# What is a programming language?

Tom Hirschowitz
Based on work with Richard Garner, Florian Hatat, and Aurélien Pardon

Southampton, March 2010

UMR 5127

# Programming languages

Concerned, e.g., with:

- Designing efficient languages, where

  One more quickly writes shorter programs, with less bugs, hopefully not much slower.

  Several possible specialisations (e.g., modularity, concurrency, synchrony).
- Code correctness (e.g., model-checking, behavioural equivalences).
- Compiler correctness.

# Programming languages

- Recent advances: formally certified programs and compilers.
- Each concerning one particular language.

Hope: generalisation.

**Example**

Sufficient conditions for a compiler to preserve a notion of behaviour.

- What is a compiler? a notion of behaviour?
- But wait . . . , what is a programming language?

# Plan

1 Intro

2 First-order theories

3 Higher-order theories

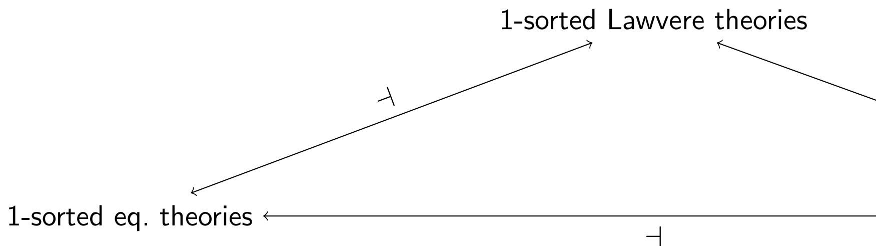
4 2-signatures

5 Examples

6 Topology

# The big picture

The categorical approach to algebra:



# Equational theories

Originally, equational theories are the basis for universal algebra.

## Example

The theory $\mathcal{S}$ for semi-groups has:

- a sort $t$,
- an operation

$$m \colon t \times t \to t,$$

- an equation

$$m(m(x, y), z) = m(x, m(y, z)).$$

# Lawvere theories

A Lawvere theory is

- a category with finite products,
- whose objects are all finite powers of a particular object $t$:

$$1, t, t \times t, \ldots$$

# Equational theories and Lawvere theories

The equational theory $\mathcal{S}$ generates a Lawvere theory $\mathcal{L}\mathcal{S}$:

- Objects: finite ordinals $p, q, \ldots$;
- Morphisms $p \longrightarrow q$: $q$-tuples of terms with variables in $p$, modulo the equations.
- Composition: substitution.
- Notably: morphisms $p \longrightarrow 1 \cong$ terms with variables in $p$.

**Theorem (Lawvere)**

*Models of $\mathcal{S}$ $\simeq$ finite product functors from $\mathcal{L}\mathcal{S}$.*

# Equational theories and monads

The (one-sorted) equational theory $\mathcal{S}$ generates a monad $\mathcal{T}_{\mathcal{S}}$ on Set:

- $\mathcal{T}_{\mathcal{S}}(X)$: terms with variables in $X$, modulo the equations;
- $\mu \colon \mathcal{T}^2_{\mathcal{S}}(X) \to \mathcal{T}_{\mathcal{S}}(X)$: substitution;
- $\eta \colon X \to \mathcal{T}_{\mathcal{S}}(X)$: variables.

**Theorem**

*Models of* $\mathcal{S}$ $\simeq$ $\mathcal{T}_{\mathcal{S}}$*-algebras.*

# Lawvere theories and finitary monads

Restricting to the one-sorted case for simplicity:

**Theorem (Lawvere, Linton)**

*Finitary monads on* Set $\simeq$ *Lawvere theories.*

Lawvere theories $\to$ monads:

$$\mathcal{T}_{\mathcal{L}}(X) = \int^{p \in \omega} X^p \times \mathcal{L}(p, 1),$$

i.e., a morphism $p \to 1$ with variables $x_1, \ldots, x_p \in X$ $(\ldots)$.

# Lawvere theories and finitary monads

Restricting to the one-sorted case for simplicity:

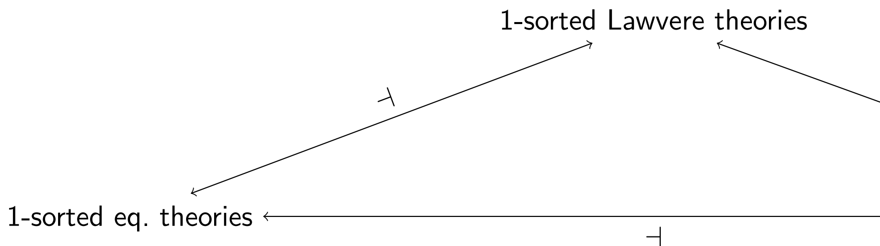
**Theorem (Lawvere, Linton)**

*Finitary monads on* Set $\simeq$ *Lawvere theories.*

Monads $\rightarrow$ Lawvere theories: $\mathcal{L}_{\mathcal{T}}$ determined by

$$\mathcal{L}_{\mathcal{T}}(p, 1) = \mathcal{T}(p).$$

# The big picture



# How about binding?

**Question**

Can monads model binding (and in which sense)?

# Can monads model binding?

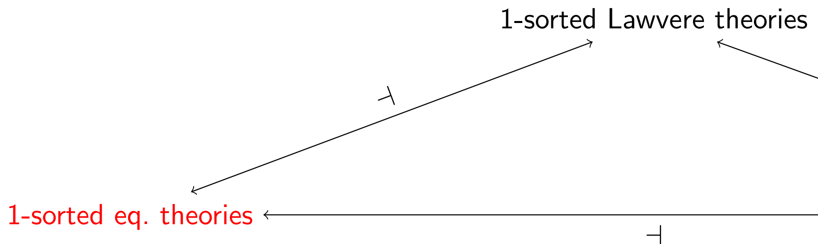
Yes!

- Example: the pure $\lambda$-calculus is a monad on Set with

  $\Lambda(X)$: $\lambda$-terms with free variables in $X$ (modulo $\beta$).
- It is finitary, but not generated by any finite equational signature.
- Reason: binding is not expressible in the language of equational signatures.

# Can monads model binding?

- So, monads do model binding.
- But outside the reach of equational theories:




- How to specify languages with binding?

# A solution

Fiore et al., Hirschowitz (father) et al. propose to extend theories:

- Allow operations to have binding arities, e.g.:

$$\lambda\colon 1 \to 0 \qquad\qquad @\colon 0 \times 0 \to 0.$$

- Replace Lawvere theories with a more complicated framework (Fiore et al.).
- Deliberate restriction to 2nd order theories:

**Reason (Fiore)**

Will to explain the $\lambda$-calculus by something strictly weaker.

# Another solution

- Accept general higher-order theories (in a minute).
- Categorically: finite products + exponentials.
- Cost (?): the syntactic category is much larger than just terms.
  Indeed it includes a form of higher-order contexts.
- Benefit: simple categorical picture.

**Slogan**

One binder to explain them all.

# Higher-order signatures

A higher-order signature $X$ is given by:

- a set $X_0$ of sorts;
- for all formulas $A, B \in \mathcal{F}(X_0)$, a set $X_1(A, B)$ of operations,

where formulas are defined by the grammar:

$$A, B, \ldots \in \mathcal{F}(X_0) ::= x \mid 1 \mid A \times B \mid B^A \qquad x \in X_0.$$

# A ho signature

**Example**

- One sort $t$.
- Two operations

$$L\colon t^t \to t \qquad A\colon t \times t \to t.$$

Call this signature $\Lambda$.

# Return of the 7th son of the $\lambda$-calculus

- Consider a ho signature $X$.
- Think of the (simply-typed) $\lambda$-terms $\mathcal{L}_1(X)$ with:
  - base types in $X_0$,
  - constants $x\colon A \vdash c(x)\colon B$ for all $c \in X_1(A, B)$,
  - modulo $\alpha$-renaming but not $\beta\eta$.

# Higher-order theories

## Definition

A higher-order theory $X$ consists of:

- a higher-order signature $(X_0, X_1)$,
- a set $X_2$ of equations, i.e., parallel terms

$$\Gamma \vdash M, N \colon A$$

in $\mathcal{L}_1(X_0, X_1)$.

# Generating a CCC

- Consider a ho theory $X = (X_0, X_1, X_2)$.
- Recall the $\lambda$-terms $\mathcal{L}_1(X_0, X_1)$.
- Consider the smallest congruence on $\mathcal{L}_1(X_0, X_1)$ generated by:
  - $\beta\eta$,
  - plus the equations in $X_2$.
- This yields a $\lambda$-calculus, and a CCC $\mathcal{C}(X)$.

**Theorem**

$\mathcal{C}(X)$ *is the free CCC on* $X$.

# Examples: variations on $\lambda$

Recall our signature $\Lambda$: $\quad L\colon t^t \to t \qquad A\colon t \times t \to t.$

**Proposition**

*For the theory $\Lambda$ with no equations:*

$$\mathcal{C}(\Lambda)(t^n, t) \cong \textit{pure } \lambda\textit{-terms in } n \textit{ variables.}$$

**Proposition**

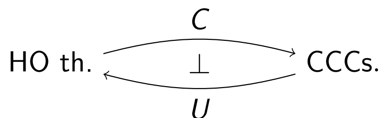
*For the theory $\Lambda'$ with one equation:*

$$A(L(\lambda x.M), N) = (\lambda x.M)N,$$

$\mathcal{C}(\Lambda')(t^n, t) \cong$ *pure $\lambda$-terms in $n$ variables, modulo $\beta$.*

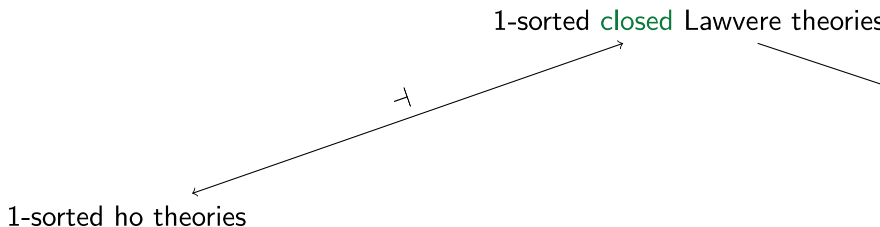
# An adjunction

We obtain a category of ho theories and a monadic adjunction:



# Updating the big picture

Restricting again to the 1-sorted case:

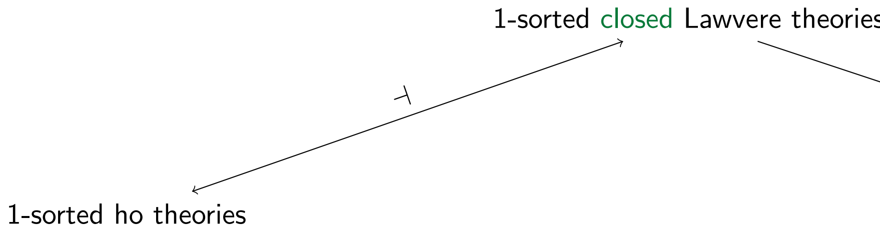



We still have the generated monad

$$\mathcal{T}_{\mathcal{L}}(X) = \int^{p \in \omega} X^p \times \mathcal{L}(p, 1).$$

# Updating the big picture

Restricting again to the 1-sorted case:




- How to get the closed structure from an arbitrary monad?
- Or by what should we replace Set to restore the triangle?
- Do not yet know.

# Equations vs. rewrite rules

- Programming languages have syntax with binding.
- But dynamics, not just equations.

$$\text{Equations } M = N \qquad \rightsquigarrow \qquad \text{Rewrite rules } M \to N.$$

**Definition**

A 2-signature $X$ consists of:

- a higher-order signature $(X_0, X_1)$,
- a set $X_2$ of rewrite rules, i.e., parallel terms

$$\Gamma \vdash M, N \colon A$$

in $\mathcal{L}_1(X_0, X_1)$.

# Rewriting with binding

- A 2-signature is formally the same as a ho theory.
- Before we used it to define a congruence on morphisms.
- Now we use it to generate a cartesian closed 2-category (2CCC).
- Thanks to a $2\lambda$-calculus $\mathcal{L}_2(X)$ in the style of Hilken.

**Theorem (in progress)**

*The $2\lambda$-calculus yields the free 2CCC on $X$.*

A categorical semantics for rewriting with binding.

# The pure $\lambda$-calculus

- Sort $t$.
- Operations

$$L \colon t^t \to t \qquad A \colon t \times t \to t.$$

- Rewrite

$$A(L(\lambda x.M), N) \to (\lambda x.M)N.$$

**Reductions**

Generated 2CCC $\approx$ Lévy category of reductions.

I.e., reductions up to permutation of independent rewrites.

# A non example: $\pi$

- Sorts $p$ and $n$ (processes and names).
- Operations

$$s\colon n \times n \times p \to p \qquad g\colon n \times p^n \to p \qquad z\colon 1 \to p$$

$$\nu\colon p^n \to p \qquad |\colon p \times p \to p$$

- Rewrites

$$x\colon p^n, y\colon p \vdash (\nu x) \mid y \leftrightarrow \nu\lambda a.(x(a) \mid y)\colon p \qquad \ldots$$

$$x\colon p^n, y\colon p, a, b\colon n \vdash s(a, b, y) \mid g(a, x) \to y \mid x(b)\colon p$$

- Non-example: illegitimate non-identity loops.
- Would require 2-theories, i.e., equations between rewrites.

## Module systems: core language

- Fix a set $a, b, \ldots \in L$ of labels.

$$
\begin{array}{lcll}
M & ::= & [E] \mid \ldots & \text{Module expressions} \\
E & ::= & M.a \mid \ldots & \text{Core expressions}
\end{array}
$$

- Common practice: abstract over part of the core language.
- Never properly formalised.
- Assume a 2-signature with sorts $t$ and $m$ and operations

$$
val \colon t \to m \qquad \pi_a \colon m \to t.
$$

No need to specify everything.

# Module systems: structures

$$
\begin{array}{rcll}
M & ::= & [E] \mid \{D\} \mid \dots & \text{Module expressions} \\
E & ::= & M.a \mid \dots & \text{Core expressions} \\
D & ::= & \emptyset \mid a \triangleright x = M; D & \text{Structure}
\end{array}
$$

- Sorts $t, m, d$.
- Operations as above plus:

$$cons_a \colon t \times d^t \to d \qquad nil \colon 1 \to d \qquad \{\cdot\} \colon d \to t.$$

- Then interpret $\{a \triangleright x = M; b \triangleright y = N\}$ as

$$\{cons_a(M, \lambda x. \; cons_b(N, \lambda y.nil))\}$$

# From synthesis to analysis

- Systematic definition of rewriting systems with binding
  - by a universal property,
  - defining a category of models at the same time.
- Future work: extension to 2-theories to deal with structural congruence.

**Beyond definition, analysis, cf. introduction**

- Code and compiler correctness.
- Behavioural equivalences.

# Towards analysis

**Thesis**

General understanding of such analyses will emerge from making the topology of programming languages explicit.

# Topology for syntax

**Main property of the free CCC over a signature**

Morphisms $p \longrightarrow 1 \cong$ terms with variables in $p$.

There are other structures satisfying this!

# Symmetric monoidal closed categories

- First example: the free SMCC.
- Girard, Trimble, then Hughes have shown its topological nature.
- Instead of finite products and exponentials: $\otimes$, $\multimap$, $I$.

# Symmetric monoidal closed categories

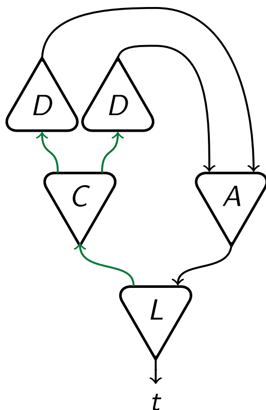
Need two sorts $t$ and $v$, with operations:

$$L\colon (v \multimap t) \to t \qquad A\colon t \otimes t \to t$$

$$D\colon v \to t \qquad C\colon v \to v \otimes v \qquad W\colon v \to I.$$

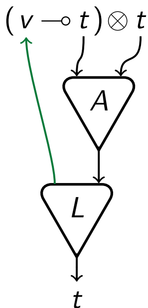
# Symmetric monoidal closed categories

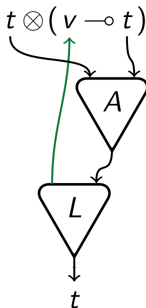
First example, $\lambda x.(x\ x)$:



# Symmetric monoidal closed categories

Example morphisms:




Both are $\lambda x.(\square_1\ \square_2)$, but with $x$ constrained differently.

# Results

- As mentioned: translation 2nd-order theory $\rightsquigarrow$ SMC theory.
- The sort $t$ becomes two:
  - $t$ for terms (linear),
  - $v$ for variables (non-linear).
- Translate an arity $2, 3 \to 1$ into

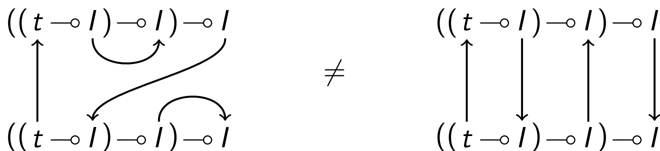
$$(v^{\otimes 2} \multimap t) \otimes (v^{\otimes 3} \multimap t) \to t.$$

- Terms with $p$ variables $\cong$ morphisms $v^{\otimes p} \to t$.
- Finer than ho theories: can handle linearity!
- Leads to a principled approach to Milner's bigraphs (Concur '09).

# Conclusions on SMCCs

- SMCCs bring topological reasoning to the world of syntax with binding.
- Finer than ho theories.
- But: there are some features of SMCCs we do not understand or use in terms of binding.

# Example: Trimble rewiring




- Different in the free SMCC.
- Equal in the free CCC.

# How about reversing the process?

- The Leinster-Weber theory of nerves extracts the algebraic structure from a specification in terms of “shapes”.
- Possible future work: use shapes and rely on Leinster-Weber to produce an algebraic structure?

# Topology for dynamics

- Using SMCCs brings topological reasoning to syntax with binding.
- No proposal yet to extend this to the dynamics.

# A sketch

**Principle**

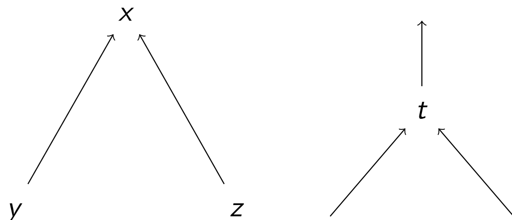
A language consists of:

- an intrinsic topology;
- a rule of the game.

- Programs come only later, as a syntax for constructing strategies.
- Similar to: labelled transition system with topology on vertices.

# Example: positions

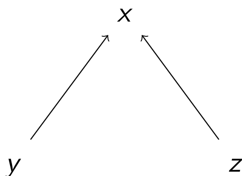
Directed (open) forests.




(I name vertices to keep track of them in the next slide.)

# Example: move

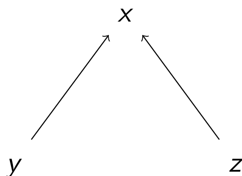
Imagine that in



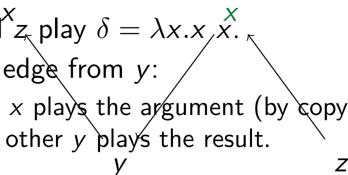

- $x$ plays $y\ z$,
- both $y$ and $z$ play $\delta = \lambda x.x\ x$.

# Example: move

Imagine that in




- $x$ plays $y\ z$,
- both $y$ and $z$ play $\delta = \lambda x.x\ x$.
- $x$ splits its edge from $y$:
  - on one $x$ plays the argument (by copycat-ing $z$),
  - on the other $y$ plays the result.



# The presheaves of strategies

- Positions.
- Morphisms 1: embeddings. Category $\mathcal{H}$.
- Restriction of a strategy to a subspace.
- Strategies form a presheaf $\mathcal{H}^{op} \longrightarrow \mathsf{Set}$.

Local strategies for a sheaf.

Amalgamation in the sheaf encompasses parallel composition.

# The presheaves of strategies

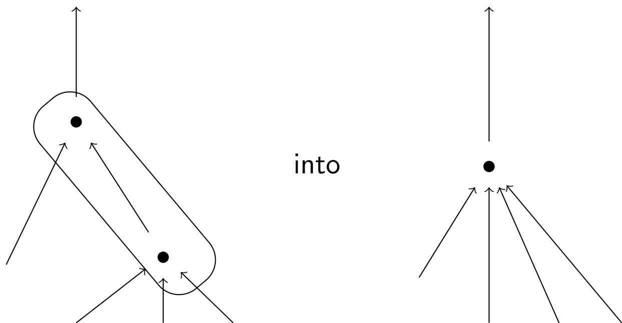
- Positions.
- Morphisms 2: plays. Category $\mathcal{P}$.
- Restriction of a strategy along a move: $m^*(\sigma) = \{p \mid mp \in \sigma\}$.
- Strategies form a functor $\mathcal{P} \to \mathrm{Set}$.

Taking as set of cones $\coprod_{\text{positions } x}$ moves to $X$,

Strategies form a model for the corresponding sketch.

# Hiding

- Hide a connected subposition




- Category $\mathcal{C}$.

Strategies form a functor $\mathcal{C} \to \mathrm{Set}$.

This encompasses game semantic hiding.

# In progress

A full abstraction result for a calculus of continuations.

# Conclusions

- Question: which universe of discourse for reasoning on programming languages?
- An algebraic approach using cartesian closed (2-)categories.
- More topological attempts under investigation.

Thanks for your attention.