

Towards a theory of programming languages

Tom Hirschowitz



LAMA, CNRS et Université de Savoie

- 1 The question
- 2 Lawvere theories
- 3 Generalising Lawvere theories
- 4 Perspectives

The ladder of abstraction (© Brett Victor): level 1

Formal reasoning over practical things like

counting: integers,

moving: transformations of the plane,

exchanging: permutations.

The ladder of abstraction: level 2

Common properties of:

- integers,
- rotations,
- translations,
- symmetries,
- isometric transformations.
- permutations on a finite set...

Abstraction

Notion of **group**.

The ladder of abstraction: towards level 3

Neighbours

- Generalisation: monoids (e.g., all transformations of the plane).
- Specialisation: rings, modules, algebras, ...

Problem

- Numerous **abstract** notions.
- A lot in common:
 - free constructions (e.g., free monoid = words);
 - notions of morphisms;
 - downcasting (any ring is a group, at least).

The ladder of abstraction: level 3

Abstraction over abstraction!

Two proposals:

- Lawvere theories;
- monads on sets.

Lawvere theories a little more general.

Programming language theory: still at level 2!

Level 1: calculi

Programming languages \rightsquigarrow **calculi**:

- pure λ -calculus,
- λ -calculus in cbv, cbn, lazy, fully lazy, optimal, ...
- λ -calculus with `let rec` / `refs` / `call/cc`, ...
- ζ , $\lambda\sigma$, $\bar{\lambda}\bar{\mu}\bar{\mu}$, ...
- π , `join`, `Ambients`, `Spy`, ...

And that's only the **untyped** tip of the iceberg!

Level 2: denotational semantics

Groups: integers, permutations, ...

Rings: integers, polynomials, ...

X-calculus: models of X.

Not all calculi are at level 2 (i.e., equipped with a notion of model).

Towards level 3?

Groups: integers, permutations, ...)	Lawvere theories
Rings: integers, polynomials, ...		
X-calculus: models of X,		Lawvere theory?

No, for at least two reasons:

- variable binding ($\lambda x. x \equiv \lambda x'. x'$);
- dynamics ($(\lambda x. M)N \rightsquigarrow M[x \mapsto N]$).

Need to generalise Lawvere theories!

Hence the question

- What is a programming language?
- What is a translation between two programming languages?
- General results?

Let us start with a (fake) poll...

Fake poll: what is a programming language?

Low-level answer

- a language on a finite alphabet,
 - a translation to x86 (...).
-
- Why definitely fix x86?
 - Low-level does evolve a lot.
 - Sometimes overkill, e.g., language of regular expressions.
 - Not canonical.
 - Limited: e.g., distributed computing.
 - Not won at the level of high-level reasoning on programs.

Fake poll: what is a programming language?

Frequent answer by researchers in programming languages

One super complex language supposed to model all other languages.

- Illusory.
- Adding features may change global properties: study of fragments.
- Eludes the crucial question of what **morphisms** should be.

Fake poll: what is a programming language?

Other frequent answer mostly in the UK

A **structural operational semantics** (Plotkin, 1981), in a certain **format**.

- Low-level notion of syntax with binding.
- Morphisms: virtually not investigated.
- Far from mainstream mathematics.

Fake poll: what is a programming language?

Other frequent answer

A **higher-order rewrite system** (Nipkow, 1991).

Close answer: a **combinatory reduction system** (Klop, 1980).

- Roughly, rewriting terms with variable binding.
- **No** notion of morphism, even google does not find anything.
- Far from mainstream mathematics.

About the last two,

- structural operational semantics (SOS) and
- higher-order rewriting (HOR).

- General results, but on **one** language:
 - congruence of bisimilarity (SOS),
 - confluence, finite developments, etc (HOR).
- Better, hints at level 3:
 - **mathematical operational semantics** (SOS, Turi et Plotkin),
 - **cartesian closed 2-categories** (HOR, main subject here).

Lawvere theories in 20 slides

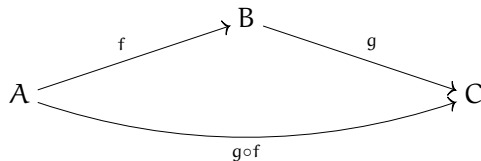
Starters: introduction to **category theory**, then **Lawvere theories**.

Categories

Definition

Category: a (directed, multi) graph equipped with

- a **composition** law on edges,
- **identities**.

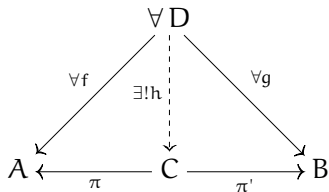


Examples

- The category Grp.
 - Vertices / **objects**: groups.
 - Edges / **morphisms**: morphisms of groups.
- **Large category**.
- Similar examples: monoids, rings, etc.
- Topological spaces and continuous functions: Top.
- Graphs: Gph.
- Even plain **sets**: Set.

Cartesian product (in any category!)

- Consider any objects $A, B \in \mathcal{C}$;
- $A \xleftarrow{\pi} C \xrightarrow{\pi'} B$ is a **product** of A and B iff



- Notation: $C = A \times B$ and $h = \langle f, g \rangle$.

Example

- Set, Grp, Top,...
- Graphs.

Terminal object

- A is a **terminal** object in \mathcal{C} iff $\forall B \dashrightarrow^{\exists! f} A$.
- Notation: $A = 1, f = !B$.

Example

- Sets: singleton.
- Graphs: what would you guess?

Finite products = products + terminal objects

Definition

Category with **finite products**:

- a product $(A \times B, \pi, \pi')$ for all A, B ,
- a terminal object 1 .

First insight of Lawvere theories

Definition

Any model of an algebraic theory 'is' a category with finite products.

I.e.,

- Any monoid is a category with finite products.
- Any ring is a category with finite products.
- ...

First insight of Lawvere theories

Definition

Any model of an algebraic theory 'is' a category with finite products.

Example

Category $\mathcal{C}_{\mathbb{N}}$ for the monoid of natural numbers and $+$:

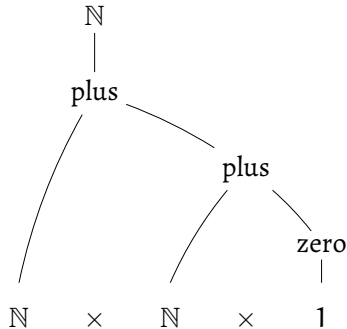
- objects are finite 'powers' of \mathbb{N} , e.g., $\mathbb{N} \times \mathbb{N} \times \mathbb{N}$;
- morphisms are functions generated by
 - addition $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$,
 - zero $1 \rightarrow \mathbb{N}$ (the map picking 0),
 - identities and composition,
 - pairing $\langle f_1, \dots, f_n \rangle : \mathbb{N}^p \rightarrow \mathbb{N}^n$,
 - projections $\pi_{n,i} : \mathbb{N}^n \rightarrow \mathbb{N}$,
 - the unique map $\mathbb{N}^p \rightarrow 1$.

This is a subcategory of Set .

The category $\mathcal{C}_{\mathbb{N}}$ (looking closer)

- Any morphism $f : \mathbb{N}^p \rightarrow \mathbb{N}^n$ decomposes as $\langle f_1, \dots, f_n \rangle$.
- Example morphism $\text{plus} \circ \langle \pi, \text{plus} \circ \langle \pi', (\text{zero} \circ !(N^2)) \rangle \rangle : \mathbb{N}^2 \rightarrow \mathbb{N}$.
- A.k.a. $\text{plus}(x, (\text{plus}(y, \text{zero})))$.

- A.k.a.



\approx circuits with sharing restricted to inputs.

- Variables: dealt with by projections.

Any monoid X 'is' a category \mathcal{C}_X with finite products

Replace X with underlying set X , etc:

Example

Category \mathcal{C}_X for the monoid (X, m, e) :

- objects are finite 'powers' of X , e.g., $X \times X \times X$;
- morphisms are functions generated by
 - $m : X \times X \rightarrow X$,
 - $e : 1 \rightarrow X$ (the map picking e),
- identities and composition,
- pairing $\langle f_1, \dots, f_n \rangle : X^p \rightarrow X^n$,
- projections $\pi_{n,i} : X^n \rightarrow X$,
- the unique map $X^p \rightarrow 1$.

Again a subcategory of Set .

The Lawvere theory for monoids

Definition

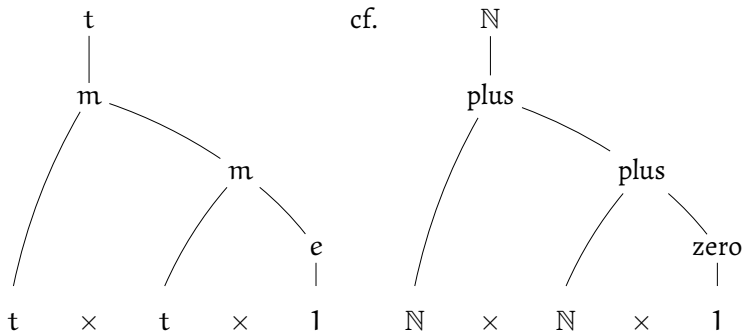
The category $\mathcal{L}_{\text{monoid}}$ defined by:

- Objects: one object, say t , and its **formal** finite powers $t \times \dots \times t$.
- Morphisms $t \times \dots \times t \rightarrow t$: **terms** generated by
 - binary m ,
 - constant e ,
 - in n (ordered) variables,up to a few equations.
- Morphisms $t^n \rightarrow t^p$: p -tuples of terms.
- Composition = simultaneous substitution.

Not directly a subcategory of sets.

The Lawvere theory for monoids

In $\mathcal{L}_{\text{monoid}}$, example morphism



Observation

There seems to be a 'map' $\mathcal{L}_{\text{monoid}} \rightarrow \mathcal{C}_{\mathbb{N}}$

Generalised monoids

Definition

A **generalised monoid** is

- a category \mathcal{C} with finite products,
- an object $X \in \mathcal{C}$,
- morphisms $\text{comp} : X \times X \rightarrow X$ and $\text{unit} : 1 \rightarrow X$,
- satisfying the obvious associativity and unitality equations.

Example

- $\mathcal{C}_{\mathbb{N}}$,
- \mathcal{C}_X , for any monoid X ,
- $\mathcal{L}_{\text{monoid}}$.
- A bigger one: Set with, e.g., \mathbb{N} and addition.

Morphisms

Definition

A **morphism of generalised monoids** is

- a **functor** $F : \mathcal{C} \rightarrow \mathcal{D}$ between underlying categories,

Wait wait, what's a functor?

Functors

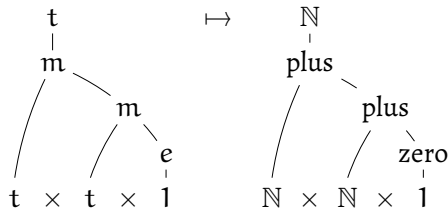
Definition

A **functor** $\mathcal{C} \rightarrow \mathcal{D}$ is a 'morphism of categories':

- a map from objects of \mathcal{C} to objects of \mathcal{D} ,
- a map between morphisms (preserving source and target),
preserving composition and identities.

Example

The 'map' $\mathcal{L}_{\text{monoid}} \rightarrow \mathcal{C}_{\mathbb{N}}$ determined by



Morphisms

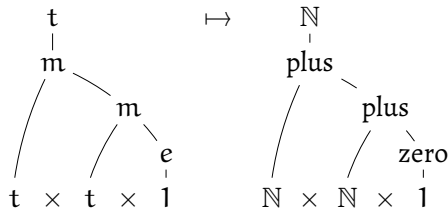
Definition

A **morphism of generalised monoids** is

- a **functor** $F : \mathcal{C} \rightarrow \mathcal{D}$ between underlying categories,
- preserving products: $F(A \times B) = F(A) \times F(B)$ (subtlety here, who can guess?),
- mapping X , comp , and unit in \mathcal{C} to their counterparts in \mathcal{D} .

Example

Again, the functor $\mathcal{L}_{\text{monoid}} \rightarrow \mathcal{C}_{\mathbb{N}}$ determined by



In particular $t \times t \mapsto \mathbb{N} \times \mathbb{N}$.

The Lawvere theory for monoids

Proposition

Generalised monoids

\simeq

categories \mathcal{C} with a finite-product preserving functor $\mathcal{L}_{\text{monoid}} \rightarrow \mathcal{C}$.

Intuitively, $\mathcal{L}_{\text{monoid}}$ serves as a **definition** of monoids.

Further example

What should be the Lawvere theory $\mathcal{L}_{\text{rings}}$ for rings?

- Objects: one object, say t , and its formal finite powers $t \times \dots \times t$.
- Morphisms $t \times \dots \times t \rightarrow t$: terms generated by
 - binary mult and add,
 - constants one and zero,
 - in n (ordered) variables,up to a few equations.
- Morphisms $t^n \rightarrow t^p$: p -tuples of terms.
- Composition = simultaneous substitution.

Lawvere theories: definition

Definition

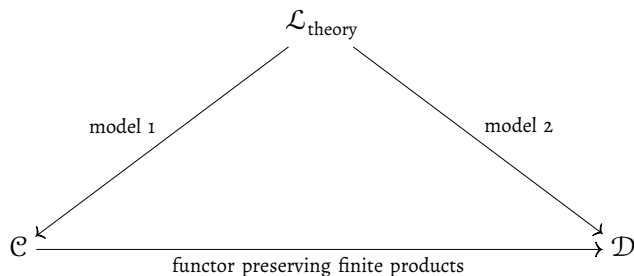
Lawvere theory: a category with

- finite products,
- objects formally generated by a set of `sorts'.

E.g., $t \times u \times t$, for sorts t and u .

What has been gained (quick summary)

- Signature + equations, i.e., theory \mapsto **category of models**



There are more general notions of morphisms...

- A notion of morphism between Lawvere theories: functors preserving finite-products; e.g.,

$$\mathcal{L}_{\text{monoid}} \hookrightarrow \mathcal{L}_{\text{ring}}$$

What is missing?

- Variable binding $(\lambda x. x \equiv \lambda x'. x')$.
- Dynamics $(\lambda x. M)N \rightsquigarrow M[x \mapsto N]$.

Need to generalise Lawvere theories!

Variable binding

We defined products by a [property](#).

- Consider any objects $A, B \in \mathcal{C}$ with finite products;
 $C \times A \xrightarrow{\text{ev}} B$ is an **exponential** of A and B iff $\forall D \in \mathcal{C}$,

$$\begin{array}{ccc}
 C \times A & \xrightarrow{\text{ev}} & B \\
 \uparrow \exists! h \times \text{id}_A & \nearrow \forall f & \\
 D \times A & &
 \end{array}$$

- Notation: $C = B^A$ and $h = \lambda f$.
- Intuition: $B^A =$ function space, $\lambda f =$ currying of f .

Variable binding

We defined products by a [property](#).

- Consider any objects $A, B \in \mathcal{C}$ with finite products;
 $B^A \times A \xrightarrow{\text{ev}} B$ is an **exponential** of A and B iff $\forall D \in \mathcal{C}$,

$$\begin{array}{ccc}
 B^A \times A & \xrightarrow{\text{ev}} & B \\
 \uparrow \exists! \lambda f \times \text{id}_A & & \nearrow \forall f \\
 D \times A & &
 \end{array}$$

- Notation: $C = B^A$ and $h = \lambda f$.
- Intuition: $B^A =$ function space, $\lambda f =$ currying of f .

Examples

- Set: B^A = set of functions $A \rightarrow B$.
- Gph, graphs: some convoluted construction of rare use (to my knowledge).
- Not Top! Have to restrict to compactly generated spaces.
- **Scott domains**. Particular posets, important in denotational semantics.

Cartesian closed categories = products + terminal object + exponentials

Definition

Cartesian closed category (CCC):

- a product $(A \times B, \pi, \pi')$ for all A, B ,
- a terminal object 1 ,
- an exponential (B^A, ev) for all A, B .

Variable binding

Synopsis

Models of theories with binding 'are' cartesian closed categories.

Example

The syntax for the pure λ -calculus yields a cartesian closed category \mathcal{L}_λ :

- objects are **formal** powers and exponentials of t , e.g., $t^t \times t$, $(t \times t \times t)^{t \times t}$, ...
- morphisms are formally generated by
 - $\text{lam} : t^t \rightarrow t$ and $\text{app} : t \times t \rightarrow t$,
 - stuff needed for \mathcal{L}_λ to be a CCC.

- Remark: \mathcal{L}_λ contains more than the usual notion of syntax.
- Morphisms = **simply-typed** λ -terms up to $\beta\eta$ -conversion.
- E.g., $\lambda x. M$ is here modelled as $\text{lam}(\lambda x : t. \llbracket M \rrbracket)$.

Dynamics

To model

$$(\lambda x. M)N \rightsquigarrow M[x \mapsto N]$$

we could add an equation

$$\text{app} \langle \text{lam} (\lambda x : t. M), N \rangle = (\lambda x : t. M)N.$$

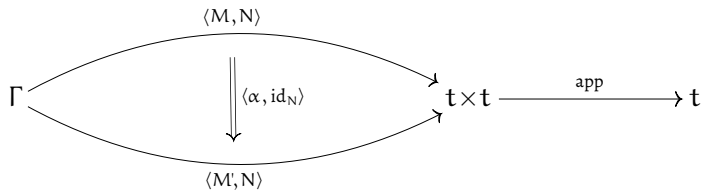
We prefer adding a **2-cell**:

$$\begin{array}{ccc}
 & & t \times t \\
 \text{lam} \times \text{id}_t \nearrow & & \searrow \text{app} \\
 t^t \times t & & t \\
 & \searrow \text{ev} & \\
 & & t
 \end{array}$$

\rightsquigarrow **Cartesian closed 2-categories!**

Example reduction

- In pure λ , if $M \rightsquigarrow M'$ then $MN \rightsquigarrow M'N$.
- Here, assuming $\alpha : M \Rightarrow N$, derive

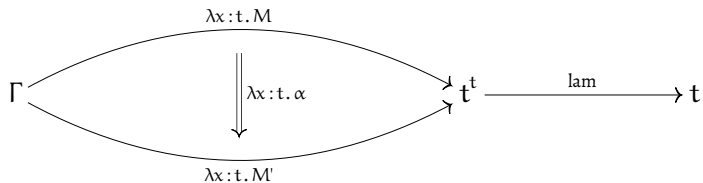
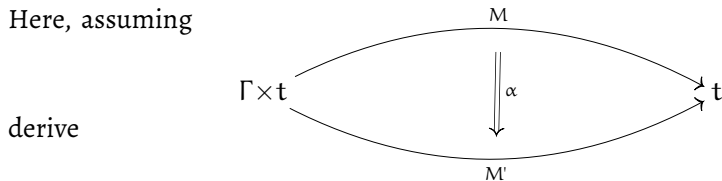


Syntactically

$$\text{app } \langle \alpha ; \text{id}_N \rangle : \text{app } \langle M ; N \rangle \Rightarrow \text{app } \langle M' ; N \rangle.$$

Other example reduction

- In pure λ , if $M \rightsquigarrow M'$ then $\lambda x. M \rightsquigarrow \lambda x. M'$
- Here, assuming



Syntactically

$$\lambda x : t. \alpha : \text{lam}(\lambda x : t. M) \Rightarrow \text{lam}(\lambda x : t. M').$$

Proposition
$$\text{2-cells } M \Rightarrow N$$
$$\cong$$

reductions up to [permutation equivalence](#) (Lévy, late 70's; Bruggink, 2003).

Possible perspectives

- More involved examples.
- Dynamic properties of 2CCCs (following Hilken).
- Dynamic properties of morphisms.
- Formal links with other approaches.
- Extensions, e.g., dependent types.
- Coq library?