

Modules over monads and operational semantics

André Hirschowitz¹ Tom Hirschowitz² Ambroise Lafont³

¹Université Côte d'Azur, CNRS, Nice, France

²Univ. Grenoble Alpes, Univ. Savoie Mont Blanc, CNRS, LAMA, France

³University of New South Wales, Australia

FSCD 2020

Summary

What is a programming language (**syntax** and **semantics**), mathematically?

This contribution:

- a notion of programming language: **transition monads**;
- a discipline for **automatically generating** them;
- generalises the reduction monads of Ahrens et al. (POPL 20)
 - new applications:

$\bar{\lambda}\mu$ -calculus	π -calculus
cbv λ -calculus	differential λ -calculus
GSOS specifications	
 - simply-typed syntax (and semantics)

Related work

- *Reduction monads* of Ahrens et al. (POPL 2020)
 - ✗ untyped
 - ✗ cbv λ -calculus out of reach
 - **this contribution** = a generalisation of *reduction monads*
- *Mathematical Operational Semantics* (Turi-Plotkin '97)
 - Deeply developed
 - ✗ Higher-order languages (such as λ -calculus) only starting to be investigated (Peressoti '17)
- Rewriting with variable binding (categorical approach)
 - e.g. Hamana 2003, T. Hirschowitz 2013, Ahrens 2016
 - ✗ only congruent transitions \Rightarrow weak reduction out of reach

Outline

- 1 Definition of transition monads
- 2 Generating transition monads (Initial Semantics)
- 3 Examples

Basic components of a *transition monad*

Example: (small-step) cbv λ -calculus.

A **term** t, u, \dots reduces to a **value** v, w, \dots

Monad of values

- $Variables \subset Values$
- $v[\vec{w}/\vec{x}] \in Values$ (value substitution)

Modules of terms and transitions

- value substitution *in terms*:

$$t[\vec{w}/\vec{x}] \in Terms$$

- value substitution *in transitions*:

$$\frac{t \rightarrow v}{t[\vec{w}/\vec{x}] \rightarrow v[\vec{w}/\vec{x}]}$$

Generalising cbv λ -calculus, and reduction monads

	Syntax	Semantics
cbv λ -calculus	<i>Values</i> (monad)	
transition monads	a monad T	T -module morphisms $M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2$
reduction monads (Ahrens et al. '20)	a monad T	$T \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} T$

- Untyped case: base category = Set
- Simply-typed case: base category = $\text{Set}^{\text{Types}}$

Quick definition of T -modules

(given a monad T)

Components of a T -module M

- a functor M
- a natural transformation $MT \xrightarrow{m} M$
 \Rightarrow **substitution** (see below)
- (+ some substitution laws)

Substitution for modules

- $m \in M(X) =$ “ m takes free variables in X ”
- $m[x \mapsto t_x] =$ “simultaneous substitution in m of every $x \in X$ with $t_x \in T(Y)$ ”

$$\begin{array}{ccccc}
 X \xrightarrow{t_-} T(Y) & & M(X) \xrightarrow{Mt_-} M(T(Y)) \xrightarrow{m_Y} M(Y) & & \\
 & & m \longmapsto & & m[x \mapsto t_x]
 \end{array}$$

Constructing *transition monads*

We have a definition of programming languages as **transition monads**.

Can we construct them from *simple specifications*?

We provide:

- a notion of *simple specification* = **signature** for transition monads
- a theorem ensuring the existence (unique up to iso) of a transition monad matching a spec

Three-level specification

Transition monad = $(T, M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2)$

Three spec steps:

Step	Component	Nature	Specification
1	T	monad (syntax)	Operations + Equations
2	M_1, M_2	T -modules	Operations + Equations
3	$\text{Trans},$ $\text{source},$ target	“transition structure”	Transition rules as $\frac{t_1 \rightarrow u_1 \dots t_n \rightarrow u_n}{t \rightarrow u}$

⇒ Three notions of signatures in the paper (one for each step), building upon

- *reduction monads*, Ahrens et al. POPL '20
- (steps 1-2) previous work by Fiore-Hur (*equational systems*) and Ahrens et al. (*signatures for monads*).

Examples

Transition monad = $(T, M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2)$

Upcoming examples

1.	cbn λ -calculus	full signature (sketched)
2.	cbn λ -calculus	signature for T
3.	cbn λ -calculus	left congruence rule for application
4.	cbn λ -calculus	congruence rule for abstraction (involves a binding variable)
5.	cbv λ -calculus	signature for M_i
6.	differential λ -calculus	signature for M_i
7.	differential λ -calculus	signature for T

Example 1/7: small-step cbn λ -calculus

Transition monad = (T , $M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2$)

Signature for cbn λ -calculus

Step	Component	Nature	Specification
1	T	monad	Operations = app, abs
2	M_1, M_2	T -modules	$M_1 = M_2 = T$
3	$\text{Trans},$ $\text{source},$ target	"transition structure"	β -rule + congruences

Example 2/7: Specify the monad of λ -terms

cbn λ -calculus: $(\text{Values}, T \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} T)$

- Syntax “**generated**” by

application	$T \times T \rightarrow T$
λ -abstraction $\lambda x.t$	$T' \rightarrow T$ $T' =$ module of terms depending on an extra variable
(variables)	$Var \rightarrow T$

Signature for T

2 operations (application/abstraction)

- Monads always have variables: no need to specify them
- “operation” = *module morphism*: compatible with substitution:

$$(t_1 t_2)[y \mapsto u_y] = t_1[y \mapsto u_y] t_2[y \mapsto u_y]$$

References “Second-order equational logic” (Fiore-Hur ’10),
“Modular specification of monads” (Ahrens et al. ’19)

Example 3/7: Left congruence for application

cbn λ -calculus: $(T, T \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} T)$

Left congruence rule for application

$$\frac{t_1 \rightarrow t_2}{\text{app}(t_1, u) \rightarrow \text{app}(t_2, u)}$$

- Easy interpretation¹ of transition rules:

Components of the rule	Interpreted as...
3 “metavariables”: t_1, t_2, u	a “metavariable” T -module $V = T \times T \times T$
1 “premise”: $t_1 \rightarrow t_2$	$V \rightarrow M_1 \times M_2$ (T -module morphism) $(t_1, t_2, u) \mapsto (t_1, t_2)$
“conclusion”: $\text{app}(t_1, u) \rightarrow \text{app}(t_2, u)$	$V \rightarrow M_1 \times M_2$ $(t_1, t_2, u) \mapsto (\text{app}(t_1, u), \text{app}(t_2, u))$

¹introduced by Ahrens et al. in the case of reduction monads.

Example 4/7: Binding variables in rules

cbn λ -calculus: $(T, T \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} T)$

Congruence rule for abstraction

$$\frac{t_1 \rightarrow t_2}{\lambda x. t_1 \rightarrow \lambda x. t_2}$$

- “metavariables” t_1 and t_2 : terms that may depend on x .
- $T' = T$ -module of terms depending on an additional variable

(in fact, abstraction is an operation $T' \xrightarrow{\text{abs}} T$)

Components of the rule	Interpreted as...
2 “metavariables”: t_1, t_2	a “metavariable” T -module $V = T' \times T'$
...	...

Example 5/7: Specify M_i for cbv

$$\begin{aligned}\text{Transition monad} &= (T, \quad M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2) \\ \text{cbv } \lambda\text{-calculus} &= (\text{Vals}, Tms \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} \text{Vals})\end{aligned}$$

Syntax of values and terms

$\text{Vals} : v, w ::= x \mid \lambda x. t$

$Tms : t, u ::= x \mid \lambda x. t \mid \underbrace{t u}_v \Rightarrow \begin{aligned} \text{terms} &= \text{binary trees of values} \\ Tms &= \text{BinTree} \circ \text{Vals} \end{aligned}$

In fact, by definition of a transition monad,

- M_i is always of the shape $S_i \circ T$. Here,

$$T = \text{Vals} \qquad M_1 = \text{BinTree} \circ T \qquad M_2 = \text{Id} \circ T (= T)$$

- Signature for M_i = Signature for S_i

Signature for BinTree

1 binary operation (accounts for $t u$ in Tms), no equation

Example 6/7: Specifying M_i for DLC

Transition monad = $(T, M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2)$

Differential λ -calculus (DLC)

Syntax monad T of terms (a variant of λ -calculus)

Semantics a term t reduces to a multiterm $t_1 + \dots + t_n$

$$M_1 = \text{Id} \circ T (= T)$$

$$M_2 = \text{FormalSum} \circ T$$

Signature for *FormalSum*

Operations	a constant 0, a binary operation +
Equations	commutativity, associativity, unitality

Example 7/7: the monad of DLC

differential λ -calculus: $(T, M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2)$

- Syntax of DLC = variant of λ -calculus

Application of DLC

$$\text{app} : (t, U) \mapsto t U$$

input of app = a term t **and** a multi-term $U = u_1 + \dots + u_n$
 = a term **and** a formal sum **of** terms

input *module* of app = $T \times (\text{FormalSum} \circ T)$

Signature for T

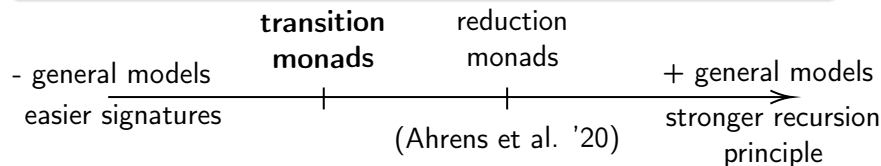
3 operations (no equation):

application $t U$	$T \times (\text{FormalSum} \circ T) \rightarrow T$
differential application $Dt \cdot u$	$T \times T \rightarrow T$
λ -abstraction	(as before)

Future work: strengthen the recursion principle

Initial semantics (general framework)

- specified object by a signature $\Sigma = \textit{initial object}$ in the category of *models* of Σ
- *initiality* \Rightarrow recursion principle



Future work alternative notion of signatures with more general models (as in Ahrens et al. '20)
 \Rightarrow stronger recursion principle

A difficulty with general models à la Ahrens et al. '20: DLC

$(T, M_1 \leftarrow Trans \rightarrow M_2)$ specified by a 3-step signature

component	Σ_1	Σ_2	Σ_3
specifies	T	M_1, M_2	$\leftarrow Trans \rightarrow$

Models of $(\Sigma_1, \Sigma_2, \Sigma_3)$

transition monads $(T, M_1 \leftarrow Trans \rightarrow M_2)$ + extra structure, s.t.

T = 'the' initial model of Σ_1

(in Ahrens et al. '20, T = any model of Σ_1)

(M_1, M_2) = 'the' initial model of Σ_2

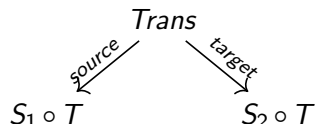
Specifying the transition rules of DLC

transitions involve intermediary syntactic constructions

T = 'the' initial model of Σ_1	T = any model of Σ_1
define them by recursion	recursion not available!

Conclusion

- A **mild** generalisation of Ahrens et al.'s *reduction monads* '20
 ⇒ new notion of programming language:

	Syntax	Semantics
transition monads	a monad T	T -module morphisms 

- Associated notion of specification
 - with easy interpretation of transition rules
- **Numerous** new examples:

$\bar{\lambda}\mu$ -calculus
cbv λ -calculus

π -calculus
differential λ -calculus

GSOS specifications