



HAL
open science

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.


1 Modules over monads and operational semantics

2 André Hirschowitz 

3 Université Côte d’Azur, CNRS, Nice, France

4 Tom Hirschowitz 

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

6 Ambroise Lafont 

7 University of New South Wales, Australia

8 Abstract

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

15 **2012 ACM Subject Classification** Theory of computation \rightarrow Operational semantics

16 **Keywords and phrases** Operational semantics, Category theory

17 **Digital Object Identifier** 10.4230/LIPIcs...

18 1 Introduction

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

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

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

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

- 36 ■ We now have two kinds of terms, called *placetakers* and *states*: variables are placeholders
37 for our placetakers, while reductions relate states. Typically, in call-by-value, small-step
38 λ -calculus, placetakers are values, while states are general terms.
- 39 ■ We also have a set of types for placetakers, and a possibly different set of types for states.
40 Typically, in call-by-value, simply-typed λ -calculus, both sets of types coincide and are
41 given by simple types, while in $\lambda\mu$ -calculus, we have two placetaker types, one for terms
42 and one for stacks, and one state type, for processes.
- 43 ■ We in fact have two possibly different kinds of states, *source* states and *target* states, so
44 that a transition now relates a source state to a target state. Typically, in call-by-value,
45 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

Leibniz International Proceedings in Informatics

LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

XX:2 Modules over monads and operational semantics

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

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

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

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

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

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

66 For this, we start in §3 by proposing a new, abstract notion of *semantic signature* over
67 a category \mathbf{C} . A semantic signature $S = (\mathbf{E}, U)$ over \mathbf{C} consists of a category \mathbf{E} of *algebras*,
68 together with a *forgetful* functor $U: \mathbf{E} \rightarrow \mathbf{C}$, such that \mathbf{E} has an initial object S° : we
69 think of such a semantic signature as *specifying* the object $S^* := U(S^\circ)$ underlying the
70 initial algebra. For instance, if \mathbf{C} is cocomplete, each finitary endofunctor on \mathbf{C} generates a
71 semantic signature via its algebras. Abstracting over this generating procedure, we introduce
72 *registers* of signatures in §3. A register R for the category \mathbf{C} consists of a class \mathbf{Sig}_R of
73 *signatures*, together with a map associating to each signature S a semantic signature $\llbracket S \rrbracket_R$,
74 say $\mathbf{U}_S: S\text{-alg} \rightarrow \mathbf{C}$. Just as for semantic signatures, omitting $\llbracket - \rrbracket_R$ for readability, we
75 think of a signature S as specifying the object $S^* = \mathbf{U}_S(S^\circ)$.

76 We may now state our goal properly: construct a register for transition monads, contain-
77 ing signatures specifying the desired examples.

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

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

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

94 Related work

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

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

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

107 Notations

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

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

115 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 ,
 116 we denote by $M^{(p_1, \dots, p_n)}$ the \mathbf{D} -valued T -module defined by $M^{(p_1, \dots, p_n)}(X) = M(X + \mathbf{y}_{p_1} + \dots + \mathbf{y}_{p_n})$,
 117 where $\mathbf{y}: \mathbb{P} \rightarrow \mathbf{Set}^{\mathbb{P}}$ is the Yoneda embedding, i.e., $\mathbf{y}_p(q) = 1$ if $p = q$ and \emptyset otherwise. If \mathbb{P}
 118 is a singleton, we abbreviate this to $M^{(n)}$.

119 2 Transition monads

120 2.1 Definition of transition monads

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

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

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

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

XX:4 Modules over monads and operational semantics

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

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

144 **The state functors S_1 and S_2** While in the λ -calculus, states are just the same as
 145 placetakers, in a general transition monad, they may differ, and more precisely the two state
 146 objects are derived from the placetaker object by applying the *state functors* $S_1, S_2: \mathbf{Set}^{\mathbb{P}} \rightarrow$
 147 $\mathbf{Set}^{\mathbb{S}}$.

148 **The reductions** In standard λ -calculi, there is a (typed!) set of reductions, which yields
 149 a graph on the set of terms. That is to say, if V is the variable object, and $LC(V)$ the term
 150 object, there is a reduction object $Red(V)$ equipped with two morphisms $src, trg: Red(V) \rightarrow$
 151 $LC(V)$. Note that we consider ‘proof-relevant’ reductions here, in the sense that two different
 152 reductions have the same source and target.

153 In a general transition monad R , we still have the variable object V in $\mathbf{Set}^{\mathbb{P}}$ and the
 154 corresponding object of placetakers $T_R(V)$ also in $\mathbf{Set}^{\mathbb{P}}$, while the reduction object $Red_R(V)$
 155 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
 156 span

$$S_1(T_R(V)) \leftarrow Red_R(V) \rightarrow S_2(T_R(V)).$$

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

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

163 ▶ **Definition 2.** For any pair $S = (S_1, S_2)$ of functors $\mathbf{Set}^{\mathbb{P}} \rightarrow \mathbf{Set}^{\mathbb{S}}$, an S-graph over an object
 164 $V \in \mathbf{Set}^{\mathbb{P}}$ consists of

- 165 ■ an object E (of edges) in $\mathbf{Set}^{\mathbb{S}}$, and
- 166 ■ a span $S_1(V) \leftarrow E \rightarrow S_2(V)$, which we alternatively view as a morphism $\partial: E \rightarrow S_1(V) \times$
 167 $S_2(V)$.

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

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

$$\begin{array}{ccc} E_G & \xrightarrow{\quad s \quad} & E_{G'} \\ \partial_G \downarrow & & \downarrow \partial_{G'} \\ S_1(V_G) \times S_2(V_G) & \xrightarrow{\quad s_1(f) \times s_2(f) \quad} & S_1(V_{G'}) \times S_2(V_{G'}) \end{array}$$

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

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

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

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

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

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

197 ▶ **Definition 6.** *A modular transition monad over (\mathbb{P}, \mathbb{S}) consists of*
198 ■ *two finitary functors $S_1, S_2: \mathbf{Set}^{\mathbb{P}} \rightarrow \mathbf{Set}^{\mathbb{S}}$*
199 ■ *a finitary monad T on $\mathbf{Set}^{\mathbb{P}}$, called the placetaker monad,*
200 ■ *a finitary T -module R , called the transition module,*
201 ■ *a source T -module morphism $s_1: R \rightarrow S_1 \circ T$,*
202 ■ *a target T -module morphism $s_2: R \rightarrow S_2 \circ T$.*
203 This is the definition that we use in the following.

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

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

207 2.2 Examples of transition monads

208 2.2.1 $\overline{\lambda}\mu$ -calculus

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

211 Processes	211 Programs	211 Stacks
$c ::= \langle e \pi \rangle$	$e ::= x \mid \mu\alpha.c \mid \lambda x.e$	$\pi ::= \alpha \mid e \cdot \pi,$

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

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

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

222 For transitions, source and target states are processes, so there is only one state type:
 223 $\mathbf{S} = 1$. Furthermore, processes are pairs of a program and a stack, so that, setting $S_1(A) =$
 224 $S_2(A) = A_{\mathbf{p}} \times A_{\mathbf{s}}$, we get $S_1(T(X)) = T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}}$ as desired. Finally, transitions with free
 225 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
 226 functions $\partial_X: R(X) \rightarrow (T(X)_{\mathbf{p}} \times T(X)_{\mathbf{s}})^2$. This family is natural in X and commutes with
 227 substitution, hence forms a T -module morphism. We thus have a transition monad.

2.2.2 The π -calculus

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

235 inductive rules
$$\frac{}{\bar{a}(b).P|a(c).Q \longrightarrow P|(Q[c \mapsto b])} \quad \frac{P \longrightarrow Q}{P|R \longrightarrow Q|R} \quad \frac{P \longrightarrow Q}{\nu a.P \longrightarrow \nu a.Q}.$$

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

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

2.2.3 Positive GSOS rules

244 An example involving labelled transitions (and $S_1 \neq S_2$) is given by Positive GSOS rules [6].
 245 They specify labelled transitions $e \xrightarrow{a} f$. For any set O of operations with arities in \mathbb{N} ,

246 Positive GSOS rules have the shape
$$\frac{x_i \xrightarrow{a_{ij}} y_{ij}}{op(x_1, \dots, x_n) \xrightarrow{c} e}$$
, where the variables x_i and y_{ij} are all

247 distinct, $op \in O$ is an operation with arity n , and e is an expression potentially depending
 248 on all the variables.

249 This yields a transition monad with $\mathbb{P} = 1$, because we are in an untyped setting, and
 250 $\mathbf{S} = 1$ because states are terms. The syntax is given by the term monad T on \mathbf{Set} . For
 251 transitions, in order to take labels into account, we take $S_1(X) = X$ and $S_2(X) = \mathbb{A} \times X$,
 252 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

253 The differential λ -calculus [7] provides a further example with $S_1 \neq S_2$. Its syntax may [25,
 254 §6] be defined by

255
$$\begin{aligned} e, f &::= x \mid \lambda x.e \mid e \ U \mid De \cdot f && \text{(terms)} \\ U, V &::= \langle e_1, \dots, e_n \rangle && \text{(multiterms),} \end{aligned}$$

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

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

258 1. *Unary multiterm substitution* $e[x \mapsto U]$ in a term e , where a term variable x is replaced
 259 with a multiterm U , and which returns a multiterm (not to be confused with the monadic
 260 substitution, which handles the particular case where U is a mere term).

261 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
 262 again returns a multiterm.

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

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

$$269 \quad (\lambda x.e) U \rightarrow e[x \mapsto U] \quad D(\lambda x.e) \cdot f \rightarrow \lambda x. \left(\frac{\partial e}{\partial x} \cdot f \right)$$

270 The second rule relies on the abbreviation $\lambda x.\langle e_1, \dots, e_n \rangle := \langle \lambda x.e_1, \dots, \lambda x.e_n \rangle$. We work with
 271 $\mathbb{P} = \mathbb{S} = 1$, i.e., only one placetaker and state type; S_1 is the identity; and $S_2 = !$.

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

273 2.2.5 Call-by-value, simply-typed λ -calculus, big-step style

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

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

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

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

288 2.3 Categories of transition monads

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

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

296 This gives a first family of categories of transition monads, that we will integrate through
 297 a simple construction¹:

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.

XX:8 Modules over monads and operational semantics

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

301 The relevant example for the present work is the following.

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

- 304 ■ 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
305 on $\mathbf{Set}^{\mathbb{P}}$ with two copies of the functor category $[\mathbf{Set}^{\mathbb{P}}, \mathbf{Set}^{\mathbb{S}}]$;
- 306 ■ 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.

307 3 Semantic signatures and registers

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

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

315 3.1 Semantic signatures

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

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

- 318 ■ a category $S\text{-alg}$ of models of S (or algebras), which admits an initial object, denoted by
319 S^\circledast , and
- 320 ■ a forgetful functor $\mathbf{U}_S: S\text{-alg} \rightarrow \mathbf{C}$.

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

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

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

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

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

335 ▶ **Notation 16.** We denote by $UR_{\mathbf{C}}$ the class of semantic signatures over the category \mathbf{C} (UR
336 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 $\mathbf{Mnd}(\mathbf{Set}^{\mathbf{P}}) \times [\mathbf{Set}^{\mathbf{P}}, \mathbf{Set}^{\mathbf{S}}]^2$ (Definition 10), which is the base category of our record category of transition monads.

► **Lemma 17.** *Given a set I and functors $U_i: \mathbf{E}_i \rightarrow \mathbf{C}_i$ 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 $u_i: \mathbf{E}_i \rightarrow \mathbf{C}_i$, the product $\prod_i u_i: \prod_i \mathbf{E}_i \rightarrow \prod_i \mathbf{C}_i$ is a semantic signature. This defines our external product of signatures $\prod_I: \prod_i UR_{\mathbf{C}_i} \rightarrow UR_{\prod_i \mathbf{C}_i}$.*

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 $\llbracket - \rrbracket_R: \mathbf{Sig}_R \rightarrow UR_{\mathbf{C}}$.

We can now motivate the notation $UR_{\mathbf{C}}$ 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 $\llbracket - \rrbracket_{UR_{\mathbf{C}}}$ is the identity (on $UR_{\mathbf{C}}$).

► **Notation 21.** When convenient, we introduce a register as $u: \mathbf{S} \rightarrow UR_{\mathbf{C}}$ to be understood as the register R with $\mathbf{Sig}_R := \mathbf{S}$ and $\llbracket - \rrbracket_R := u$. Moreover, we sometimes omit $\llbracket - \rrbracket_{UR_{\mathbf{C}}}$, thus identifying any $s \in \mathbf{Sig}_R$ with its associated semantic signature $\llbracket s \rrbracket_R$.

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 $[\mathbf{C}, \mathbf{C}] \rightarrow UR_{\mathbf{C}}$ sending any finitary endofunctor F to the forgetful functor $F\text{-alg} \rightarrow \mathbf{C}$ from its category of algebras.*

Let us now define simple constructions on registers.

► **Proposition 23.** *For any register R for \mathbf{C} and functor $F: \mathbf{C} \rightarrow \mathbf{D}$, postcomposition of semantic signatures with F induces a register $\Sigma_F(R)$ 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 $f: \mathbf{S} \rightarrow \mathbf{Sig}_R$, 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 $f: \mathbf{S} \rightarrow \mathbf{Sig}_{UEF_{\mathbf{C}}}$.*

XX:10 Modules over monads and operational semantics

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

378 ▶ **Definition 27.** For any endofunctorial register $R = \Delta_f(\text{UEF}_{\mathbf{C}})$, let R^* denote the endo-
379 functorial register whose signatures are families of signatures in \mathbf{Sig}_R , and whose semantics
380 maps any family to the coproduct of associated endofunctors.

381 A final basic construction on registers is the external product.

382 ▶ **Definition 28.** The product of a family $(u_i: S_i \rightarrow \text{UR}_{\mathbf{C}_i})_{i \in I}$ of registers is obtained by post-
383 composing $\prod_i u_i$ with the product of semantic signatures: $\prod_i S_i \xrightarrow{\prod_i u_i} \prod_i \text{UR}_{\mathbf{C}_i} \xrightarrow{\prod_i} \text{UR}_{\prod_i \mathbf{C}_i}$.

384 ▶ **Example 29.** The product register $\text{OMnd}_b := \mathbf{MonReg}(\mathbf{Set}^{\mathbf{P}}) \times \text{EqSys}[\mathbf{Set}^{\mathbf{P}}, \mathbf{Set}^{\mathbf{S}}]^2$ for
385 monads and state functors allows us to specify the base components of our transition monads.

386 3.4 Equational registers

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

$$X^n \xrightarrow{\langle e!, \Sigma_n^*, X \rangle} \Sigma^*(n) \times [\Sigma^*(n), \Sigma^*(X)] \xrightarrow{ev} \Sigma^*(X) \rightarrow X,$$

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

389 Furthermore, this technique generalises to families $(t_i = u_i)_i$ of equations, with each
390 $t_i, u_i \in \Sigma^*(n_i)$, by replacing $(-)^n$ with the coproduct functor $\sum_i (-)^{n_i}$. This works even for the
391 empty family of course.

392 Equational systems are obtained by abstracting over this situation.

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

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

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

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

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

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

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

410 ▶ **Example 35.** The state functor $S(X) = X_p \times X_s$ from §2.2.1 is specified by the equational
 411 system defined on $[\mathbf{Set}^2, \mathbf{Set}]$ by $\Sigma(F)(X) = X_p \times X_s$ and no equation. Algebras are functors
 412 F equipped with a natural transformation $X_p \times X_s \rightarrow F(X)$, so the initial algebra is clearly
 413 $X_p \times X_s$ itself.

414 ▶ **Example 36.** The first state functor of call-by-value, simply-typed λ -calculus could be
 415 specified by taking $\Sigma(F)(X) = S_1(X)$, with S_1 as in §2.2.5, and no equation. However, let us
 416 observe that it may also be specified by the simpler endofunctor $\Sigma(F)(X)_t = X_t + \sum_{t'} F(X)_{t' \rightarrow t} \times$
 417 $F(X)_{t'}$ with no equation.

4 Registers for monads and slice module categories

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

4.1 A register for monads

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

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

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

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

429 ▶ **Example 38.** For pure λ -calculus, the relevant endofunctor on $[\mathbf{Set}, \mathbf{Set}]$ is $\Sigma(F)(X) =$
 430 $X + F(F(X)) + F(X)^2 + F(X + 1)$.

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

432 ▶ **Example 39.** We enforce the usual equation $(M N)[\sigma] = M[\sigma] N[\sigma]$ through the equation
 433 $L = R: \Sigma\text{-alg} \rightarrow \Gamma\text{-alg}$, where $\Gamma(F)(X) = F(F(X))^2$, and the structure maps of $L(\Sigma(F) \xrightarrow{\rho} F)$
 434 and $R(\Sigma(F) \xrightarrow{\rho} F)$ at X are respectively: $F(F(X))^2 \xrightarrow{\textcircled{F}(X)} F(F(X)) \xrightarrow{\mu_X} F(X)$ and $F(F(X))^2 \xrightarrow{\mu^2}$
 435 $F(X)^2 \xrightarrow{\textcircled{X}} F(X)$. The maps $\textcircled{}$ and μ follow from the Σ -algebra structure ρ on F . E.g., \textcircled{X}
 436 is defined as the composite $F(X)^2 \xrightarrow{\text{in}_3} \Sigma(F)(X) \xrightarrow{\rho_X} F(X)$.

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

439 Let us now describe the general pattern, by defining a register for monads. The idea is
 440 that a signature should be an equational system on endofunctors of the considered category
 441 \mathbf{C} whose operation endofunctor Σ contains Σ_{Mnd} , and whose equations include the monad
 442 equations. One way of enforcing this consists in asking the endofunctors to have the shape
 443 $\Sigma_{Mnd} + \Sigma$ and $\Gamma_{Mnd} + \Gamma$. For equations, we rely on the following well-known fact.

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

XX:12 Modules over monads and operational semantics

446 Thus, given functors $L_1: \mathbf{D} \rightarrow F\text{-alg}$ and $L_2: \mathbf{D} \rightarrow G\text{-alg}$ mapping objects and morphisms of \mathbf{D} to the same underlying objects and morphisms of \mathbf{C} , we may form their pairing
 447 $\langle L_1, L_2 \rangle: \mathbf{D} \rightarrow (F + G)\text{-alg}$. Denoting by \downarrow_F the forgetful functor $(F + G)\text{-alg} \rightarrow F\text{-alg}$, we
 448 state:
 449

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

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

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

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

4.2 Registers for slice module categories

459 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
 460 M . We then define an endofunctorial register $\mathbf{Rule}_{T,M}^*$ for the category $T\text{-Mod}/M$. Later
 461 on, we will use this register with $M := (S_1 \circ T) \times (S_2 \circ T)$, i.e., for the category of transition
 462 monads over (T, S_1, S_2) .

463 **The naive register** We start by defining a much simpler endofunctorial sub-register,
 464 $\mathbf{NRule}_{T,M}$. A signature in $\mathbf{NRule}_{T,M}$ consists of

- 465 ■ a *metavariable* module V ,
- 466 ■ a *conclusion* module morphism $t: V \rightarrow M_\tau$ for some *conclusion* state type $\tau \in \mathbb{S}$, and
- 467 ■ a list of *premise* module morphisms of the form $s: V \rightarrow M_\sigma$, for some *premise* state types
 468 $\sigma \in \mathbb{S}$.

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

$$472 \quad \begin{array}{ccc} T^3 & \rightarrow & T^2 \\ (e, e', f) & \mapsto & (e f, e' f) \end{array} \quad \text{and} \quad \begin{array}{ccc} T^3 & \rightarrow & T^2 \\ (e, e', f) & \mapsto & (e, e'). \end{array}$$

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

$$474 \quad \begin{array}{ccc} T\text{-Mod}(\mathbf{Set})/\prod_i M_{\sigma_i} & \xrightarrow{\Delta_{\langle s_i \rangle_i}} & T\text{-Mod}(\mathbf{Set})/V \xrightarrow{\Sigma_t} T\text{-Mod}(\mathbf{Set})/M_\tau \\ \Pi_i(-)_{\sigma_i} \uparrow & & \downarrow \\ T\text{-Mod}(\mathbf{Set}^{\mathbb{S}})/M & & T\text{-Mod}(\mathbf{Set}^{\mathbb{S}})/M, \end{array}$$

475 of four functors, where

- 476 ■ $\prod_i(\partial: R \rightarrow M)_{\sigma_i}$ denotes $\prod_i \partial_{\sigma_i}: \prod_i R_{\sigma_i} \rightarrow \prod_i M_{\sigma_i}$,
- 477 ■ $\Delta_{\langle s_i \rangle_i}$ is defined by pullback along the tupling $\langle s_i \rangle_i: V \rightarrow \prod_i M_{\sigma_i}$ of all premises,
- 478 ■ Σ_i is defined by postcomposition with the conclusion $t: V \rightarrow M_\tau$.

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

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

$$\begin{array}{ccc}
 \prod_i R_{\sigma_i} & \longleftarrow & P \\
 \Pi_i \partial_{\sigma_i} \downarrow & & \downarrow \\
 \prod_i M_{\sigma_i} & \xleftarrow{\langle s_i \rangle_i} & V \xrightarrow{t} M_\tau
 \end{array}
 \begin{array}{c}
 \swarrow q \\
 \searrow
 \end{array}
 \quad (1)$$

484 ▶ **Proposition 46.** The assignment $S \mapsto \Sigma_S$ defines a register $\mathit{NRule}_{T,M}$ for $T\text{-Mod}(\mathit{Set}^{\mathbf{S}})/M$.

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

- 487 ■ 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
- 488 r is a transition $e \rightarrow e'$,
- 489 ■ with projection to T^2 mapping any (r, e, e', f) to $(e f, e' f)$.

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

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

494 ▶ **Example 48.** Consider the ξ rule of pure λ -calculus: $\frac{e \rightarrow f}{\lambda x. e \rightarrow \lambda x. f}$,

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

498 ▶ **Definition 49.** The register $\mathit{Rule}_{T,M}$ is defined by:

- 499 ■ signatures are just as in $\mathit{NRule}_{T,M}$, except that the premises now have the shape $s: V \rightarrow$
 500 $M_\sigma^{(\vec{p})}$, for $\sigma \in \mathbf{S}$ and \vec{p} a list of placetaker types; and
- 501 ■ the semantics is defined exactly as for naive rules, replacing $\prod_i R_i$ with $\prod_i R_i^{(\vec{p}^i)}$.

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

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

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

507 5 Record registers

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

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

XX:14 Modules over monads and operational semantics

519 ▶ **Definition 52.** An indexed register (R_b, R_f) for a record category $\sum_b \mathbf{P}(b)$, with $\mathbf{P}: \mathbf{ob}(\mathbf{B}) \rightarrow$
 520 \mathbf{CAT} , consists of
 521 ■ a base register R_b for \mathbf{B} , together with,
 522 ■ for each signature B in \mathbf{Sig}_{R_b} , a fibre register $R_f(B)$ for the fibre \mathbf{P}_{B^*} over the initial
 523 B -algebra.

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

528 From any fixed indexed register $R = (R_b, R_f)$ for K , let us now construct a proper register
 529 $\sum R$, which we call the *record register* of R . First of all, let us define the signatures of $\sum R$.

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

531 ▶ **Example 55.** A signature record for the indexed register $OMnd$ from Example 53 consists
 532 of a triple $\Sigma = (\Sigma_0, \Sigma_1, \Sigma_2)$ of signatures, specifying a monad $T = \Sigma_0^*$ and state functors
 533 $S_i = \Sigma_i^*$, $i = 1, 2$, together with a family \mathbf{R} of binding $(T, \overline{S_1} \times \overline{S_2})$ -rules.

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

535 ▶ **Definition 56.** Given a signature record (B, F) for some indexed register $\sum_b \mathbf{P}$ for a record category
 536 $K = \sum_b \mathbf{P}_b$, the semantic signature $\sum(B, F)$ associated to (B, F) is $F\text{-alg} \xrightarrow{\mathbf{U}_F} \mathbf{P}_{B^*} \hookrightarrow K$.

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

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

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

6 Applications

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

547 In this section, we present in some detail the signature for differential λ -calculus, as
 548 a transition monad with $\mathbf{P} = \mathbf{S} = 1$, introduced in §2.2.4. A signature in the register of
 549 transition monads consists of two components: a (product) signature for the state functors
 550 and monad, given in §6.1, and a signature for the β and ∂ -reduction rules. Both are straight-
 551 forwardly modelled by a signature over as explained in §4.2, but they first require us to
 552 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

554 The first state functor is the identity functor $\text{Id}: \mathbf{Set} \rightarrow \mathbf{Set}$, and thus is specified by the
 555 signature of Example 34. The second state functor is $!$, the multiset functor, and is specified
 556 by an equational system $(\Sigma_2 \triangleright \Gamma_2 \vdash L_2 = R_2)$ on $[\mathbf{Set}, \mathbf{Set}]$, where $\Sigma_2(F)(X) = X + F(X) \times$
 557 $F(X) + 1$, so that an algebra of Σ_2 is an endofunctor equipped with a binary operation and

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

560 Next, the monad of differential λ -calculus is specified by a monadic equational system
561 $((\Sigma_{Mnd} + \Sigma) \triangleright (\Gamma_{Mnd} + \Gamma) \vdash \langle L_{Mnd^\circ} \downarrow_{\Sigma_{Mnd}}, L \rangle = \langle R_{Mnd^\circ} \downarrow_{\Sigma_{Mnd}}, R \rangle)$ on **[Set, Set]**, which we now
562 define.

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

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

567 6.2 Intermediate constructions for differential λ -calculus

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

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

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

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

583 This recipe relies on the following lemma:

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

586 7 Conclusion and perspectives

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

595 References

- 596 1 Benedikt Ahrens. Modules over relative monads for syntax and semantics. *MSCS*, 26:3–
597 37, 2016.

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

- 598 **2** Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Modular
599 specification of monads through higher-order presentations. In Herman Geuvers, editor,
600 *Proc. 4th Int. Conf. on Formal Structures for Comp. and Deduction*, LIPIcs. Schloss
601 Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019.
- 602 **3** Benedikt Ahrens, André Hirschowitz, Ambroise Lafont, and Marco Maggesi. Re-
603 duction monads and their signatures. *PACMPL*, 4(POPL):31:1–31:29, 2020. doi:
604 10.1145/3371099.
- 605 **4** Thorsten Altenkirch, James Chapman, and Tarmo Uustalu. Monads need not be endo-
606 functors. *Logical Methods in Computer Science*, 11(1), 2015. doi:10.2168/LMCS-11(1:
607 3)2015.
- 608 **5** Thorsten Altenkirch, Peter Morris, Fredrik Nordvall Forsberg, and Anton Setzer. A
609 categorical semantics for inductive-inductive definitions. In Andrea Corradini, Bartek
610 Klin, and Corina Cirstea, editors, *Proc. 4th Alg. and Coalgebra in Comp. Sci.*, volume
611 6859 of *LNCS*, pages 70–84. Springer, 2011. doi:10.1007/978-3-642-22944-2_6.
- 612 **6** Bard Bloom, Sorin Istrail, and Albert R. Meyer. Bisimulation can’t be traced. *J. ACM*,
613 42(1):232–268, 1995. doi:10.1145/200836.200876.
- 614 **7** Thomas Ehrhard and Laurent Regnier. The differential lambda-calculus. *TCS*, 309:1–41,
615 2003.
- 616 **8** Marcelo Fiore and Chung-Kil Hur. On the construction of free algebras for equational
617 systems. *TCS*, 410:1704–1729, 2009.
- 618 **9** Marcelo P. Fiore. Second-order and dependently-sorted abstract syntax. In *Proc. 23rd*
619 *Logic in Comp. Sci.*, pages 57–68. IEEE, 2008. doi:10.1109/LICS.2008.38.
- 620 **10** Marcelo P. Fiore and Sam Staton. A congruence rule format for name-passing process
621 calculi from mathematical structural operational semantics. In *Proc. 21st Logic in Comp.*
622 *Sci.*, pages 49–58. IEEE, 2006. doi:10.1109/LICS.2006.7.
- 623 **11** Marcelo P. Fiore and Daniele Turi. Semantics of name and value passing. In *Proc. 16th*
624 *Logic in Comp. Sci.*, pages 93–104. IEEE, 2001. doi:10.1109/LICS.2001.932486.
- 625 **12** Richard Garner. Combinatorial structure of type dependency. *CoRR*, abs/1402.6799,
626 2014. URL: <http://arxiv.org/abs/1402.6799>, arXiv:1402.6799.
- 627 **13** Makoto Hamana. Term rewriting with variable binding: An initial algebra approach. In
628 *Proc. 5th Princ. and Practice of Decl. Prog.* ACM, 2003. doi:10.1145/888251.888266.
- 629 **14** Hugo Herbelin. *Séquents qu’on calcule: de l’interprétation du calcul des séquents*
630 *comme calcul de lambda-termes et comme calcul de stratégies gagnantes*. PhD thesis,
631 Paris Diderot University, France, 1995. URL: [https://tel.archives-ouvertes.fr/
632 tel-00382528](https://tel.archives-ouvertes.fr/tel-00382528).
- 633 **15** André Hirschowitz and Marco Maggesi. Modules over monads and linearity. In *WoLLIC*,
634 volume 4576 of *LNCS*, pages 218–237. Springer, 2007. doi:10.1007/3-540-44802-0_3.
- 635 **16** Tom Hirschowitz. Cartesian closed 2-categories and permutation equivalence in higher-
636 order rewriting. *LMCS*, 9(3), 2013. URL: [http://hal.archives-ouvertes.fr/
637 hal-00540205](http://hal.archives-ouvertes.fr/hal-00540205), doi:10.2168/LMCS-9(3:10)2013.
- 638 **17** Tom Hirschowitz. Cellular monads from positive GSOS specifications. In Jorge A.
639 Pérez and Jurriaan Rot, editors, *Proc. 26th International Workshop on Expressiveness in*
640 *Concurrency and 16th Workshop on Structural Operational Semantics, EXPRESS/SOS*,
641 volume 300 of *EPTCS*, pages 1–18, 2019. doi:10.4204/EPTCS.300.1.
- 642 **18** Tom Hirschowitz. Familial monads and structural operational semantics. *PACMPL*,
643 3(POPL):21:1–21:28, 2019. URL: <https://dl.acm.org/citation.cfm?id=3290334>.
- 644 **19** Ambroise Lafont. *Signatures and models for syntax and operational semantics in the*
645 *presence of variable binding*. PhD thesis, École Nationale Supérieure Mines – Telecom

- 646 Atlantique Bretagne Pays de la Loire – IMT Atlantique, 2019. URL: <https://arxiv.org/abs/1910.09162v2>.
- 647
- 648 **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.
- 649
- 650 **21** Jan Reiterman. A left adjoint construction related to free triples. *JPAA*, 10:57–71, 1977.
- 651 **22** Davide Sangiorgi and David Walker. *The π -calculus - a theory of mobile processes*. Cambridge University Press, 2001.
- 652
- 653 **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.
- 654
- 655 **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.
- 656
- 657 **25** Lionel Vaux. *λ -calcul différentiel et logique classique : interactions calculatoires*. PhD thesis, Université Aix-Marseille 2, 2007.
- 658

659	Contents	
660	1 Introduction	1
661	2 Transition monads	3
662	2.1 Definition of transition monads	3
663	2.2 Examples of transition monads	5
664	2.2.1 $\bar{\lambda}\mu$ -calculus	5
665	2.2.2 The π -calculus	6
666	2.2.3 Positive GSOS rules	6
667	2.2.4 Differential λ -calculus	6
668	2.2.5 Call-by-value, simply-typed λ -calculus, big-step style	7
669	2.3 Categories of transition monads	7
670	3 Semantic signatures and registers	8
671	3.1 Semantic signatures	8
672	3.2 The external product of semantic signatures	9
673	3.3 Registers of signatures	9
674	3.4 Equational registers	10
675	4 Registers for monads and slice module categories	11
676	4.1 A register for monads	11
677	4.2 Registers for slice module categories	12
678	5 Record registers	13
679	6 Applications	14
680	6.1 State functors and monad of differential λ -calculus	14
681	6.2 Intermediate constructions for differential λ -calculus	15
682	7 Conclusion and perspectives	15
683	A Proof of Proposition 50	19

A

 Proof of Proposition 50

684

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

688 As coproducts of finitary functors are finitary, it is enough to show that given one (T, M) -
 689 rule ρ , the endofunctor Σ_{ρ} is finitary. Such a rule comes with a conclusion module morphism
 690 $V \rightarrow M_{\tau}$, and a list of premise module morphisms $(V \rightarrow M_{\sigma_i}^{\vec{p}_i})_{i \in n}$.

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

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

694

695 where \mathcal{D}^{ρ}/M maps any $\partial: R \rightarrow M$ to $\mathcal{D}^{\rho}(\partial): \mathcal{D}^{\rho}(R) \rightarrow \mathcal{D}^{\rho}(M)$, and Δ and Σ respectively
 696 denote pullback and postcomposition functors.

697 Now, Σ_{ρ} is a composite of four functors, three of which are left adjoints (because we
 698 restrict to finitary), hence readily finitary. It remains to show that the fourth factor,
 699 \mathcal{D}^{ρ}/M , is finitary. Because the domain functors $T_0\text{-Mod}(\mathbf{Set}^{\mathcal{S}})/M \rightarrow T_0\text{-Mod}(\mathbf{Set}^{\mathcal{S}})$ and
 700 $T_0\text{-Mod}(\mathbf{Set})/\mathcal{D}^{\rho}(M) \rightarrow T_0\text{-Mod}(\mathbf{Set})$ create colimits, this reduces to \mathcal{D}^{ρ} being finitary.
 701 But finitary functors are closed under finite products, so, because colimits are pointwise in
 702 presheaf categories, this in turn reduces to each $(-)^{\vec{p}_i}$ being finitary, which follows from their
 703 being left adjoints. (They may be viewed as precomposition with an endofunctor of $\mathbf{Kl}(T_0)$,
 704 hence admit a right adjoint given by right Kan extension.)

XX:20 REFERENCES

705 **Contents**