R(V)$ 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 the same as placetakers, in a general transition monad, they may differ, and more precisely both state objects are derived from the placetaker object by applying the **state functors** $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$.

The transitions. In standard λ-calculi, there is a (typed!) set of transitions, 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 transition object $Trans(V)$ equipped with two morphisms $src_V, trg_V : Trans(V) \to LC(V)$. Note that we consider "proof-relevant" transitions here, in the sense that two different transitions may have the same source and target. (Appendix G discusses how proof irrelevance can be recovered.)

In a general transition monad R, we still have a transition object $Trans_R(V)$, which now lives in $\mathbf{Set}^{\mathbb{S}}$, together with state objects $S_1(T_R(V))$ and $S_2(T_R(V))$, so that src_V and trg_V form a span $S_1(T_R(V)) \leftarrow Trans_R(V) \to S_2(T_R(V))$.

The S-graph of transitions. Now we rephrase the previous status of transitions in terms of a graph-like notion which we call S-graph: here $S := (S_1, S_2)$ is the pair of state functors. In the untyped λ-calculus, $Trans(V)$ and the maps src_V and trg_V 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.

▶ **Definition 1.** *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 $\partial : E \to S_1(V) \times S_2(V)$.*

*An **S-graph** consists of an object $V \in \mathbf{Set}^{\mathbb{P}}$ and an S-graph over V.*

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

The category of S-graphs. A reduction monad (in particular the untyped λ-calculus) is just a monad relative to the "discrete graph" functor from sets to graphs [3]. 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 $f : V_G \to V_{G'}$ together with a morphism for edges $g : E_G \to E_{G'}$ 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 2.** *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. First of all, let us recall [4] that, given any functor $J : \mathbf{C} \to \mathbf{D}$, a **monad relative to** J, or J**-relative monad**, consists of

- an object mapping $T : \mathbf{ob}(\mathbf{C}) \to \mathbf{ob}(\mathbf{D})$, together with
- morphisms $\eta_X : J(X) \to T(X)$, and
- for each morphism $f : J(X) \to T(Y)$, an **extension** $f^\star : T(X) \to T(Y)$,

satisfying coherence conditions. Any J-relative monad T has an underlying functor $\mathbf{C} \to \mathbf{D}$, and is said **finitary** when this functor is.

We will consider monads relative to functors of the following form.

▶ **Definition 3.** *For any functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, letting $S = (S_1, S_2)$, the **discrete S-graph** functor $J_S : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$ maps any $V \in \mathbf{Set}^{\mathbb{P}}$ to the S-graph on V with no edges.*

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

▶ **Definition 4.** *A **monadic transition monad** over $(\mathbb{P}, \mathbb{S})$ consists of*

- *two finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, and*
- *a finitary J_S-relative monad, where $S = (S_1, S_2)$.*

Given any J_S-relative monad T, we think of $T(X)$ as having terms with free variables in X as vertices, with all transitions between them as edges. A morphism $\sigma : J_S(X) \to T(Y)$ amounts to a mapping from X to terms in $T(Y)$, i.e., a substitution, and its extension $T(X) \to T(Y)$ models the action of σ both on terms and on transitions.

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 5.** *A **transition monad** over $(\mathbb{P}, \mathbb{S})$ consists of*

- *two finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$,*
- *a finitary monad T on $\mathbf{Set}^{\mathbb{P}}$, called the **placetaker** monad,*
- *a finitary T-module $R : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, called the **transition** module,*
- *a **source** T-module morphism $src : R \to S_1 T$,*
- *a **target** T-module morphism $tgt : R \to S_2 T$.*

This is the definition that we use in the rest of the paper.

▶ **Proposition 6.** *Modular and monadic transition monads are in one-to-one correspondence.*

Proof. See Appendix F. ◀

2.2 Examples of transition monads

In this section, we introduce informally a few example transition monads, which will be more rigorously defined in the appendix.

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

The $\overline{\lambda}\mu$-calculus [17] is an example with two placetaker types. Its grammar is given by

$$\begin{array}{ccc} \text{Processes} & \text{Programs} & \text{Stacks} \\ c ::= \langle e|\pi\rangle & e ::= x \mid \mu\alpha.c \mid \lambda x.e & \pi ::= \alpha \mid e \cdot \pi, \end{array}$$

where x and α 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 transition rules: $\langle \mu\alpha.c|\pi\rangle \to c[\alpha \mapsto \pi]$ $\langle \lambda x.e|e' \cdot \pi\rangle \to \langle e[x \mapsto e']|\pi\rangle$.

Let us show how this calculus gives rise to a transition monad. First, 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: the first one gives the available free program variables, and the second one the available free stack variables. The syntax may be viewed as a monad $T : \mathbf{Set}^2 \to \mathbf{Set}^2$: 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, up to bound variable renaming. As usual, monad multiplication is given by capture-avoiding 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 $S_i(T(X)) = T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}$ for $i = 1, 2$ 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 map $\langle src_X, tgt_X\rangle : Trans(X) \to (T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}})^2$. 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 standard presentation of π-calculus [23]. 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 $a(b).P$ and in $\nu b.P$. Processes are identified when related by the smallest context-closed equivalence relation $\equiv$ satisfying

$$0|P \equiv P \qquad P|Q \equiv Q|P \qquad P|(Q|R) \equiv (P|Q)|R \qquad (\nu a.P)|Q \equiv \nu a.(P|Q),$$

where in the last equation a should not occur free in Q. Transition is then given by the rules

$$\frac{}{\overline{a}\langle b\rangle.P|a(c).Q \longrightarrow P|(Q[c \mapsto b])} \qquad \frac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \qquad \frac{P \longrightarrow Q}{\nu a.P \longrightarrow \nu a.Q}.$$

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 $T : \mathbf{Set}^2 \to \mathbf{Set}^2$: given a variable object $X = (X_{\mathbf{c}}, X_{\mathbf{p}}) \in \mathbf{Set}^2$, the placetaker object $T(X) = (X_{\mathbf{c}}, T(X)_{\mathbf{p}}) \in \mathbf{Set}^2$ consists of the sets of channels and processes with free variables in X (modulo $\equiv$). Note that $T(X)_{\mathbf{c}} = X_{\mathbf{c}}$ as there is no operation on channels. Transitions 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 [7]. 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 $\dfrac{x_i \xrightarrow{a_{i,j}} y_{i,j}}{op(x_1,\dots,x_n) \xrightarrow{c} e}$, 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.

Each family of operations and rules 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 gives the term monad T. 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 set over $X \times (\mathbb{A} \times X)$ as desired.

2.2.4 Differential λ-calculus

The differential λ-calculus [9] provides a further example with $S_1 \neq S_2$. Its syntax may [26, §6] be defined by

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

denotes a (possibly empty) multiset, i.e., the ordering is irrelevant. Terms induce a monad T on **Set**, which we take as the placetaker monad (hence $\mathbb{P} = 1$).

Transitions relate terms to multiterms, hence $\mathbb{S} = 1$, S_1 is the identity, and $S_2 = !$ is the functor mapping any set X to the set of (finite) multisets over X.

The definition of transition is based on two intermediate notions:

1. **Unary multiterm substitution** $e[x \mapsto U]$ of a multiterm U for a variable x in a term e, which returns a multiterm (not to be confused with unary monadic substitution, which handles the particular case where U is just a singleton).
2. **Partial derivative** $\frac{\partial e}{\partial x} \cdot U$ 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 (see [26]) and induce T-module morphisms $T^{(1)} \times\, ! \circ T \to\, ! \circ T$.

Unary multiterm substitution and partial derivation are used to define the transition relation as the smallest context-closed relation satisfying the rules below.

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

The second rule relies on the abbreviation $\lambda x.\langle e_1, \dots, e_n \rangle := \langle \lambda x.e_1, \dots, \lambda x.e_n \rangle$.

One can show that transitions are 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 the simply-typed, call-by-value, big-step λ-calculus into a transition monad. Most often, big-step semantics describes evaluation of closed terms. Our approach requires to treat open terms as well, so we consider a variant describing the evaluation of open terms [21]. In this setting, the main 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 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 type τ the set $T(X)_\tau$ of values of type τ taking free (typed) variables in X.

In big-step semantics, transition relates terms to values. Hence, we are seeking state functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{P}}$ 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 subset of values therein. 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, transitions are stable under value substitution, so we obtain a transition monad.

2.3 Categories of transition monads

In the next sections, we show how to generate transition monads such as the examples of the previous section from basic data. For this, we follow the recipe of initial semantics; this requires as input a category of "models" equipped with a "forgetful" functor to the category of transition monads, and it outputs the image of the initial model by this functor (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 7.** *For any sets $\mathbb{P}$ and $\mathbb{S}$, finitary monad T over $\mathbf{Set}^{\mathbb{P}}$, and finitary functors $S_1, S_2 : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, let $\mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ denote the slice category T-$\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/S_1T \times S_2T$.*

This gives a first family of categories of transition monads, that we will integrate through a simple construction[2]:

▶ **Definition 8.** *A **record category** is a category of the form $\sum_{B \in \mathbf{ob}(\mathbf{B})} \mathbf{P}_B$ where B ranges over the objects of a **base category** $\mathbf{B}$, and each $\mathbf{P}_B$, called the **fibre over** B, is a category. In other words, it is given by a (base) category $\mathbf{B}$ equipped with a map $\mathbf{P} : \mathbf{ob}(\mathbf{B}) \to \mathbf{CAT}$.*

The relevant example for the present work is the following.

▶ **Definition 9.** *Given two sets $\mathbb{P}$ and $\mathbb{S}$, let $\mathbf{TMnd}_{\mathbb{P},\mathbb{S}}$ denote the following record category of transition monads with $\mathbb{P}$ and $\mathbb{S}$ as sets of types for placetakers and states:*

- *its base category is the product $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}}) \times [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f^2$ of the category of monads on $\mathbf{Set}^{\mathbb{P}}$ with two copies of the functor category $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$;*
- *the fibre over a triple (T, S_1, S_2) is the category $\mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ of Definition 7.*

3 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 the category of models associated to a suitable signature.

We start in §3.1 by introducing a general notion of semantic signature over a category. In §3.2, we define registers: a register is just a family of semantic signatures. Our main goal (achieved in Definition 63) is to propose a register for transition monads.

[2] There is a more comprehensive construction, obtained by observing that the assignment $(T, S_1, S_2) \mapsto \mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ forms a pseudofunctor and applying the so-called Grothendieck construction. Signatures for the earlier definitions of transition monads presented in [3] and [20, Chapter 6] fully acknowledge this fact, but here we choose to ignore it in order to make the development simpler.

3.1 Semantic signatures

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

▶ **Definition 10.** *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* $\mathbf{U}_s : S\text{-alg} \to \mathbf{C}$.

▶ Remark 11. The term "forgetful functor" is merely the name of the corresponding component of a semantic signature; it does not impose any further constraint on it.

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

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

▶ **Example 14.** Any object c of any given category $\mathbf{C}$ is 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 $F : \mathbf{Set} \to \mathbf{Set}$ with $F(X) = X + 1$. We define a semantic signature over $\mathbf{Set}$ 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 $\mathbb{N}$.

▶ **Definition 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 19, Section 3.2).*

▶ **Proposition 17.** *The assignment* $\mathbf{C} \mapsto UR_{\mathbf{C}}$ *extends to a functor* $\mathbf{CAT} \to \mathbf{SET}$*. The action of a functor* $F : \mathbf{C} \to \mathbf{D}$*, denoted by* UR_F*, is given by postcomposition.*

3.2 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 yielding a notion of model together with the existence of an initial one.

▶ **Definition 18.** *A* ***register*** R *for a given category* $\mathbf{C}$ *consists of*
- *a class* $\mathbf{Sig}_R$ *(of* ***signatures****), and*
- *a* ***semantics*** *map* $[\![-]\!]_R : \mathbf{Sig}_R \to UR_{\mathbf{C}}$.

We can now motivate the notation UR_C above:

▶ **Definition 19.** *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 20. When convenient, we introduce a register as $u : S \to UR_{\mathbf{C}}$ to be understood as the register R with $\mathbf{Sig}_R := S$ and $[\![-]\!]_R := u$. Moreover, we sometimes implicitly identify a signature s in a register 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 [22, p62].

▶ **Definition 21.** *For a given cocomplete category* $\mathbf{C}$*, the* universal finitary endofunctorial register $UFE_{\mathbf{C}}$ *is defined as the map* $[\mathbf{C},\mathbf{C}]_f \to UR_{\mathbf{C}}$ *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 of registers. Recalling Proposition 17, we have:

▶ **Definition 22.** *For any register* R *for* $\mathbf{C}$ *and functor* $F : \mathbf{C} \to \mathbf{D}$*, postcomposition with* UR_F *induces a register* $F_!(R) := (\mathbf{Sig}_R, UR_F \circ [\![-]\!]_R)$ *for* $\mathbf{D}$*.*

▶ **Definition 23.** *For any register* R *for* $\mathbf{C}$ *and map* $f : \mathbf{S} \to \mathbf{Sig}_R$*, precomposition with* f *induces a register* $f^*(R)$ *for* $\mathbf{C}$ *whose signatures are elements of* $\mathbf{S}$*. We say that* $f^*(R)$ *is a* ***subregister*** *of* R*.*

Here is an important application.

▶ **Definition 24.** *We call* ***endofunctorial*** *all registers of the form* $f^*(UFE_{\mathbf{C}})$*, for some map* $f : \mathbf{S} \to \mathbf{Sig}_{UFE_{\mathbf{C}}}$*.*

A useful fact is that endofunctorial registers are closed under the family construction:

▶ **Definition 25.** *For any endofunctorial register* R*, we denote by* R^* *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.*

4 Basic registers

In this section, we construct registers for monads and functors. Both of our initiality proofs follow from Fiore and Hur's theory of **equational systems** [11].

4.1 A register for monads

In this section, we fix a set $\mathbb{P}$ and construct a register $\mathbf{MndReg}(\mathbb{P})$ for monads on $\mathbf{Set}^{\mathbb{P}}$, generalising [2] to the simply-typed setting (see also Fiore and Hur [10]).

Let us first construct a naive register $\mathbf{MndReg}^0(\mathbb{P})$ which only allows us to specify operations. We will then deal with equations.

4.1.1 A naive register for specifying operations

We first describe signatures for $\mathbf{MndReg}^0(\mathbb{P})$. The basic idea for specifying operations is that the arity of an operation is a pair of ($\mathbf{Set}$-valued) **parametric** modules, in the sense of modules that are definable for any monad on $\mathbf{Set}^{\mathbb{P}}$.

▶ **Definition 26.** *Given a category* $\mathbf{D}$*, Let* $\mathbf{Mod}(\mathbf{D})$ *denote the category*

- *whose objects are pairs* (T, M) *of a finitary monad* T *on* $\mathbf{Set}^{\mathbb{P}}$ *and a finitary* T*-module* $M : \mathbf{Set}^{\mathbb{P}} \to \mathbf{D}$*,*
- *and whose morphisms* $(T, M) \to (U, N)$ *are pairs* (α, β) *of a monad morphism* $\alpha : T \to U$ *and a natural transformation* $\beta : M \to N$ *commuting with action.*

The first projection yields a forgetful functor $\mathbf{p} : \mathbf{Mod}(\mathbf{D}) \to \mathbf{Mnd}_f$*.*

▶ **Definition 27.** *A (*$\mathbf{D}$*-valued)* ***parametric*** *module is a section of* $\mathbf{p}$*, i.e., a functor* $s : \mathbf{Mnd}_f \to \mathbf{Mod}(\mathbf{D})$ *such that* $\mathbf{p} \circ s = id_{\mathbf{Mnd}_f}$*.*

▶ **Terminology 28.** *In the following, parametric modules are implicitly* **Set***-valued by default.*

▶ **Example 29.** Let us start by a few basic constructions of parametric modules:

- we denote by Θ the $\mathbf{Set}^{\mathbb{P}}$-valued parametric module mapping a monad T to itself, as a module over itself;
- for any $p_1, \dots, p_n \in \mathbb{P}$ and $\mathbf{D}$-valued parametric module M, let $M^{(p_1,\dots,p_n)}$ associate to each monad T the T-module $M(T)^{(p_1,\dots,p_n)}$ as in the notations of §1, i.e., $M^{(p_1,\dots,p_n)}(T)(X) = M(T)(X + \mathbf{y}_{p_1} + \dots + \mathbf{y}_{p_n})$; when $\mathbb{P} = 1$, we merely count the p_i's and write $M^{(n)}$;
- for any finitary functor $F : \mathbf{D} \to \mathbf{E}$ and $\mathbf{D}$-valued parametric module M, the $\mathbf{E}$-parametric module $F \circ M$ maps any monad T to the T-module $F \circ M(T)$; as particular cases:
  - the terminal $\mathbf{Set}$-valued parametric module $1 = 1 \circ \Theta$ maps any monad T to the constant T-module 1;
  - for any $p \in \mathbb{P}$ and $\mathbf{Set}^{\mathbb{P}}$-valued parametric module M, we denote by M_p the $\mathbf{Set}$-valued parametric module mapping any monad T to the T-module $X \mapsto M(X)_p$ (see § 2.2.1);
  - given a finite family $(M_i)_{i\in I}$ of $\mathbf{Set}$-valued parametric modules, I, let $\prod_i M_i$ associate to any monad T the T-module $\prod_i M_i(T)$.

▶ **Example 30.** An operation will be specified by two parametric modules, one for the source and one for the target. Let us give the parametric modules for a few operations from our examples.

Language	Operation	Source	Target
Pure $\overline{\lambda}\mu$	Push	$\Theta_{\mathbf{p}} \times \Theta_{\mathbf{s}}$	$\Theta_{\mathbf{s}}$
Pure $\overline{\lambda}\mu$	Abstraction	$\Theta^{(1)}_{\mathbf{p}}$	$\Theta_{\mathbf{p}}$
π-calculus	Input $a(b).P$	$\Theta_{\mathbf{c}} \times \Theta^{(\mathbf{c})}_{\mathbf{p}}$	$\Theta_{\mathbf{p}}$

Morally, a signature (without equations) should be a family of pairs of parametric modules. However, in order to ensure existence of an initial model, we restrict this as follows.

▶ **Definition 31.** *A signature of* $\mathbf{MndReg}^0(\mathbb{P})$ *is a family of pairs* (d, c) *of parametric modules, in which*

- *c has the shape Θ_p for some $p \in \mathbb{P}$, and*
- *d is* ***elementary****, in the sense that it is a finite product of parametric modules of the shape $(F \circ \Theta)^{(p_1,\dots,p_n)}$ for some $p_1, \dots, p_n \in \mathbb{P}$ and finitary functor $F : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}$.*

▶ **Example 32.** Typically, an elementary parametric module is a finite product of parametric modules of the shape $\Theta_p^{(p_1,\dots,p_n)}$, for some $p, p_1, \dots, p_n \in \mathbb{P}$.

▶ **Definition 33.** *The category of models associated to a signature $(d_i, c_i)_{i\in I}$ is defined by:*

- *A model is a monad T equipped with module morphisms $d_i(T) \to c_i(T)$ for all $i \in I$.*
- *A model morphism is a monad morphism commuting with these morphisms.*

▶ **Lemma 34.** *Any such category of models admits an initial object.*

With the obvious forgetful functor to the category of monads, this defines the semantic signature associated to a signature of $\mathbf{MndReg}^0(\mathbb{P})$, as a register for monads on $\mathbf{Set}^{\mathbb{P}}$.

4.1.2 A register for specifying operations and equations

Let us now define our register $\mathbf{MndReg}(\mathbb{P})$, following Ahrens et al.'s approach to specifying equations [2]. A signature of $\mathbf{MndReg}(\mathbb{P})$ will consist of a signature of $\mathbf{MndReg}^0(\mathbb{P})$, plus a family of "equations". An equation is essentially a pair of "metaterms", which may have "metavariables". The idea is that metavariables are given by a parametric module.

▶ **Example 35.** Consider associativity of parallel composition in π-calculus, $P|(Q|R) \equiv (P|Q)|R$: the metavariables are P, Q, and R. The corresponding parametric module is $\Theta^3_{\mathbf{p}}$.

Intuitively, a metaterm will be a parametric module morphism from metavariables to some Θ_p. However, it should potentially rely on constructions from the considered signature Σ of $\mathbf{MndReg}^0(\mathbb{P})$, as in Example 35. We thus consider a modified notion of parametric module morphism, which is parametric in models of Σ instead of mere monads.

▶ **Definition 36.** *Given any signature Σ for $\mathbf{MndReg}^0(\mathbb{P})$,*

- *a Σ-**module morphism** $M \to N$ between parametric modules M and N is a natural family of morphisms $(\alpha_T : M(T) \longrightarrow N(T))_{T \in \Sigma\text{-alg}}$, such that α_T is a T-module morphism, for each Σ-model T;*
- *a Σ-**equation** consists of an elementary parametric module V, called the **metavariable module**, and two parallel Σ-module morphisms $V \longrightarrow \theta_p$, for some $p \in \mathbb{P}$, called the **metaterms (of type p)**.*

▶ **Definition 37.** *A signature of $\mathbf{MndReg}(\mathbb{P})$ is a pair of a signature Σ of $\mathbf{MndReg}^0(\mathbb{P})$ and a family of Σ-equations.*

▶ **Definition 38.** *The category of models associated to a signature (Σ, E) is defined as follows.*

- *A model is a model T of Σ such that for all equations $(L, R) \in E$, $L(T) = R(T)$.*
- *A morphism of models of (Σ, E) is a morphism of models of Σ.*

The following generalises [2, Theorem 32]:

▶ **Lemma 39.** *Any such category of models admits an initial object.*

With the obvious forgetful functor to the category of monads, this defines the semantic signature associated to a signature of $\mathbf{MndReg}(\mathbb{P})$, as a register for monads on $\mathbf{Set}^{\mathbb{P}}$.

▶ **Example 40.** Let us revisit Example 35: the relevant signature Σ has in particular an operation $par : \Theta^2_{\mathbf{p}} \to \Theta_{\mathbf{p}}$ for parallel composition, which gives our two metaterms

$$\Theta^3_{\mathbf{p}} \xrightarrow{par \times \Theta_{\mathbf{p}}} \Theta^2_{\mathbf{p}} \xrightarrow{par} \Theta_{\mathbf{p}} \qquad \text{and} \qquad \Theta^3_{\mathbf{p}} \xrightarrow{\Theta_{\mathbf{p}} \times par} \Theta^2_{\mathbf{p}} \xrightarrow{par} \Theta_{\mathbf{p}}.$$

▶ Notation 41 (Format for equations). We have already started to write pairs (d, c) of parametric modules as $d \to c$. Given any signature Σ for $\mathbf{MndReg}^0(\mathbb{P})$, we write any Σ-equation $\begin{array}{ccc} V & \to & \Theta^2_p \\ x & \mapsto & (L, R) \end{array}$ as $x : V \vdash L \equiv R : \Theta_p$, or just $L \equiv R$ when the rest may be inferred.

▶ **Example 42.** We write associativity from Example 40 as just $par(P, par(Q, R)) \equiv par(par(P, Q), R)$. In this case, the argument x is the triple (P, Q, R).

4.2 A register for state functors

In this section, we sketch a register $\mathbf{FunReg}(\mathbb{P}, \mathbb{S})$, which is an adaptation of $\mathbf{MndReg}(\mathbb{P})$ to the case of state functors. Parametric modules are replaced with parametric premodules:

▶ **Definition 43.** *A **parametric premodule** is a functor $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}]_f$.*

We introduce the following notations:

▶ Notation 44. We denote by Θ the identity endofunctor on $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f$, and by $\Gamma : [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f \to [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{P}}]_f$ the constant functor mapping anything to the identity endofunctor.

The constructions $\prod_i M_i$ and $M^{(p_1,\dots,p_n)}$ carry over essentially verbatim.

▶ **Example 45.** We have seen in §2.2.5 that the source state functor S_1 for call-by-value, simply-typed λ-calculus is built with application binary trees. Intuitively, it has two (type-indexed families of) operations: the first one injects values, thus maps X_t, into $S_1(X)_t$, and the second one forms application binary trees, with components $X_t \to S_1(X)_t$ and $S_1(X)_{t\to t'} \times S_1(X)_t \to S_1(X)_{t'}$.
This yields a specification with two families of operations $\Gamma_t \to \Theta_t$ and $\Theta_{t\to t'} \times \Theta_t \to \Theta_{t'}$.

Operations, equations, and models are defined exactly as for monads, and a signature in $\mathbf{FunReg}(\mathbb{P},\mathbb{S})$ again consists of families of operations and equations[3]. The only difference lies in the notion of elementary parametric premodule, which becomes the following:

▶ **Definition 46.** *A parametric premodule is* ***elementary*** *iff it is a finite product of parametric premodules of the shape* $(F \circ \langle \Gamma, \Theta\rangle)^{(p_1,\dots,p_n)}$ *for some* $p_1,\dots,p_n \in \mathbb{P}$ *and finitary functor* $F : \mathbf{Set}^{\mathbb{P}} \times \mathbf{Set}^{\mathbb{S}} \to \mathbf{Set}$.

▶ **Example 47.** Typically, an elementary parametric premodule is a product of parametric premodules of the shape $\Gamma_p^{(p_1,\dots,p_n)}$ or $\Theta_\sigma^{(p_1,\dots,p_n)}$, for some $p, p_1,\dots,p_n \in \mathbb{P}$ and $\sigma \in \mathbb{S}$.

▶ Remark 48. Any finitary functor F admits a trivial signature consisting of the family $((F_\sigma \circ \Gamma) \to \Theta_\sigma)_{\sigma\in\mathbb{S}}$ of operations. Here are a few examples from §2.2:

Language	State functor	Specification
$\overline{\lambda}\mu$	$S_1(X) = S_2(X) = X_{\mathbf{p}} \times X_{\mathbf{s}}$	$\langle -\vert -\rangle : \Gamma_{\mathbf{p}} \times \Gamma_{\mathbf{s}} \to \Theta$
π	$S_1(X) = S_2(X) = X_{\mathbf{p}}$	$\Gamma_{\mathbf{p}} \to \Theta$
Call-by-value, simply-typed λ	$S_2(X) = X$	$\eta_t : \Gamma_t \to \Theta_t$ (for all t)
Positive GSOS specifications	$S_1(X) = X$ $S_2(X) = \mathbb{A} \times X$	$\Gamma \to \Theta$ $\mathbb{A} \times \Gamma \to \Theta$

▶ Notation 49. We adopt Notation 41 for state functors. E.g., Example 42 applies verbatim for associativity of multiset union in the target state functor for differential λ-calculus.

5 Constructions of registers

In this section, we provide constructions of new registers out of existing ones.

5.1 Product registers

Let us start by considering products. We first describe the product of semantic signatures, and then, based on that, we define product registers.

Given semantic signatures for some family of categories, we want to construct a semantic signature for the product category. The application we have in mind is the product category $\mathbf{Mnd}_f(\mathbf{Set}^{\mathbb{P}}) \times [\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]_f^2$ (Definition 9), which is the base category of our record category of transition monads (see below Example 53).

▶ **Lemma 50.** *Given a set* I *and functors* $U_i : \mathbf{E}_i \to \mathbf{C}_i$ *for* $i \in I$*, if each* $\mathbf{E}_i$ *has an initial object, then so does the product* $\prod_i \mathbf{E}_i$.

[3] Existence of an initial object in the category of models relies on the theory of equational systems, as mentioned above.

▶ **Definition 51.** *Given a family* $\mathbf{C} := (\mathbf{C}_i)_{i\in I}$ *of categories, and a corresponding family of semantic signatures* $u_i : \mathbf{E}_i \to \mathbf{C}_i$*, the product* $\prod_i u_i : \prod_i \mathbf{E}_i \to \prod_i \mathbf{C}_i$ *is a semantic signature. This defines our (external) product of signatures* $\prod_{\mathbf{C}} : \prod_i UR_{\mathbf{C}_i} \to UR_{\prod_i \mathbf{C}_i}$.

Let us now define the product of a family of registers.

▶ **Definition 52.** *The* ***product*** *of a family* $(u_i : S_i \to UR_{\mathbf{C}_i})_{i\in I}$ *of registers is obtained by post-composing* $\prod_i u_i$ *with the product of semantic signatures:*

$$\prod_i S_i \xrightarrow{\prod_i u_i} \prod_i UR_{\mathbf{C}_i} \xrightarrow{\prod_{\mathbf{C}}} UR_{\prod_i \mathbf{C}_i}.$$

▶ **Example 53.** The product $\mathbf{MndReg}(\mathbb{P}) \times \mathbf{FunReg}(\mathbb{P},\mathbb{S})^2$ of the register $\mathbf{MndReg}(\mathbb{P})$ with two copies of the register $\mathbf{FunReg}(\mathbb{P},\mathbb{S})$.

5.2 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 $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M$. Later on, we will use the register $\mathbf{Rule}^*(T,M)$ (recalling Definition 25) with $M := S_1T \times S_2T$, i.e., for the category of transition monads over (T, S_1, S_2).

5.2.1 The naive register $\mathbf{Rule}^0$

For expository purposes, we start by defining a naive endofunctorial register, $\mathbf{Rule}^0(T,M)$. A signature of $\mathbf{Rule}^0(T,M)$ consists of

- a **metavariable** $\mathbf{Set}$-valued T-module V,
- a **conclusion** module morphism $t : V \to M_\tau$ for some **conclusion** state type $\tau \in \mathbb{S}$, and
- a list of **premise** module morphisms $s_i : V \to M_{\sigma_i}$, for some **premise** state types $\sigma_i \in \mathbb{S}$.

▶ **Example 54.** 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

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

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

$$\begin{array}{ccccc} T\text{-}\mathbf{Mod}_f(\mathbf{Set})/\prod_i M_{\sigma_i} & \xrightarrow{\Delta_{\langle s_i\rangle_i}} & T\text{-}\mathbf{Mod}_f(\mathbf{Set})/V & \xrightarrow{\sum_t} & T\text{-}\mathbf{Mod}_f(\mathbf{Set})/M_\tau \\ \prod_i (-)_{\sigma_i} \uparrow & & & & \downarrow \\ T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M & & & & T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M, \end{array} \tag{1}$$

of four functors, where

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

- 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 every X by $(R \cdot \mathbf{y}_\tau)(X)_\tau = R(X)$ and $(R \cdot \mathbf{y}_\tau)(X)_\sigma = \emptyset$ for $\sigma \neq \tau$.

▶ Remark 55. The embedding $(-) \cdot \mathbf{y}_\tau$ is left adjoint to evaluation at τ: $(-) \cdot \mathbf{y}_\tau \dashv (-)_\tau$. Thus Σ_S maps any $\partial : R \to M$ to the transpose of the right-hand composite q below.

$$\begin{array}{ccccc} \prod_i R_{\sigma_i} & \longleftarrow & P & & \\ {\scriptstyle \prod_i \partial_{\sigma_i}}\downarrow & & \downarrow & \searrow^{q} & \\ \prod_i M_{\sigma_i} & \xleftarrow[\langle s_i \rangle_i]{} & V & \xrightarrow[t]{} & M_\tau \end{array} \tag{2}$$

By Lemma 59 below, each Σ_S is finitary, which completes the definition of our register $\mathbf{Rule}^0(T, M)$ for $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M$.

▶ **Example 56.** Consider the endofunctor associated to the left application rule of Example 54. Because $\mathbb{S} = 1$, the functor $(-) \cdot \mathbf{y}_\tau$ is the identity functor, so the endofunctor maps any $\partial : R \to T^2$:

- 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 $\partial : R \to T^2$ which, to each such tuple (r, e, e', f) associates a transition over $(e\ f, e'\ f)$, as desired.

5.2.2 The register Rule

In this section, we define the endofunctorial register $\mathbf{Rule}(T, M)$, refining the naive register $\mathbf{Rule}^0(T, M)$ of the previous section. The motivation lies in rules whose premises have additional free variables.

▶ **Example 57.** Consider the ξ rule of pure λ-calculus: $\dfrac{e \to f}{\lambda x.e \to \lambda x.f}$.

The metavariables 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 $\mathbf{Rule}^0(T, M)$ to let rules have premises of this shape:

▶ **Definition 58.** *The endofunctorial register* $\mathbf{Rule}(T, M)$ *for* $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M$ *is defined by:*

- *signatures are just as in* $\mathbf{Rule}^0(T, M)$*, except that the premises now have the shape* $s : V \to M_\sigma^{(\vec{p})}$*, for* $\sigma \in \mathbb{S}$ *and* $\vec{p}$ *a list of elements of* $\mathbb{P}$*; and*
- *the induced endofunctor is defined exactly as for naive rules, replacing* $\prod_i R_i$ *with* $\prod_i R_i^{(\vec{p_i})}$.

This register is well defined thanks to the following lemma (proved in Appendix H):

▶ **Lemma 59.** *Given a signature* S*, the endofunctor* Σ_S *is finitary.*

Using Definition 25, we obtain a register $\mathbf{Rule}^*(T, M)$ for $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M$, whose signatures are families of signatures in $\mathbf{Rule}(T, M)$.

5.2.3 A format for signatures in Rule and Rule*

When $M = S_1T \times S_2T$, we adopt the following notational conventions for signatures in $\mathbf{Rule}(T, M)$:

- for each premise or conclusion $\begin{array}{ccc} V & \to & W \\ x & \mapsto & (L, R) \end{array}$ of a rule, we write $x : V \vdash L \rightsquigarrow R : W$,
- we organise the premises and conclusion as usual:

$$\frac{x : V \vdash L_1 \rightsquigarrow R_1 : W_1 \qquad \dots \qquad x : V \vdash L_n \rightsquigarrow R_n : W_n}{x : V \vdash L \rightsquigarrow R : W},$$

or just $\dfrac{L_1 \rightsquigarrow R_1 \qquad \dots \qquad L_n \rightsquigarrow R_n}{L \rightsquigarrow R}$ when the rest may be inferred from context.

▶ Remark 60. The module V is often a product and thus x is a tuple.

▶ Remark 61 ([3]). In practice, there are several choices for building the transition rule out of such a schematic presentation, depending on the order of metavariables. This order is irrelevant: all interpretations yield isomorphic semantics, in the obvious sense.

5.3 The record construction for registers

The 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 monads and functors previously defined. We may even assemble these signatures into a single signature Σ for the product register of Definition 52. Then, we may specify the desired transition monad as an object of the fibre $\mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$, using a signature $\mathbf{R}$ of the register $\mathbf{Rule}^*(T, M)$ from Section 5.2.2, with $M = S_1T \times S_2T$.

In this section, we show how to assemble Σ and $\mathbf{R}$ into a single signature of some compound register for the record category $\mathbf{TMnd}_{\mathbb{P},\mathbb{S}}$. Our construction can be performed in general for an arbitrary record category.

▶ **Definition 62.** *Consider any record category $K = \sum_{B \in \mathbf{ob}(\mathbf{B})} \mathbf{P}(B)$, with $\mathbf{P} : \mathbf{ob}(\mathbf{B}) \to \mathbf{CAT}$, together with*

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

*The **record register** $\sum(R_b, R_f)$ for the record category K is defined as follows.*

- *Signatures are pairs (S, F) with $S \in \mathbf{Sig}_{R_b}$ and $F \in \mathbf{Sig}_{R_f(B)}$.*
- *The semantic signature associated to any (S, F) is the composite F-alg $\xrightarrow{\mathbf{U}_F} \mathbf{P}_{S^*} \hookrightarrow K$.*

Our main example application is:

▶ **Definition 63.** *Let $\mathbf{TMndReg} := \sum(R_b, R_f)$, where*

- *R_b is the product register $\mathbf{MndReg}(\mathbb{P}) \times \mathbf{FunReg}(\mathbb{P}, \mathbb{S})^2$ of Example 53 for monads and state functors, and*
- *for all signatures S of R_b, the register $R_f(S)$ is defined as $\mathbf{Rule}^*(T, S_1T \times S_2T)$, where $(T, S_1, S_2) = S^*$.*

We provide signatures for all examples from §2.2 in the appendix.

6 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 to investigate more general forms of state modules. E.g., using an arbitrary module covers the subtle labelled transition system for π-calculus. We also plan to generalise the Grothendieck construction to signatures/registers along the line in [3]. In the longer term, we plan to refine our register in a way ensuring that the generated transition system satisfies important properties like congruence of observational equivalences, confluence, or type soundness. In this direction, a result on congruence of applicative bisimilarity for a simpler register has recently been obtained by Borthelle et al. [8]. Finally, quantitative (e.g., probabilistic [6]) operational semantics would be worth investigating in our setting.

References

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

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 Falk Bartels. GSOS for probabilistic transition systems: (extended abstract). *Electronic Notes in Theoretical Computer Science*, 65(1):29–53, 2002. CMCS '2002, Coalgebraic Methods in Computer Science. doi:10.1016/S1571-0661(04)80358-X.

7 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.

8 Peio Borthelle, Tom Hirschowitz, and Ambroise Lafont. A cellular Howe theorem. In *Proc. 35th ACM/IEEE Logic in Comp. Sci.* ACM, 2020. doi:10.1145/3373718.3394738.

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

10 M. P. Fiore and C.-K. Hur. Second-order equational logic. In *Proceedings of the 19th EACSL Annual Conference on Computer Science Logic (CSL 2010)*, 2010.

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

12 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.

13 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.

14 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.

15 Richard Garner. Combinatorial structure of type dependency. *Journal of Pure and Applied Algebra*, 219(6):1885 – 1914, 2015. doi:10.1016/j.jpaa.2014.07.015.

16 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.

17 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.

18 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.

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

20 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.

21 Luca Paolini and Simona Ronchi Della Rocca. Call-by-value solvability. *RAIRO - Theor. Inf. and Applic.*, 33(6):507–534, 1999. doi:10.1051/ita:1999130.

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

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

24 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.

25 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.

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

A Specifying the call-by-value, simply-typed, big-step λ-calculus

In the setting of §2.2.5, let $F : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$ be specified by two families of operations $app_{t,t'} : \Theta_{t\to t'} \times \Theta_t \to \Theta_{t'}$ and $val_t : \Gamma_t \to \Theta_t$. Our signature for call-by-value, simply-typed, big-step λ-calculus is presented in the following table

Monad and state functors	T	S_1	S_2
	$\lambda_{t,t'} : (F_{t'} \circ \Theta)^{(t)} \to \Theta_{t\to t'}$	F	Id
Rules	$\dfrac{}{val_t(v) \rightsquigarrow v}$		$\dfrac{e_1 \rightsquigarrow \lambda_{t,t'}(e_3) \qquad e_2 \rightsquigarrow w \qquad e_3[w] \rightsquigarrow v}{app_{t,t'}(e_1, e_2) \rightsquigarrow v}$

where

- $-[-] : (S_1T)^{(t)}_{t'} \times T_t \to (S_1T)_{t'}$ denotes the substitution morphism;
- $S_1 = F$ and $S_2 = \mathrm{Id}$ are specified by easy signatures as in Remark 48;
- the rules should be understood as families of rules indexed by suitable types.

In a bit more detail, the first rule is indexed by the type t of v. The second one is indexed by two types t and t'. There are five metavariables, e_1, e_2, e_3, v, and w. We thus take $V := (S_1T)_{t\to t'} \times (S_1T)_t \times (S_1T)^{(t)}_{t'} \times T_{t'} \times T_t$.

B Specifying the $\overline{\lambda}\mu$-calculus

For $\overline{\lambda}\mu$-calculus, the state functor has been specified in Remark 48. The monad is specified by operations

$$\mu : \Theta^{(\mathbf{s})}_{\mathbf{p}} \times \Theta^{(\mathbf{s})}_{\mathbf{s}} \to \Theta_{\mathbf{p}} \qquad \lambda : \Theta^{(\mathbf{p})}_{\mathbf{p}} \to \Theta_{\mathbf{p}} \qquad \cdot : \Theta_{\mathbf{p}} \times \Theta_{\mathbf{s}} \to \Theta_{\mathbf{s}},$$

with no equation.

The transition rules are almost as usual:

$$\frac{}{\langle \mu\langle e|\pi'\rangle|\pi\rangle \to \langle e[\pi], \pi'[\pi]\rangle} \qquad \frac{}{\langle \lambda(e)|e'\cdot\pi\rangle \to \langle e[e']|\pi\rangle}$$

The first rule has metavariable module $V := T_{\mathbf{p}}^{(\mathbf{s})} \times T_{\mathbf{s}}^{(\mathbf{s})} \times T_{\mathbf{s}}$, the argument being (e, π', π). The second rule has $V := T_{\mathbf{p}}^{(\mathbf{p})} \times T_{\mathbf{p}} \times T_{\mathbf{s}}$.

C Specifying the π-calculus

For π-calculus, the state functor has been specified in Remark 48. The placetaker monad T is specified by operations

$$0 : 1 \to \Theta_{\mathbf{p}} \quad |\, : \Theta_{\mathbf{p}} \times \Theta_{\mathbf{p}} \to \Theta_{\mathbf{p}} \quad \nu : \Theta_{\mathbf{p}}^{(\mathbf{c})} \to \Theta_{\mathbf{p}} \quad out : \Theta_{\mathbf{c}}^2 \times \Theta_{\mathbf{p}} \to \Theta_{\mathbf{p}} \quad in : \Theta_{\mathbf{c}} \times \Theta_{\mathbf{p}}^{(\mathbf{c})} \to \Theta_{\mathbf{p}}$$

with equations $0|P \equiv P \quad P|Q \equiv Q|P \quad P|(Q|R) \equiv (P|Q)|R \quad \nu(P)|Q \equiv \nu(P|\mathbf{w}_{\mathbf{c}}(Q))$, almost copied verbatim from §2.2.2, where $\mathbf{w}_{\mathbf{c}}(Q)$ denotes the action of $T(X) \to T(X + \mathbf{y}_{\mathbf{c}})$ on Q. Finally, the transition rules are

$$\frac{}{out(a,b,P)|in(a,Q) \longrightarrow P|(Q[b])} \qquad \frac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \qquad \frac{P \longrightarrow Q}{\nu(P) \longrightarrow \nu(Q)}\,.$$

In particular, the third rule has as metavariable module $V := (\Theta_{\mathbf{p}}^{(\mathbf{c})})^2$.

D The register $GSOS^+$

In this section, we define a register $GSOS^+$ for specifying positive GSOS systems [7]. This is a subregister of our record register **TMndReg**, for untyped ($\mathbb{P} = \mathbb{S} = 1$) transition monads. Let us recall that signatures in this register consist of pairs (B, F) where B is a signature in the product register of Example 53, and F is a signature in $\mathbf{Rule}^*(B)$.

In order to describe this subregister, we have to describe its class of signatures, and then assign to each such signature a pair (B, F) as above. Before performing this task we recall the standard format of a $GSOS^+$ rule:

$$\frac{\dots \quad V_i \xrightarrow{a_{i,j}} V_{i,j} \quad \dots}{op(V_1, \dots, V_n) \xrightarrow{c} e}.$$

A signature of the register $GSOS^+$ consist of

- three sets O (for operations), $\mathbb{A}$ (for labels), and R (for rules),
- for each element o of O, a number m_o (the arity),
- for each rule,
  - an operation $o \in O$ (for the source of the conclusion),
  - a label $c \in \mathbb{A}$ (the label of the conclusion),
  - for each $i \leq m_o$,
    - a number n_i (the number of premises for this argument),
    - for each $j \leq n_i$, an element a_{ij} of $\mathbb{A}$ (for the label of the premise),
    - a term e in the syntax generated by O, potentially depending on $m_o + \sum_i n_i$ variables.

We now describe the pair (B, F) associated to a signature as above:

- the signatures for both state functors have been given in Remark 48;

- the signature for the underlying monad is $\sum_{o \in O} \Theta^{m_o} \to \Theta$ (following §4.1).

These three signatures yield our base signature B. Finally, each Positive GSOS rule yields a rule $\dfrac{\dots \quad V_i \rightsquigarrow (a_{i,j}, V_{i,j}) \quad \dots}{op_o(V_1, \dots, V_{m_o}) \rightsquigarrow (c, e)}$ in our fibre signature F (in the register $\mathbf{Rule}^*(T, S_1T \times S_2T)$).

E Specifying the differential λ-calculus

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 §E.1, and a signature for the β and ∂-transition rules. Both are straightforwardly modelled by a signature over as explained in §5.2, but they first require us to construct some intermediate operations $-[x \mapsto -]$ and $\frac{\partial -}{\partial x} \cdot -$. We tackle this task in §E.2.

E.1 State functors and monad of differential λ-calculus

The first state functor is the identity functor $\mathrm{Id} : \mathbf{Set} \to \mathbf{Set}$, and thus is specified by the arity $\Gamma \to \Theta$. The second state functor is !, the multiset functor, and is specified by three arities $1 \to \Theta$ (for the empty multiset), $\Gamma \to \Theta$ (for the singleton multiset), and $\Theta \times \Theta \to \Theta$ (for the union operation), subject to commutativity, associativity, and unitality.

Next, the monad of differential λ-calculus is specified by the arities $\Theta^{(1)} \to \Theta$, $\Theta \times !\Theta \to \Theta$, and $\Theta \times \Theta \to \Theta$, modelling the operations $\lambda x.-$, $- \; -$, and $D - \cdot -$. No equation is required.

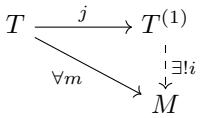
E.2 Intermediate constructions for differential λ-calculus

Specifying the transition 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 $T^{(1)} \times !T \to !T$, or equivalently $T^{(1)} \to (!T)^{!T}$. [4]

In [26, §6], the underlying maps are defined by induction. Here, we define them by using a special induction principle for building module morphisms out of $T^{(1)}$, that we now describe. Let us denote by $\Sigma^\uparrow$ the endofunctor on T-modules defined by the same formula than the parametric module specifying the differential λ-calculus monad T (see Section E.1): $\Sigma^\uparrow(M) = M^{(1)} + M \times !M + M \times M$. Note that T has a canonical $\Sigma^\uparrow$-algebra structure, $T^{(1)}$ a canonical $(\Sigma^\uparrow + 1)$-algebra structure, and the embedding $T \to T^{(1)}$ is a $\Sigma^\uparrow$-algebra morphism.

The following lemma induces a useful induction principle:

▶ **Lemma 64.** *Given any $(\Sigma^\uparrow + 1)$-algebra M and $\Sigma^\uparrow$-algebra morphism $m : T \to M$, there exists a unique $(\Sigma^\uparrow + 1)$-algebra morphism $i : T^{(1)} \to M$ making the following diagram commute.*



[4] The category of finitary **Set**-valued T-modules is equivalent to the category of presheaves on the full subcategory of the Kleisli category of T consisting of finite sets. As such, it has exponentials.

Proof. By [11], T is the initial algebra of $\Sigma^{\uparrow} + \mathrm{Id}$, as a (finitary) endofunctor on $[\mathbf{Set}, \mathbf{Set}]_f$. By inspecting the colimit of the relevant initial chains, it can be shown that $T^{(1)}$ is the initial algebra of $\Sigma^{\uparrow} + \mathrm{Id} + 1$. Given the data, M has $(\Sigma^{\uparrow} + \mathrm{Id} + 1)$-algebra structure: the natural transformation $\mathrm{Id} \to M$ is merely the composite $\mathrm{Id} \to T \to M$ with the unit of T.

By initiality, we have a unique $(\Sigma^{\uparrow} + \mathrm{Id} + 1)$-algebra morphism $i : T^{(1)} \to M$ as functors, such that $i \circ j = m$. At this stage, we can already deduce uniqueness of the desired morphism.

It remains to show that i is a T-module morphism. We omit the proof for lack of space. ◀

The lemma legitimates the following general recipe for constructing a T-module morphism $T^{(1)} \to M$:

1. provide M with a $(\Sigma^{\uparrow} + 1)$-algebra structure,
2. provide a T-module morphism $T \to M$, and
3. check that this morphism is a $\Sigma^{\uparrow}$-algebra morphism.

Indeed, by the above adjunction, we get a $(\Sigma^{\uparrow} + 1)$-algebra morphism $T^{(1)} \to M$, and thus in particular a T-module morphism $T^{(1)} \to M$.

We now apply this recipe to define unary multiterm substitution (the case of partial derivation is similar).

1. We first equip $(!T)^{!T}$ with $(\Sigma^{\uparrow} + 1)$-algebra structure. This structure should reflect the recursive equations in [26, Definition 6.3] defining unary multiterm substitution. In fact, the recursive equations will follow from the fact that the constructed morphism $T^{(1)} \to (!T)^{!T}$ is a $(\Sigma^{\uparrow} + 1)$-algebra morphism.

   By universal property of exponential and coproduct, a $(\Sigma^{\uparrow} + 1)$-algebra structure on $(!T)^{!T}$ decomposes into a 4-tuple of maps to $!T$, each one corresponding to a recursive equation. We define these maps, recalling the corresponding recursive equation, in the following table.

Inductive case	Recursive equation	T-module morphism
Abstraction	$(\lambda x.t)[x \mapsto U] = \lambda x.t[x \mapsto U]$	$((!T)^{!T})^{(1)} \times !T \to !T$ $(t, U) \mapsto \lambda.t(U)$
Application	$((s)\ V)[x \mapsto U] = (s[x \mapsto U])\ V[x \mapsto U]$	$(!T)^{!T} \times !(!T)^{!T} \times !T \to !T$ $(s, V, U) \mapsto (s(U))\ V(U)$
Differential application	$(Ds \cdot u)[x \mapsto U] = D(s[x \mapsto U]) \cdot u[x \mapsto U]$	$(!T)^{!T} \times (!T)^{!T} \times !T \to !T$ $(s, u, U) \mapsto D(s(U)) \cdot u(U)$
Substituted variable	$x[x \mapsto U] = U$	$!T \to !T$ $U \mapsto U$

   This covers four out of five recursive equations. The missing one is $y[x \mapsto U] = y$ when $x \neq y$; it corresponds to the morphism $T \to (!T)^{!T}$, which more generally deals with any term not depending on x.

   Let us explain some cases, beginning with the last one. The morphism $!T \to !T$ corresponds to the mapping $U \mapsto x[x \mapsto U]$, thus we choose the identity morphism.

   Next, e.g., the morphism for application is a composite

$$(!T)^{!T} \times !(!T)^{!T} \times !T \quad \to \quad !T \times !!T \quad \to \quad !(T \times !T) \quad \xrightarrow{!app} \quad !T,$$

   where

   - the first morphism duplicates $!T$ and evaluates both exponentials, and
   - the second follows from the well-known fact that ! is a commutative monad.

   The other cases are similar, and require to lift the other operations to the level of multiterms.

2. Now, following our recipe, we need to give a T-module morphism from T to $(!T)^{!T}$, or equivalently, a T-module morphism $m : T \times !T \to !T$. This morphism corresponds to the mapping $(t, U) \mapsto t[x \mapsto U]$, when t does not depend on x. Thus, we define $m_X(t, U) = t$. More formally, m is the composite $T \times !T \xrightarrow{\pi_1} T \xrightarrow{\eta^{!}T} !T$.
3. It remains to check that the induced morphism $T \to (!T)^{!T}$ is a $\Sigma^{\uparrow}$-algebra morphism, that is, that this morphism is compatible with each operation, which is routine.

F Proof of Proposition 6

In this section, we show that the modular and the monadic definitions of transition monads are equivalent. The proof consists merely in unfolding the definitions.

Consider the modular definition. We have:

- a finitary monad T on $\mathbf{Set}^{\mathbb{P}}$, that is:
  - an object mapping $T : \mathbf{ob}(\mathbf{Set}^{\mathbb{P}}) \to \mathbf{ob}(\mathbf{Set}^{\mathbb{P}})$, together with
  - morphisms $X \to T(X)$ for the variables, and
  - for each morphism $f : X \to T(Y)$, an extension $f^\star : T(X) \to T(Y)$, subject to the usual equations;
- a finitary T-module $R : \mathbf{Set}^{\mathbb{P}} \to \mathbf{Set}^{\mathbb{S}}$, called the **transition** module, that is:
  - an object mapping $R : \mathbf{ob}(\mathbf{Set}^{\mathbb{P}}) \to \mathbf{ob}(\mathbf{Set}^{\mathbb{S}})$, together with
  - for each morphism $f : X \to T(Y)$, an extension $f^\star : R(X) \to R(Y)$ subject to the usual equations;
- two T-module morphisms $s_i : R \to S_i T$, that is, families of morphisms $R(X) \to S_i(T(X))$ commuting with T-substitution.

The monadic definition was already detailed after Definition 4. It is straightforward to check that the assignement $X \mapsto (src_X, tgt_X : R(X) \to S_i(T(X)))$ defines a monadic transition monad. Conversely, given a monadic transition monad mapping X to some $(src_X, tgt_X : R(X) \to S_i(T(X)))$, the assignement $X \mapsto T(X)$ defines a monad T, the assignment $X \mapsto R(X)$ defines a T-module R, and src and tgt induce T-module morphisms $R \to S_1 T$ and $R \to S_2 T$, respectively. Hence we get a modular transition monad.

G Proof-irrelevant variant

▶ **Proposition 65.** *Let* $\mathbf{ITMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ *denote the full subcategory of transition monads* $\langle src, tgt\rangle : R \to S_1 T \times S_2 T$ *such that* $\langle src, tgt\rangle$ *is a pointwise inclusion. Then, the embedding* $U : \mathbf{ITMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) \hookrightarrow \mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ *is reflective.*

Proof. The left adjoint $L : \mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2) \to \mathbf{ITMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)$ maps a transition monad $\partial : R \to S_1 T \times S_2 T$ to the monomorphism $\overline{R} \hookrightarrow S_1 T \times S_2 T$ obtained from the (strong epi)-mono factorisation[5] of ∂. Then, the natural bijection $\mathbf{TMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)(T_1, UT_2) \cong \mathbf{ITMnd}_{\mathbb{P},\mathbb{S}}(T, S_1, S_2)(LT_1, T_2)$ follows from the lifting property of strong epimorphisms. ◀

Thanks to Definition 22, we then get a register for proof-irrelevant transition monads from the register **TMndReg** of Definition 63.

[5] As mentioned before, the category of finitary **Set**-valued T-modules is a presheaf category, and thus has (strong epi)-mono factorisations.

H Proof of Lemma 59

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 show that given any signature ρ in $\mathbf{Rule}(T, M)$, Σ_ρ on $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M$ is finitary.

Now, Σ_ρ is a composite of four functors as in (1) with $\prod_i (-)_{\sigma_i}$ replaced by $\mathcal{D}^\rho := \prod_i (-)^{(\vec{p_i})}_{\sigma_i}$ as explain in §5.2.2. The last three of these functors are left adjoints (because we restrict to finitary modules), hence readily finitary. It remains to show that the fourth factor, $\mathcal{D}^\rho/M : T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M \to T\text{-}\mathbf{Mod}_f(\mathbf{Set})/\mathcal{D}^\rho(M)$, is finitary. Because the domain functors $T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})/M \to T\text{-}\mathbf{Mod}_f(\mathbf{Set}^{\mathbb{S}})$ and $T\text{-}\mathbf{Mod}_f(\mathbf{Set})/\mathcal{D}^\rho(M) \to T\text{-}\mathbf{Mod}_f(\mathbf{Set})$ create colimits, this reduces to $\mathcal{D}^\rho$ 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 $\mathbf{Kl}(T)$, hence admit a right adjoint given by right Kan extension.)