Higher Multicategories and Programming Languages

Damiano Mazza
CNRS, LIPN, Université Sorbonne Paris Nord

Journée Opérades LAGA-LIPN
5 November 2020

Motivation

- There is a host of interesection type systems for various programming languages, characterizing termination properties.
- Empirical observation: these all arise by looking at the language in question under the lens of linear logic.
- Can this observation be made precise?
- **Theorem (∼Cook-Levin):** P *has polysize circuits.*
- The idea: computation is local and Boolean circuits may approximate arbitrary computations with a polynomial overhead.
- Can this idea be given a more conceptual form?
- It turns out the the language of operads/multicategories is of great help with these questions!

What's a Programming Language?

First example: Boolean circuits.
Draw it on the tablet!

What's a Programming Language?

- Expressions (or terms):

$$
\begin{array}{lll}
M, N ::= 0 \mid M+1 \mid M-1 & & \text{integer arithmetic} \\
\quad \mid \text{True} \mid \text{False} \mid M=0 \mid \text{if } P.M \text{ else } N & & \text{Booleans and conditional} \\
\quad \mid x \mid \text{def } f(x).M \mid M(N) & & \text{variables and functions}
\end{array}
$$

- Computation rules (or reduction rules):

$$
\begin{array}{rcl@{\qquad}rcl}
(M+1)-1 & \to & M & 0-1 & \to & 0 \\
0=0 & \to & \text{True} & (M+1)=0 & \to & \text{False}
\end{array}
$$

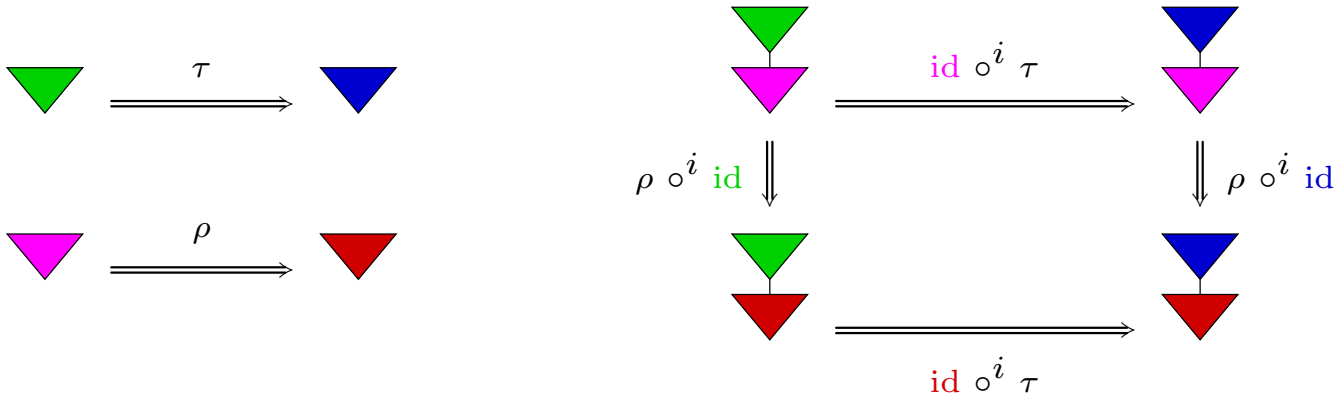
$$
\text{if } b.M \text{ else } N \quad \to \quad \begin{cases} M & \text{if } b = \text{True} \\ N & \text{if } b = \text{False} \end{cases}
$$

$$
(\text{def } f(x).M)(N) \quad \to \quad M[N/x][\text{def } f(x).M/f]
$$

- Substitution: $P[Q/y]$ is P in which every free occurrence of y is replaced by a copy of Q.

Programming Languages as 2-Operads

2-operad = symmetric (semi)multicategory enriched over categories



The multicategory may be *cartesian*, but not always (circuits, linear logic...).

Examples. Circuits are obviously a 2-operad. Also the above programming language:

0. one object;
1. multimorphisms of arity n: terms with free variables in $\{x_1, \ldots, x_n\}$;
2. 2-arrows $M \Rightarrow N$: reductions $M \to^* N$ modulo permutation equivalence;

operadic composition is substitution. In this case, the 2-operad is cartesian.

What's a Type System?

Something that looks like this:

- Types: $A, B ::= \mathsf{int} \mid \mathsf{bool} \mid A \to B$.
- Typing judgements: $\Gamma \vdash M : A$, with $\Gamma = x_1 : A_1, \ldots, x_n : A_n$.
- Typing rules:

$$\frac{}{\Gamma \vdash 0 : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash M + 1 : \mathsf{int}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash M - 1 : \mathsf{int}}$$

$$\frac{b \in \{\mathsf{True}, \mathsf{False}\}}{\Gamma \vdash b : \mathsf{bool}} \qquad \frac{\Gamma \vdash M : \mathsf{int}}{\Gamma \vdash M = 0 : \mathsf{bool}} \qquad \frac{\Gamma \vdash P : \mathsf{bool} \quad \Gamma \vdash M : A \quad \Gamma \vdash N : A}{\Gamma \vdash \mathsf{if}\ P.M\ \mathsf{else}\ N : A}$$

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{\Gamma, f : A \to B, x : A \vdash M : B}{\Gamma \vdash \mathsf{def}\ f(x).M : A \to B} \qquad \frac{\Gamma \vdash M : A \to B \quad \Gamma \vdash N : A}{\Gamma \vdash M(N) : B}$$

What's a Type System?

[A] *type system is a logical system comprising a set of rules that assigns a property called type to the various constructs of a computer program, such as variables, expressions, functions or modules. These types formalize and enforce the otherwise implicit categories the programmer uses for algebraic data types, data structures, or other components (e.g. "string", "array of float", "function returning boolean"). The main purpose of a type system is to reduce possibilities for bugs in computer programs.*

— Wikipedia

The fundamental purpose of a type system is to prevent the occurrence of execution errors during the running of a program.

— Luca Cardelli

A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

— Benjamin Pierce

What's a Type System?

[A] *type system is a logical system comprising a set of rules that assigns a property called type to the various constructs of a computer program, such as variables, expressions, functions or modules. These types formalize and enforce the otherwise implicit categories the programmer uses for algebraic data types, data structures, or other components (e.g. "string", "array of float", "function returning boolean"). The main purpose of a type system is to reduce possibilities for bugs in computer programs.*

— Wikipedia

The fundamental purpose of a type system is to prevent the occurrence of execution errors during the running of a program.

— Luca Cardelli

A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute.

— Benjamin Pierce

A type system is a functor. — Paul-André Melliès and Noam Zeilberger

"Church-style" Types

- Naive approach:

  type system = category

  A program M of type B with a parameter x of type A is an arrow

$$A \xrightarrow{M[x^A]^B} B$$

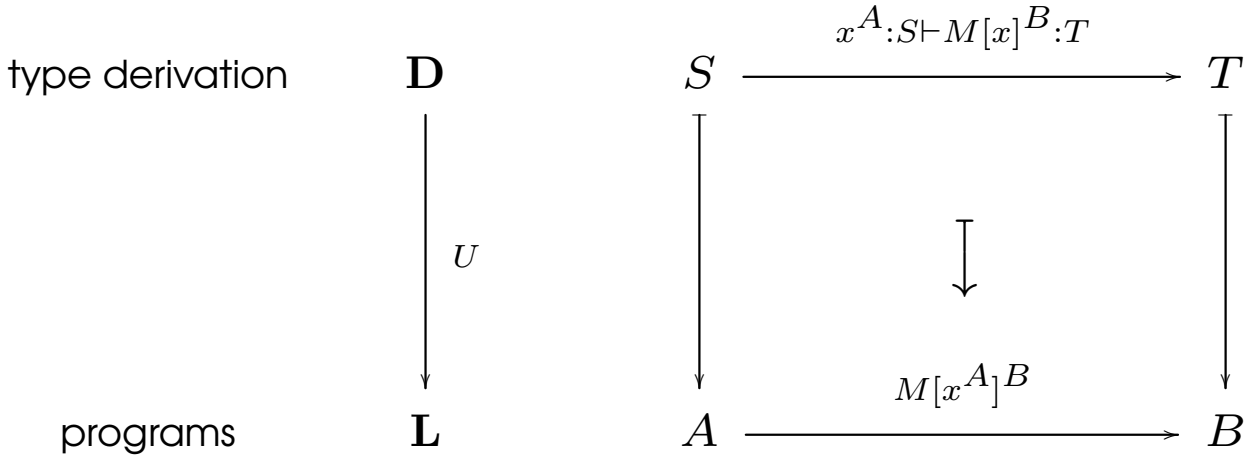
- Not general enough:

$$\frac{x : A \vdash M : \text{int} \qquad \text{int} \leq \text{float}}{x : A \vdash M : \text{float}}\ \text{subtyping}$$

  but an arrow in a category cannot have several targets...

Functors are Type Refinement Systems (Melliès and Zeilberger 2015)

- We may consider instead a functor



- This is a type refinement system:
  - $U(S) = A$ means that S *refines* A;
  - $U(S \xrightarrow{\delta} T) = A \xrightarrow{M} B$ means that δ is a type derivation for M;
  - $U(S \xrightarrow{\delta} T) = \mathrm{id}_A$ means that S is a subtype of T.

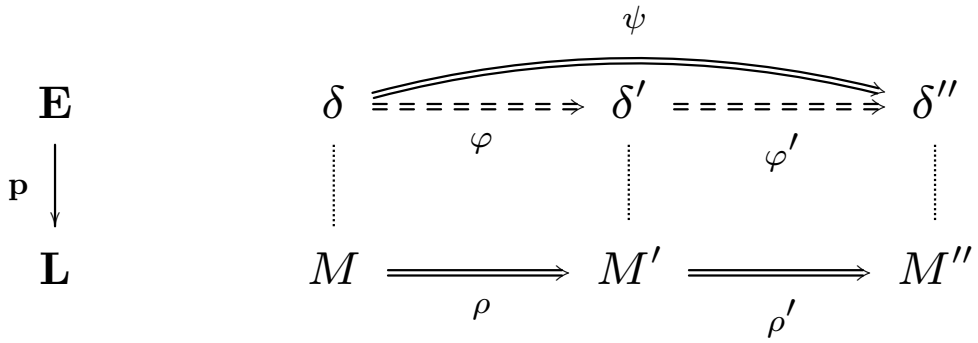
Example

- Let $\mathbf{Progs}$ be the following monoid (category with only one object):
  - elements: terms with at most a free variable x;
  - operation: $M \bullet N := M[N/x]$ (substitution); identity: x.
- Let $\mathbf{Ders}$ be the following category:
  - objects: types;
  - arrows $A \to B$: type derivations of $x : A \vdash M : B$;
  - composition $=$ "cut"; $\mathrm{id}_A = \overline{x : A \vdash x : A}$.
- The type system above corresponds to the functor

$$\begin{array}{ccc} \mathbf{Ders} & & \vdots \\ & & \Gamma \vdash M : A \\ \Big\downarrow U & & \Big\downarrow \\ \mathbf{Progs} & & M \end{array}$$

Back to 2-Operads

- Melliès and Zeilberger's viewpoint of course carries over to 2-operads.
- *Niefield fibrations* allow us to speak of subject reduction/expansion:

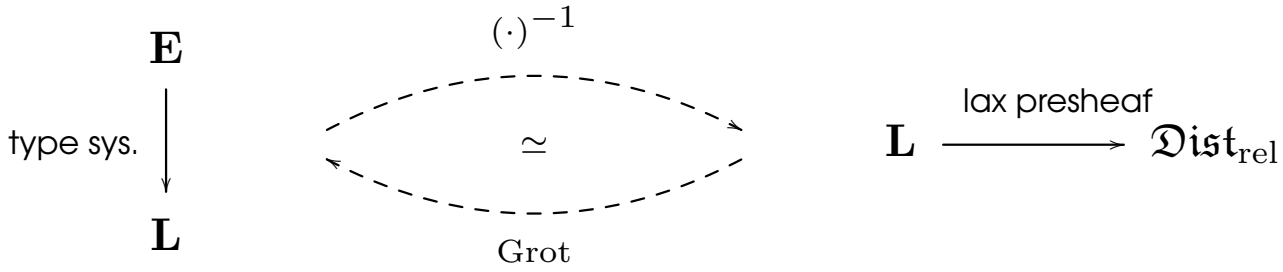


$\mathbf{p}(\psi) = \rho' \circ \rho$ implies

$\exists \varphi, \varphi'.\ \psi = \varphi' \circ \varphi$ and

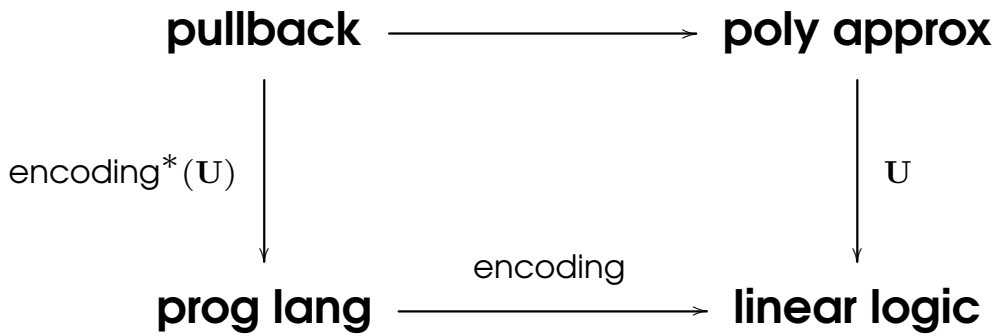
$\mathbf{p}(\varphi) = \rho,\ \mathbf{p}(\varphi') = \rho'$

Grothendieck construction:



Intersection Types Systems via "Change of Base"

- M., Pellissier, Vial 2018: intersection types systems arise by pulling back a "universal" system (Neifield fibration) $\mathbf{U}$ along encodings in linear logic:



- **Theorem:** if the encoding has certain properties, then the corresponding type system characterizes usual termination properties.
- $\mathbf{U}$ arises from the Grothendieck construction applied to an *approximation presheaf*. In the linear case, this is (a rigid version of) the *Taylor expansion* (Ehrhard and Regnier 2006, de Carvalho 2007).
- Allows constructing intersection types for concurrent languages (Dal Lago, de Visme, M., Yoshimizu 2019).

Church Meets Cook and Levin

Theorem. *A $O(f)$-time DTM induces Boolean circuits of size $O(\mathsf{poly}(f))$.*

PROOF.

- Poset on Boolean circuits. Evaluation is monotonic.
- Finite compatible suprema exist and $|C \sqcup C'| \leq \max(|C|, |C'|)$.
- Approx. relation circuits $\sqsubset$ programs $\Rightarrow$ intersection types system!
- The type system enjoys polynomial subject expansion.
- Let $M(x)$ be a program s.t., $\forall w \in \{0,1\}^*$, $M(w) \to^{f(|w|)} b_w \in \{0,1\}$. Then

$$\vdash b_w : \mathsf{Bool} \overset{\text{subj.exp.}}{\Longrightarrow} \vdash M(w) : \mathsf{Bool} \overset{\text{synt.dir.}}{\Longrightarrow} C_w :: x : \mathsf{Bool}^{\otimes k} \vdash M : \mathsf{Bool}$$

  and $|C_w| = O(\mathsf{poly}(f(|w|)))$. By continuity, $C_w(w) \to^* b_w$.
- Let

$$C_n := \bigsqcup_{|w|=n} C_w. \qquad \text{Monotonicity} \Rightarrow C_n(w) \to^* b_w,$$

  so C_n is a $O(\mathsf{poly}(f))$-size family of circuits deciding the language of M. $\square$

Classical Lambda-Calculus in Modern Dress (Hyland 2017)

- The classical notion of λ-*theory* is presented as a cartesian operad (some would say *abstract clone*, or *Lawvere theory*) with a *semi-closed structure*

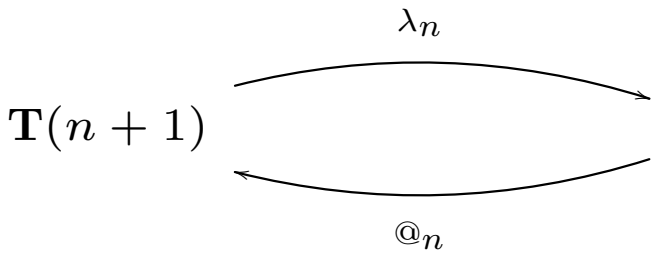
$$\mathbf{T}(n+1) \underset{@_n}{\overset{\lambda_n}{\rightleftarrows}} \mathbf{T}(n) \qquad \text{s.t.} \qquad @_n \circ \lambda_n = \mathrm{id}_{\mathbf{T}(n+1)}$$

- The untyped λ-calculus modulo β-equivalence is the initial λ-theory Λ_β.
- **Fundamental Theorem:** there is an equivalence of categories

$$\lambda\textbf{-theories} \;\simeq\; \Lambda_\beta\textbf{-algebras}.$$

The Second Dimension

- One may define a semi-closed structure on a cartesian 2-operad:



s.t. $\quad @_n \circ \lambda_n \Rightarrow \mathrm{id}_{\mathbf{T}(n+1)}$

- The "initial" such 2-operad Λ is the untyped λ-calculus with β-reduction:

$$(\lambda x.M)x \to M$$

- Λ-algebras (in $\mathbf{Cat}$) are "*higher denotational semantics*". Is this useful?

Recap

- Prog langs seen as small 2-operads ($\mathbf{Cat}$-enriched symmetric multicategories).
  - Not general enough...
  - Rencently, better perspective (using monads) by Hirschowitz and Lafont.
- Morphisms of 2-operads

$$\mathbf{f} : \mathbf{A} \longrightarrow \mathbf{B}$$

  model several concepts from the theory of programming languages:
  - $\mathbf{A}, \mathbf{B}$ small: encoding/compilation of $\mathbf{A}$ in $\mathbf{B}$.
  - $\mathbf{A}, \mathbf{B}$ small, $\mathbf{f}$ Neifield fibration: type system for $\mathbf{B}$.
  - $\mathbf{A}$ small, $\mathbf{B}$ discrete homsets: denotational semantics for $\mathbf{A}$.
  - $\mathbf{A}$ small, $\mathbf{B}$ groupoid homsets: 2-semantics (?) for $\mathbf{A}$ (Galal, Olimpieri...).

The dimensional ladder

5. ...
4. residual equivalence (ask P.-A. Melliès)
3. standardization
2. cut-elimination/execution
1. proofs/programs
0. formulas/types

- A "kōan": what's the Curry-Howard isomorphism an isomorphism of?
- Semi-closed structure on a symmetric $(\infty, 2)$-$\mathbf{Cat}$-multicategory?
- The initial such thing should be the simply-typed λ-calculus, with all of its higher arrows/equivalences being "justified" by the structure... (Finster started doing this for the linear case).