

Mathematical specification of programming languages using monads and modules over them

Ambroise Lafont¹

¹University of New South Wales
Sydney, Australia

November 26, 2020

That is the question

What is a programming language, mathematically?

Differential λ -calculus [Ehrhard-Regnier '03]

~10 pages (section 2 \rightarrow beginning of section 3) describing the programming language and proving some [properties](#).

Contributions presented in this talk

- a notion of programming language, [transition monads](#) [Hirschowitz-Hirschowitz-Lafont '20], and
- a discipline for [automatically generating](#) them.

Features of this approach

- monads and modules to take care of substitution.
- works with simple types (in this talk: untyped case)

Related work

Syntax with variable binding

Two main notions of syntax:

- ① **Nominal sets** [Gabbay-Pitts '99].
 - Injective renamings of variables built-in.
- ② **Substitution monoids** [Fiore-Plotkin-Turi '99].
 - Simultaneous substitution of variables built-in.
 - Transition monads (syntax): a variant of this approach

Related work

Notion of programming language and specification

- *Reduction monads* [Ahrens-Hirschowitz-Lafont-Maggesi '20]
 - cbv λ -calculus out of reach
 - + **transition monads** = 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 [Peressotti '17]
- Rewriting with variable binding (categorical approach)
 - e.g. [Hamana '03, Hirschowitz '13, Ahrens '16]
 - only congruent transitions \Rightarrow weak reduction out of reach

Examples of transition monads

- cbv/cbn λ -calculus (big/small-step)
- $\bar{\lambda}\mu$ -calculus
- π -calculus
- differential λ -calculus
- computational λ -calculus (variant of [Dal Lago-Gavazzo-Levy '17])
- GSOS systems

What is still missing

Metatheorems, e.g., congruence of bisimilarity

- [Borthelle-Hirschowitz-Lafont '20], Howe's method in a different setting.
- [Dal Lago-Gavazzo-Levy '17], Howe's method for particular cases of transition monads (computational λ -calculus).
- Can we generalize both approaches?

Limitations

- π -calculus as a labelled transition system?
- signature for the computational λ -calculus?
- simple types ok
 - linear types?
 - polymorphic/dependent types?
 - subtyping?

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

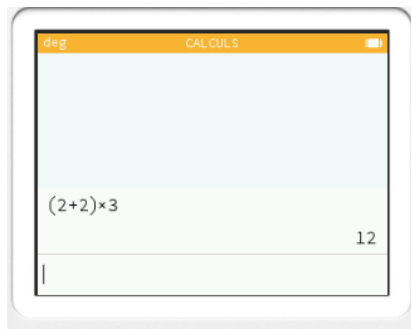
Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

What is a programming language?

2 components:

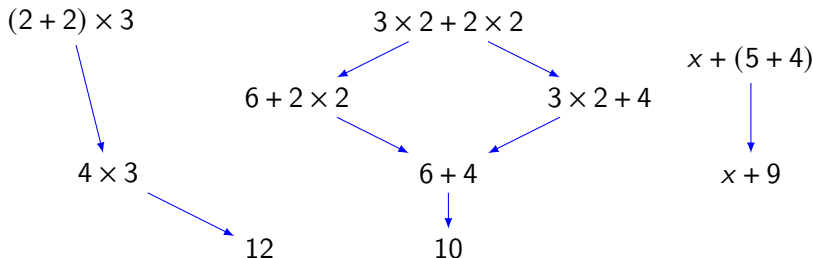
- **Syntax:** formal language for writing programs;
- **Operational semantics:** how do programs *execute*.



$$(2 + 2) \times 3 \xrightarrow{\text{1 execution step}} 4 \times 3 \xrightarrow{\text{1 execution step}} 12$$

What is a programming language?

A graph whose vertices are programs.



Variables = placeholders for expressions

- Substitution: $(x + (5 + 4))[x := 12] = 12 + (5 + 4)$
- Reductions are stable under substitution

$$\frac{x + (5 + 4) \rightarrow x + 9}{12 + (5 + 4) \rightarrow 12 + 9}.$$

\leadsto Transition monads!

Ingredients

- Programming languages (PLs) as graphs
 - (**Syntax**) vertices = terms
 - (**Semantics**) arrows = reductions between terms
- Simultaneous substitution: variables \mapsto terms
 - monads and modules over them

Example

λ -calculus with β -reduction:

- **Syntax:** $S, T ::= x \mid S T \mid \lambda x. S$
- Modulo α -**equivalence**, e.g.

$$\lambda x. x = \lambda y. y$$

- **Reductions:** $(\lambda x. t) u \xrightarrow{\beta} t[x := u]$ + congruences

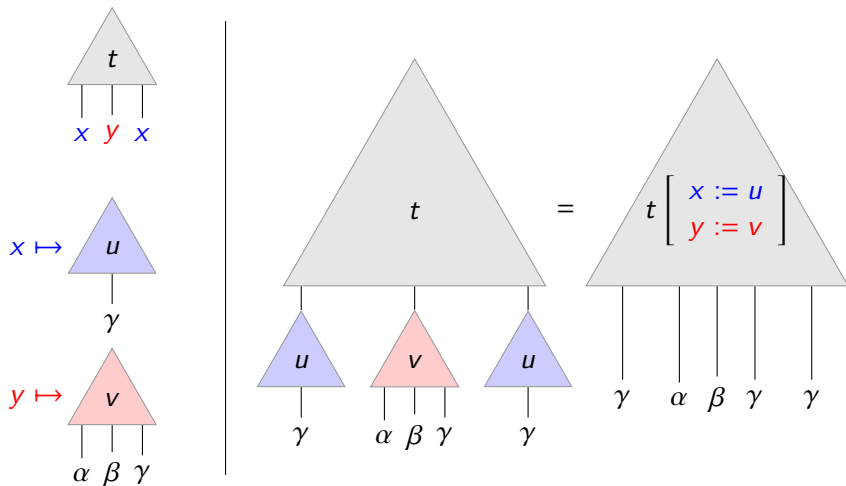
Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

Simultaneous substitution

Syntax comes with substitution

terms as syntax trees (free variables as distinguished leaves).



Simultaneous substitution made formal

Free variables indexing

$$L(X) = \{\text{terms taking free variables in } X\}$$

Example: λ -calculus

$$L(\{x, y\}) = \left\{ \begin{array}{c} \triangle \\ \lambda z. z \end{array} , \begin{array}{c} \triangle \\ x \\ | \\ x \end{array} , \begin{array}{c} \triangle \\ y \\ | \\ y \end{array} , \begin{array}{c} \triangle \\ x \ y \\ | \quad | \\ x \quad y \end{array} , \dots \right\}$$

Simultaneous substitution (bind)

$$\forall f : X \rightarrow L(Y),$$

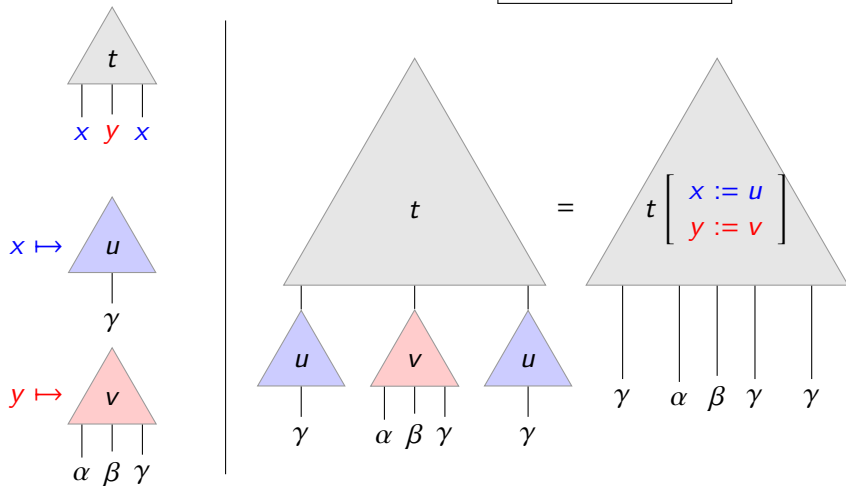
$$\begin{array}{l} L(X) \rightarrow L(Y) \\ t \mapsto t[x \mapsto f(x)] \quad (\text{or } t[f]) \end{array}$$

Simultaneous substitution

$$\forall f : X \rightarrow L(Y),$$

$$X = \{x, y\} \quad Y = \{\alpha, \beta, \gamma\}$$

$$\boxed{\begin{array}{l} L(X) \rightarrow L(Y) \\ t \mapsto t[f] \end{array}}$$

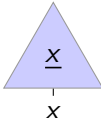


Monads model simultaneous substitution

λ -calculus as a monad $(L, _[_], \eta)$

① Simultaneous substitution $(L, _[_])$

② Variables are terms

$$\eta_X : X \rightarrow L(X)$$


$$x \mapsto \begin{array}{c} \triangle \\ \underline{x} \\ | \\ x \end{array}$$

③ Substitution laws:

$$\underline{x}[f] = f(x) \qquad t[x \mapsto \underline{x}] = t$$

+ associativity:

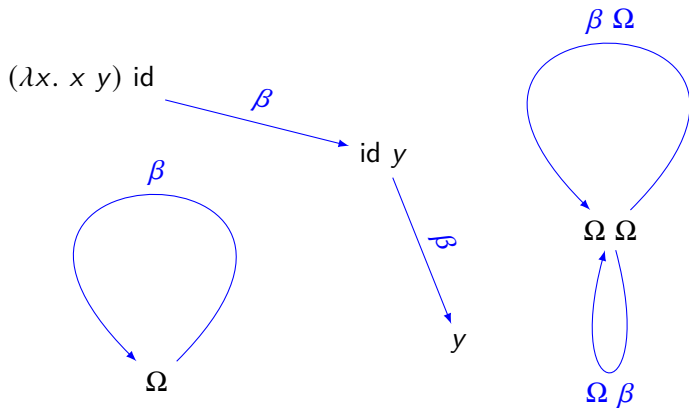
$$t[f][g] = t[x \mapsto f(x)[g]]$$

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

PLs as graphs

Example: λ -calculus with β -reduction



- **(Syntax)** vertices = terms e.g. $\Omega = (\lambda x. x x) (\lambda x. x x)$
- **(Semantics)** arrows = reductions

Graphs

Definition

Graph = a quadruple (A, V, σ, τ) where

$$A = \{\text{arrows}\}$$

$$V = \{\text{vertices}\}$$

$$A \begin{array}{c} \xrightarrow{\text{source}} \\ \xrightarrow{\text{target}} \end{array} V$$

$$\sigma : \begin{array}{c} A \\ t \xrightarrow{r} u \end{array} \rightarrow V \qquad \tau : \begin{array}{c} A \\ t \xrightarrow{r} u \end{array} \rightarrow V$$

$$t \xrightarrow{r} u \mapsto t \qquad t \xrightarrow{r} u \mapsto u$$

$$\sigma(r) \xrightarrow{r} \tau(r)$$

Substitution for semantics

Syntax supports substitution. This is also true of semantics.

Our notion of PL:

- **Syntax:** a monad $(L, _[_], \eta)$
- **Semantics:**

- graphs $R(X) \overset{source_X}{\underset{target_X}{\rightrightarrows}} L(X)$ for each X

$R(X) =$ total set of reductions between terms taking free variables in X

- substitution of reduction: variables \mapsto **L -terms**.

$$\frac{t \xrightarrow{r} u}{t[f] \xrightarrow{r[f]} u[f]} \quad f : X \rightarrow L(Y)$$

$\Rightarrow R$ is a L -module, and *source*, *target* are module morphisms (see next slide)

Substitution for semantics made formal

R as a **module** over L

R supports L -monadic substitution:

$$\forall f : X \rightarrow \mathbf{L}(Y),$$

$$\begin{array}{l} R(X) \rightarrow R(Y) \\ r \mapsto r[x \mapsto f(x)] \quad (\text{or } r[f]) \end{array}$$

+ substitution laws

Other examples of L -modules: L , $L \times L$, 1 , \dots

source and *target* as L -module morphisms

if $\text{source}(r) \xrightarrow{r} \text{target}(r)$ then $\text{source}(r[f]) \xrightarrow{r[f]} \text{target}(r[f])$.

We want $\text{source}(r)[f] \xrightarrow{r[f]} \text{target}(r)[f]$, i.e.,

$$\boxed{\text{source}(r)[f] = \text{source}(r[f])}$$

$$\boxed{\text{target}(r)[f] = \text{target}(r[f])}$$

Commutation with substitution \Leftrightarrow Module morphisms $R \begin{array}{c} \xrightarrow{\text{source}} \\ \xRightarrow{\text{target}} \end{array} L$.

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

Transition monads (first attempt)

Summary: graphs + substitution.

Definition

A **transition monad** $R \begin{smallmatrix} \xrightarrow{\text{source}} \\ \xRightarrow{\phantom{\text{source}}} \\ \xrightarrow{\text{target}} \end{smallmatrix} T$ consists of

- T = monad (= module over itself)
- R = module over T
- $\text{source}, \text{target} : R \rightarrow T$ are T -module morphisms.

Example

λ -calculus with β -reduction.

What about cbv λ -calculus?

- Reductions are stable under substitution with values, not with terms!

Transition monads

cbv λ -calculus (big-step)	Values (monad)	<p style="text-align: center;"> $\begin{array}{ccc} & \text{Transitions} & \\ \swarrow \text{source} & & \searrow \text{target} \\ \text{Terms} & & \text{Values} \end{array}$ </p>
transition monads	a monad T	<p style="text-align: center;"> T-module morphisms $M_1 \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} M_2$ (bipartite graph) </p>
reduction monads ¹	a monad T	<p style="text-align: center;"> $T \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} T$ </p>

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

¹[Ahrens-Hirschowitz-Lafont-Maggesi '20]

Morphisms of transition monads

Simple case $M_i = T$

PLs	\Leftrightarrow	Transition monads
Compilations	\Leftrightarrow	Morphisms of transition monads

Morphism $(T \leftarrow Trans \rightarrow T) \longrightarrow (T' \leftarrow Trans' \rightarrow T') =$

(Syntax) A *monad morphism*¹ $T \xrightarrow{c} T'$

(Semantics) *Forward simulation*²: if $t_1 \xrightarrow{r} t_2$, then $c(t_1) \xrightarrow{\llbracket r \rrbracket} c(t_2)$

Examples (detailed later)

- λ -calculus + fixpoint op. $\longrightarrow \lambda$ -calculus
- λ -calculus + explicit substitution $t[x/u] \longrightarrow \lambda$ -calculus

¹mapping preserving substitution and variables

²backward simulations are often considered as a correctness criteria

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

Constructing *transition monads*

programming language = **transition monad**.

Can we construct them from *simple specifications*?

Overview

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

Specification through **initial semantics**

- Notion of signature

Example (syntax)

A list of operation symbols with associated arities

- To each signature is associated
 - a notion of model

Example

a **monad** equipped with the operations of the signature

- a notion of morphism of models

Example

a **monad morphism** preserving operations

- a proof that the category of models has an initial object
- object specified by the signature $\stackrel{def}{=} \text{initial model}$
- Initiality \Rightarrow **recursion principle**.

Three-level specification

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

Three spec steps:

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

\Rightarrow signature for transition monads = signature for each component

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

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

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

- 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)	$\text{Var} \rightarrow T$	

Signature for T

2 operations (application/abstraction)

- Monads always have variables: no need to specify them
- “operation” = *module morphism*, i.e., 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,
[Ahrens-Hirschowitz-Lafont-Maggesi. ’19]

Disgression on T'

- $M' =$ **derivative** of a module M :

X extended with a fresh variable x

$$M'(X) = M(\overbrace{X \amalg \{x\}})$$

used to model an operation binding a variable.

$$\text{abs} : L' \rightarrow L \quad \left\{ \begin{array}{l} \text{abs}_X : L(X \amalg \{x\}) \rightarrow L(X) \\ t \mapsto \lambda x. t \end{array} \right.$$

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 interpretation 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”: $\text{app}(t_1, u) \rightarrow \text{app}(t_2, u)$	$V \rightarrow M_1 \times M_2$ $(t_1, t_2, u) \mapsto (\text{app}(t_1, u), \text{app}(t_2, u))$

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

Components of the rule	Interpreted as...
2 “metavariables”: t_1, t_2	a “metavariable” T -module $V = T' \times T'$
1 “premise”: $t_1 \rightarrow t_2$	$V \rightarrow T' \times T'$ (T -module morphism) $(t_1, t_2) \mapsto (t_1, t_2)$
“conclusion”: $\lambda x. t_1 \rightarrow \lambda x. t_2$	$V \rightarrow T \times T$ $(t_1, t_2) \mapsto (\lambda x. t_1, \lambda x. t_2)$

Example 5/7: Specify M_i for cbv

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

big-step cbv λ -calculus = $(\text{Vals}, \text{Tms} \xleftarrow{\text{source}} \text{Trans} \xrightarrow{\text{target}} \text{Vals})$

Syntax of values and terms

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

$\text{Tms} : t, u ::= \underbrace{x | \lambda x. t}_v | t u \quad \Rightarrow \quad \begin{array}{l} \text{terms} = \text{binary trees of values} \\ \text{Tms} = \text{BinTree} \circ \text{Vals} \end{array}$

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

variables (= labelled leaves) + 1 binary operation (building nodes)

Example 6/7: Specify 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 $t_1 + \dots + t_n$

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

multiterms = **formal sum** of terms

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

Signature for *FormalSum*

Operations	a constant 0, a binary operation $+$, variables
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

$$app : (t, U) \mapsto t \ U$$

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

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

Signature for T

3 operations (no equation):

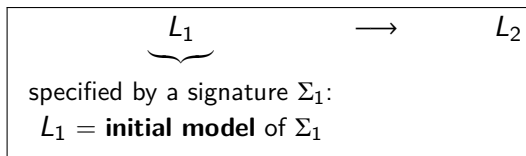
application $t \ U$	$T \times (\text{FormalSum} \circ T) \rightarrow T$
differential application $Dt \cdot u$	$T \times T \rightarrow T$
λ -abstraction	(as before)

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

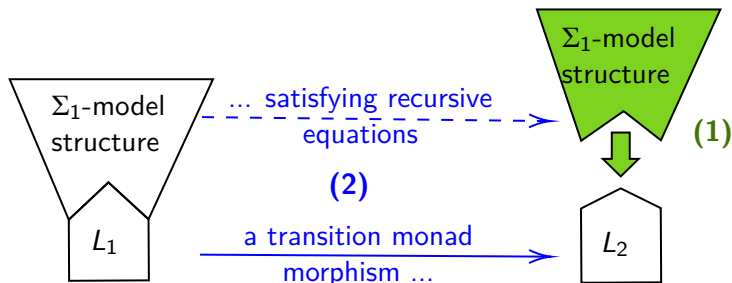
Generating compilations by initiality

Initiality \approx recursion principle



Data generating a compilation: a Σ_1 -model structure for L_2

\Rightarrow By recursion/**initiality**, get a model morphism $L_1 \rightarrow L_2$



Examples

$$\underbrace{L_1} \longrightarrow L_2$$

specified by a signature Σ_1 :

Recipe:

- ① provide a Σ_1 -model structure for L_2
- ② as a model morphism, the induced compilation satisfies recursive equations.

Upcoming examples

- λ -calculus + formal fixpoint op. $\longrightarrow \lambda$ -calculus
 - ① construct a fixpoint operator in λ -calculus
 - ② formal fixpoint operator \mapsto constructed fixpoint operator
- λ -calculus + explicit substitution¹ $t[x/u] \longrightarrow \lambda$ -calculus
 - ① consider λ -calculus with its unary substitution operation
 - ② explicit substitution \mapsto real substitution

¹"A Theory of Explicit Substitutions with Safe and Full Composition",
Kesner '09

Example 1/2: compiling λ -calculus + formal fixpoint op.

$$\underbrace{L_{\text{fix}}}_{\text{specified by } \Sigma_L + \Sigma_{\text{fix}}} \longrightarrow \underbrace{L}_{\text{specified by } \Sigma_L} \quad (\lambda\text{-calculus})$$

Signature Σ_{fix} specifying a fixpoint operator

- an operation $T' \xrightarrow{\text{fix}} T$
- reductions $\text{fix}(t) \rightarrow t[\underbrace{x}_{\text{the fresh variable}} := \text{fix}(t)]$

Model structure on L for Σ_{fix} (\Rightarrow compilation $L_{\text{fix}} \rightarrow L$)

- choose a fixpoint combinator: a term Y s.t. $Y u \rightarrow_{\beta}^* u (Y u)$
- define $\text{fix}(t) := Y(\lambda x.t)$

$$\underbrace{Y(\lambda x.t)}_{\text{fix}(t)} \rightarrow_{\beta}^* (\lambda x.t)(Y(\lambda x.t)) \rightarrow_{\beta} \underbrace{t[x := Y(\lambda x.t)]}_{t[x := \text{fix}(t)]}$$

Example 2/2: compiling λ -calculus + explicit substitution

$$\underbrace{L_{\text{ex}}}_{\text{specified by } \Sigma_L \setminus \{\beta\} + \Sigma_{\text{ex}}} \longrightarrow \underbrace{L}_{\text{specified by } \Sigma_L} \quad (\lambda\text{-calculus})$$

Signature Σ_{ex} for the explicit substitution

- an operation $T' \times T \xrightarrow{(t,u) \mapsto t[x/u]} T$ s.t.

$$\boxed{t[x/u][y/v] = t[y/v][x/u]} \quad \text{if } x \notin \text{fv}(v), y \notin \text{fv}(u)$$

- β -reduction $\boxed{(\lambda x.t)u \rightarrow t[x/u]}$ + congruences +

$$t[x/u][y/v] \rightarrow t[y/v][x/u[y/v]] \quad x \notin \text{fv}(v), y \in \text{fv}(u) \quad (1)$$

Model structure on L for Σ_{ex} (\Rightarrow compilation $L_{\text{ex}} \rightarrow L$)

- use the real substitution $T' \times T \xrightarrow{(t,u) \mapsto t[x:=u]} T$
- β -reduction + congruences + reflexive reduction (1)

Outline

- 1 Transition monads
 - Substitution
 - Graphs
 - Definition
- 2 Generating transition monads
 - Three-level specification
 - Examples
- 3 Generating compilations by initiality
- 4 Conclusion

Conclusion

Summary

- PLs as transition monads
- Compilation as transition monad morphisms
- Signatures for transition monads

Perspectives

- Develop the metatheory (e.g., congruence of bisimilarity)
- Extensions to linear types, subtyping, polymorphism, ...
- π -calculus as a transition system?
- signature for computational λ -calculus?

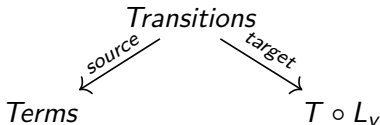
Example: computational λ -calculus¹

Parameterized by:

- a set Σ of operation symbols σ with specified arities
- a monad T with operations $T \times \cdots \times T \xrightarrow{\sigma^T} T$.

$$\begin{aligned} M, N &::= \text{return } V \mid VW \mid M \text{ to } x.N \mid \sigma(M, \dots, M); \\ V, W &::= x \mid \lambda x.M. \end{aligned}$$

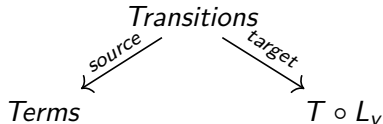
\Rightarrow a monad L_V of **values** + a L_V -module of terms



¹*Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method*,
Dal Lago-Gavazzo-Levy LICS 2017

Example: computational λ -calculus¹

Semantics



$$\frac{}{\text{return } V \Downarrow \eta(V)} \text{ (ret)}$$

$$\frac{M \Downarrow X \quad N[x := V] \Downarrow Y_V}{M \text{ to } x.N \Downarrow X \gg (V \mapsto Y_V)} \text{ (seq)}$$

$$\frac{M[x := V] \Downarrow X}{(\lambda x.M)V \Downarrow X} \text{ (app)}$$

$$\frac{M_1 \Downarrow X_1 \quad \dots \quad M_k \Downarrow X_k}{\sigma(M_1, \dots, M_k) \Downarrow \sigma^T(X_1, \dots, X_k)} \text{ (op)}$$

¹*Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method*,
Lago-Gavazzo-Levy LICS 2017