Mathematical specification of programming languages using monads and modules over them

Ambroise Lafont[1]

[1]University of New South Wales
Sydney, Australia

February 22, 2021

That is the question

- What is a programming language, mathematically?
- How do we specify them?

Differential λ-calculus [Ehrhard-Regnier '03]

$\sim$10 pages (section 2 $\longrightarrow$ 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)
- $\overline{\lambda}\mu$-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

- 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

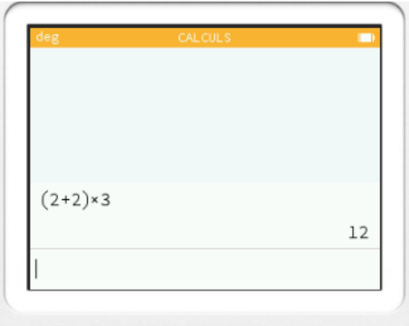
Outline

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.

$$\begin{tikzcd}
(2+2)\times 3 \arrow[dd] & & 3\times 2+2\times 2 \arrow[dl] \arrow[dr] & & x+(5+4) \arrow[dd] \\
& 6+2\times 2 \arrow[dr] & & 3\times 2+4 \arrow[dl] & \\
4\times 3 \arrow[dr] & & 6+4 \arrow[d] & & x+9 \\
& 12 & 10 & &
\end{tikzcd}$$

Variables = placeholders for expressions

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

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

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

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

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

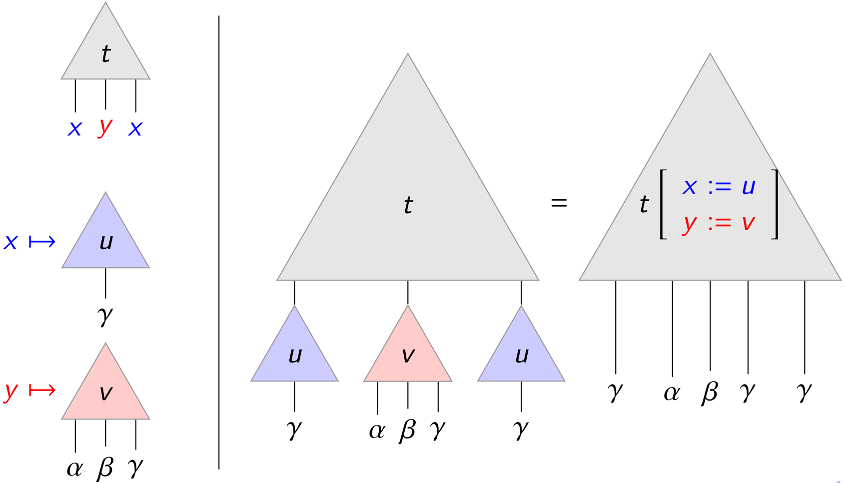
Outline

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).



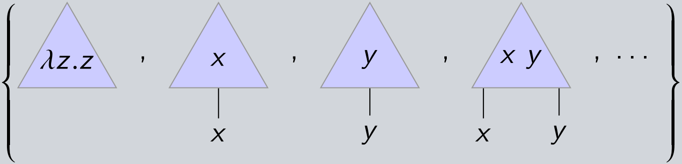
Simultaneous substitution made formal

Free variables indexing

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

Example: λ-calculus

$L(\{x, y\}) =$



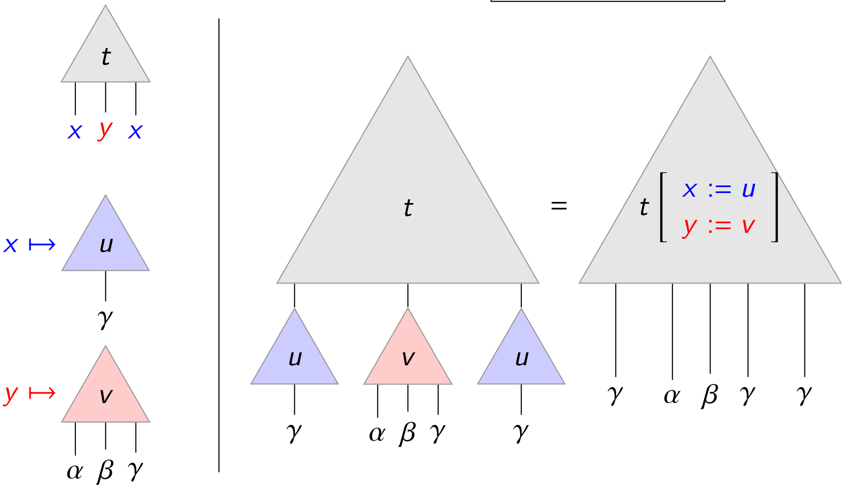
Simultaneous substitution (bind)

$$\forall f : X \to L(Y), \quad \boxed{\begin{array}{rl} L(X) & \to L(Y) \\ t & \mapsto t[x \mapsto f(x)] \quad (\text{or } t[f]) \end{array}}$$

Simultaneous substitution

$$\forall f : X \to L(Y),$$
$$X = \{x, y\} \qquad Y = \{\alpha, \beta, \gamma\}$$

$$\boxed{\begin{array}{rl} L(X) & \to L(Y) \\ t & \mapsto t[f] \end{array}}$$



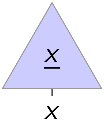
Monads model simultaneous substitution

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

1. Simultaneous substitution $(L, \_[\_])$
2. Variables are terms

$$\eta_X : \quad X \quad \to \quad L(X)$$

$$x \quad \mapsto$$



3. Substitution laws:

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

$+$ associativity:

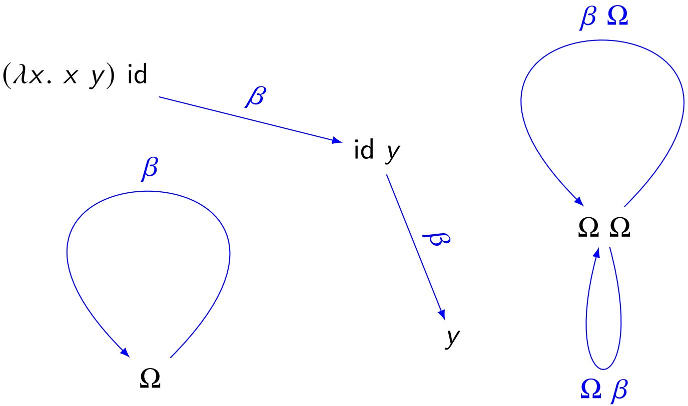
$$t[f][g] = t\,[x \mapsto f(x)[g]]$$

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

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

$$\begin{tikzcd} A \arrow[r, shift left, "source"] \arrow[r, shift right, "target"'] & V \end{tikzcd}$$

$$\sigma : \begin{array}{ccc} A & \to & V \\ t \xrightarrow{r} u & \mapsto & t \end{array} \qquad \tau : \begin{array}{ccc} A & \to & V \\ t \xrightarrow{r} u & \mapsto & u \end{array}$$

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

Substitution for semantics

Syntax supports substitution. This is also true of semantics.

Our notion of PL:

- **Syntax**: a monad $(L, \_[\_], \eta)$
- **Semantics**:
  - graphs $R(X) \underset{target_X}{\overset{source_X}{\rightrightarrows}} L(X)$ for each X

    $R(X) =$ total set of reductions between terms taking free variables in X
  - substitution of reduction: variables $\mapsto$ L-**terms**.

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

$\Rightarrow$ 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:

$$\forall f : X \to \boldsymbol{L}(Y), \qquad \boxed{\begin{array}{rl} R(X) & \to R(Y) \\ r & \mapsto r[x \mapsto f(x)] \quad (\text{or } r[f]) \end{array}}$$

$+$ substitution laws

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

source and *target* as L-**module morphisms**

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

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

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

Commutation with substitution $\Leftrightarrow$ Module morphisms $R \underset{target}{\overset{source}{\rightrightarrows}} L$.

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** $R \underset{target}{\overset{source}{\rightrightarrows}} T$ consists of

- T = monad (= module over itself)
- R = module over T
- $source, target : R \to T$ 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)	$\begin{tikzcd} & \mathit{Transitions} \arrow[dl, "source"'] \arrow[dr, "target"] & \\ \mathit{Terms} & & \mathit{Values} \end{tikzcd}$
transition monads	a monad T	T-module morphisms $M_1 \xleftarrow{source} \mathit{Trans} \xrightarrow{target} M_2$ (bipartite graph)
reduction monads[1]	a monad T	$T \xleftarrow{source} \mathit{Trans} \xrightarrow{target} T$

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

[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* $c(t_1) \xrightarrow{[\![r]\!]} c(t_2)$

Examples (detailed later)

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

[1] mapping preserving substitution and variables

[2] backward 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 $\stackrel{def}{=}$ initial model
- Initiality $\Rightarrow$ recursion principle.

Three-level specification

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

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 $\dfrac{t_1 \to u_1 \dots t_n \to u_n}{t \to u}$

$\Rightarrow$ 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

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

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

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	*Trans*, *source*, *target*	"transition structure"	β-rule + congruences

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

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

- Syntax "generated" by

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

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:

$$M'(X) = M(\overbrace{X \amalg \{x\}}^{X \text{ extended with a fresh variable } x})$$

used to model an operation binding a variable.

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

Fun facts

$$M' \cong M^L \text{ in the category of } L\text{-modules}$$

For L = monad of λ-calculus modulo β- and η-equation,

$$L^L \cong L \text{ in the category of } L\text{-modules}$$

Example 3/7: Left congruence for application

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

Left congruence rule for application

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

- 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": $t_1 \longrightarrow t_2$	$M_1 \leftarrow V \rightarrow M_2$ $t_1 \leftarrow\!\shortmid (t_1, t_2, u) \mapsto t_2$
"conclusion": $app(t_1, u) \longrightarrow app(t_2, u)$	$M_1 \leftarrow V \rightarrow M_2$ $app(t_1, u) \leftarrow\!\shortmid (t_1, t_2, u) \mapsto app(t_2, u)$

Example 4/7: Binding variables in rules

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

Congruence rule for abstraction

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

- "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": $t_1 \to t_2$	$M'_1 \leftarrow V \to M'_2$ $t_1 \leftarrow\!\shortmid (t_1,t_2) \mapsto t_2$
"conclusion": $\lambda x.t_1 \to \lambda x.t_2$	$M_1 \leftarrow V \to M_2$ $\lambda x.t_1 \leftarrow\!\shortmid (t_1,t_2) \mapsto \lambda x.t_2$

Example 5/7: Specify M_i for cbv

$$\text{Transition monad} = (T, \quad M_1 \xleftarrow{\text{source}} \mathit{Trans} \xrightarrow{\text{target}} M_2)$$
$$\text{big-step cbv } \lambda\text{-calculus} = (\mathit{Vals}, \mathit{Tms} \xleftarrow{\text{source}} \mathit{Trans} \xrightarrow{\text{target}} \mathit{Vals})$$

Syntax of values and terms

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

In fact, by definition of a transition monad,

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

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

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

Differential λ-calculus (DLC)

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

Semantics a $\underbrace{\text{term } t}_{M_1 = Id \circ T\ (=T)}$ reduces to a $\underbrace{\text{multiterm } t_1 + \cdots + t_n}$

multiterms = formal sum of terms

$M_2 = FormalSum \circ T$

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: $(T, M_1 \xleftarrow{source} Trans \xrightarrow{target} M_2)$

- 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 $U = u_1 + \cdots + u_n$
= a term and a formal sum *of* terms
input *module* of *app* = $T \times (FormalSum \circ T)$

Signature for T

3 operations (no equation):

application $t\,U$	$T \times (FormalSum \circ T) \to T$
differential application $Dt \cdot u$	$T \times T \to T$
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 ≈ recursion principle

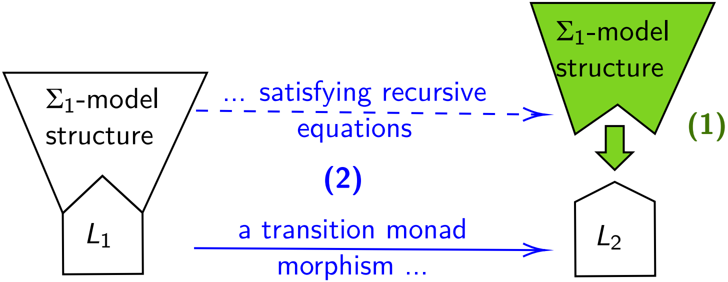
$$\underbrace{L_1}_{} \longrightarrow L_2$$

specified by a signature Σ_1:

L_1 = **initial model** of Σ_1

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

⇒ By recursion/**initiality**, get a model morphism $L_1 \to L_2$



Examples

$$\underbrace{L_1}_{\text{specified by a signature } \Sigma_1:} \longrightarrow L_2$$

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. $\longrightarrow$ λ-calculus
  1. construct a fixpoint operator in λ-calculus
  2. formal fixpoint operator $\mapsto$ constructed fixpoint operator
- λ-calculus + explicit substitution[1] $t[x/u]$ $\longrightarrow$ λ-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.

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

Signature Σ_{fix} specifying a fixpoint operator

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

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

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

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

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

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

Signature Σ_{ex} for the explicit substitution

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

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

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

$$t[x/u][y/v] \to t[y/v][x/u[y/v]] \quad x \notin fv(v),\ y \in fv(u) \quad (1)$$

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

- 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, ...
- Signature for computational λ-calculus?
- Strengthen the recursion principle (i.e., enlarge the category of models)

Future work: strengthen the recursion principle

Initial semantics (general framework)

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

- general models — easier signatures

transition monads

reduction monads (Ahrens et al. '20)

+ general models — stronger 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

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

component	Σ_1	Σ_2	Σ_3 **(to be generalised)**
specifies	T	M_1, M_2	$\leftarrow \mathit{Trans} \rightarrow$

Future work

Σ_3 specifies transition rules for

T = ~~'the' initial~~ *any* model of Σ_1 (as in Ahrens et al. '20)

(M_1, M_2) = ~~'the' initial~~ *any* 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!

Example: computational λ-calculus[1]

Parameterized by:

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

$$
\begin{array}{lcl}
M, N & ::= & \mathsf{return}\ V \mid VW \mid M\ \mathsf{to}\ x.N \mid \sigma(M, \ldots, M); \\
V, W & ::= & x \mid \lambda x.M.
\end{array}
$$

$\Rightarrow$ a monad L_v of **values** + a L_v-module of terms

$$
\begin{tikzcd}
 & \mathit{Transitions} \arrow[dl, "\mathit{source}"'] \arrow[dr, "\mathit{target}"] & \\
\mathit{Terms} & & T \circ L_v
\end{tikzcd}
$$

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

Example: computational λ-calculus[1]

Semantics

$$
\begin{tikzcd}
 & \mathit{Transitions} \arrow[dl, "source"'] \arrow[dr, "target"] & \\
\mathit{Terms} & & T \circ L_v
\end{tikzcd}
$$

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

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

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

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

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